

SIEMENS EDA

Calibre® Verification User's Manual

Software Version 2021.2
Document Revision 14

SIEMENS

Unpublished work. © 2021 Siemens

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software, Inc., its subsidiaries or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with Siemens. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This document is for information and instruction purposes. Siemens 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 Siemens to determine whether any changes have been made. Siemens disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' **End User License Agreement** may be viewed at: www.plm.automation.siemens.com/global/en/legal/online-terms/index.html.

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 Siemens whatsoever.

TRADEMARKS: The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: support.sw.siemens.com

Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Revision History

Revision	Changes	Status/ Date
14	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released April 2021
13	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released January 2021
12	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released October 2020
11	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released July 2020

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Siemens EDA documentation source. For specific topic authors, contact the Siemens Digital Industries Software documentation department.

Revision History: Released documents maintain a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation which are available on <https://support.sw.siemens.com/>.

Table of Contents

Revision History

Chapter 1		
Overview		25
Calibre nmDRC and Calibre nmDRC-H		25
Layout Utilities		26
Calibre nmLVS and Calibre nmLVS-H		26
Related Calibre Verification Tools		27
Syntax Conventions		28
Chapter 2		
Setup and Tool Invocation		31
Required Input Files		32
Rule File		32
Layout Databases		33
Source Databases		34
Calibre nmDRC and Calibre nmDRC-H Command Line		36
Calibre nmLVS and Calibre nmLVS-H Command Line		45
Calibre nmLVS Reconnaissance Command Line		66
Calibre CB		76
Calibre Version Information		76
Chapter 3		
Layout Processing Concepts		77
Layers		78
Original Layers		78
Layer Definitions		80
Derived Polygon Layers		82
Derived Edge Layers		83
Derived Error Layers		84
Unmerged Polygon Layers		86
Layer Type Summary		87
Layer Operations		89
Layer Constructors and Selectors		89
Node-Preserving Layer Operations		90
Layer of Origin		92
Hierarchical Database Construction		93
Hierarchy-Specific Statements		94
Layout Input Control		96
Area-Based Filtering		96
Cell Exclusion		96
Cell Renaming		97

Duplicate Cell Processing	97
Database Pre-Merging	97
Input Layout Database Magnification	98
Wildcards in Layout Primary	100
Layout Read Error Tolerance Settings	100
Layout Database End Segment Warning	100
Layout Property Handling	101
Supported Geometric Layout Formats	102
GDSII Layout Format	102
OASIS Layout Format	104
Third-Party Layout Formats	104
ASCII Layout Format	107
Chapter 4	
Calibre nmDRC Concepts	109
Data Flow in Calibre nmDRC	109
Invocation	110
Calibre nmDRC Specification Statements	111
Rule Check Statements	112
Rule Comments	113
Check Text	114
Rule Check Selection	116
Rule Check Selection and Mask Results Databases	116
Rule Check Result Limits	117
Control of Empty Rule Checks	118
Concurrency	118
Result Redundancy Elimination	119
Layer Operation Scheduling	119
Maximizing DRC Capacity and Minimizing Run Time	120
Conjunctive Checks	120
Efficient Width and Spacing Checks	121
Polygon Segmentation	124
Rule File Compilation	126
Calibre nmDRC Exit Values	126
Advanced Calibre nmDRC Concepts	128
Wide Metal Spacing Rules	128
Advanced DRC Toolset	135
Introduction to DFM Property	136
Introduction to DFM Function	139
Introduction to DFM RDB	140
Introduction to DFM Copy and DFM OR	141
Chapter 5	
Hierarchical Calibre nmDRC	143
Calibre nmDRC-H Results Data Storage	143
Flat Versus Hierarchical Results Presentation	145
Multithreaded Modes of Operation	146
Hierarchical Operation Efficiency	147

Table of Contents

Flat Instantiations	148
Path Length Variances	148
Layer Area Printing	150
Text Objects in Mask Results Output	150
Calibre nmDRC-H Use of Hcells	151
Chapter 6	
Dimensional Check Operations	153
Primary Dimensional Check Operations	153
Secondary Keywords	155
Edge Measurement	156
Measurement Region Construction	156
Metrics	159
Special Considerations for the Opposite Metric	160
Advanced Metrics for Specialized Applications	163
Opposite Symmetric Metrics	164
Unidirectional Metrics	166
Square Orthogonal Metric	168
Edge Cluster Generation	171
Trivial Edges	172
Four-Edge Output Cluster	173
Point-to-Point Measurement Output	174
Interval Constraints for Output Suppression	175
Appropriateness Criteria	177
Edge Shielding	179
Intersection Criteria	181
Edge Breaking	182
Polygon Containment Criteria	183
Output Modes	185
Edge-Directed Output	186
Polygon-Directed Output	186
Skew Edge and Sliver Polygon Handling	189
False Measurement Reduction	189
Measurement Tolerances	191
Chapter 7	
Specialized Calibre nmDRC Applications	193
Mask DRC Results	194
Attribute Specification	194
Mapping Algorithm for Output	195
AREF Output	198
AREF Structures and Fill Cells	198
AUTOREF Output	199
Original Shape Flagging	199
Original Geometry Snapping	200
Soft Connection Checks	201
Disk-Based Layers	203

Chapter 8

Calibre nmDRC Results.....	205
Session Transcripts	206
Header Information.....	206
Rule File Compilation.....	206
Layout Data Input.....	208
Limiting Text Object Output	211
Hierarchical and Flat Counts	211
Results Database Initialization Section.....	211
Executive Process	213
Time Statistics	213
Layer Statistics in Flat Processing	213
Layer Statistics in Hierarchical Processing	215
LVHEAP Statistics.....	217
Run Summary	217
Transcript Features Based Upon Run Mode.....	221
MT and MTflex Modes With Hyperscaling.....	221
Unique Transcript Features of MTflex Mode With Hyperscaling.....	226
MT Mode Without Hyperscaling	231
MTflex Mode Without Hyperscaling.....	232
DRC Results Database.....	234
ASCII DRC Results Database Format	236
RDBs	237
Cell Name and Database Precision.....	237
Rule Check Name, Result Count, and Execution Time	237
Check Text Report	237
Check Text for RDBs	238
Check Text for PRINT Files.....	239
Calibre nmDRC Result Listing	239
DRC Cell Name Results.....	240
Properties in ASCII DRC Results Databases.....	241
GDSII DRC Results Database Format	242
OASIS DRC Results Database Format.....	243
Result Count Limits	245
Hierarchical Calibre nmDRC Results Database.....	246
DRC Summary Report.....	247

Chapter 9

Connectivity Extraction	249
Mask Connectivity Extraction	249
Connectivity and Rule File Compilation	250
Electrical Net Recognition.....	252
Use of Text in Calibre Applications	258
Net Name Specification	258
Logical Information on Merged Layers	260
Net Label Attachment.....	261
Open and Short Circuits	265
Incremental Connectivity and Antenna Checks.....	266

Table of Contents

Virtual Connections in an Incremental Connect Flow	272
Incremental Connectivity and Runtime Efficiency	272
Suppression of Connectivity Extraction Warnings for Incremental Connect	273
Disconnect Operation	274
Connectivity Warnings in Calibre nmDRC	274
LVS Circuit Extraction	275
Hyperscaling for LVS Connectivity Extraction	276
Hcells and Circuit Extraction	276
Hierarchy Modification Suppression in LVS Connectivity Extraction	277
Selection of Extracted Netlist Names	278
Extraction of Subcircuits	279
Hierarchical Treatment of Net Labels	280
Netlisting of Ports and Pins	282
Hierarchical Processing of Port Text and Polygon Objects	283
Unattached Ports	285
Propagation of Power and Ground Names from Lower-Level Ports	285
Omission of Floating Pins	285
Virtual Connect Statements	286
LVS Connectivity Extraction Messages	288
Detection of Soft Connections	289
LVS Softchk Debug Method	290
LVS Softchk Results Database File Format	293
Report Soft Connections Using Contact Counts	294
lvs::softchk	298
Short Isolation	300

Chapter 10 Device Recognition 307

Device Recognition Overview	307
Concepts and Terminology	308
Recognition Logic	310
Layer Relations	310
Pin Relations	311
Fill-In Algorithm	313
Ill-Formed Devices	314
Device Signatures	315
DFM Property in ADP Device Recognition	318
Bad Device Reporting	320
Hierarchical Device Recognition	322
Property Computation	325
Default Property Computations	326
User-Defined Property Computation	330
Comment-Coded Properties for C and Q Elements	330
Units of Measurement	331
Property Computation Structure	332
Coding Efficiency Considerations	334
Property Computation Debugging	341
Debug Example	341

Hierarchical Run Considerations	343
Debug Statement Placement	344
Debug Output	344
Chapter 11	
Electrical Rule Checks	351
ERC Statements and Operations	351
Execution of ERC Operations in LVS	354
ERC Licensing	355
Rule Check Selection in LVS	356
Abort an LVS Run Due to ERC Errors	356
Case Sensitivity of Supply Net Names	356
User-Defined Devices in Path Check Operations	357
ERC Results Database	359
ERC Auxiliary Files	359
ERC Summary Report	361
Trivial Pin Removal Behavior	361
ERC Examples	363
ERC Operations in Calibre nmDRC	364
Chapter 12	
LVS Circuit Comparison	365
Data Flow in LVS	365
LVS Comparison	367
Initial Correspondence Points	369
Component Types	370
Naming Conventions	372
Instance Pins and Pin Names	372
Net and Instance Names	373
Ports and Port Names	374
Power and Ground Nets	374
User-Given Names	374
Case-Sensitive Handling of Names	375
Built-In Device Types	379
Capacitors	380
Diodes	382
MOS Transistors	383
Resistors	386
Bipolar Transistors	387
JFET Transistors	389
Inductors	390
Voltage Sources	391
X+ Devices	392
MS and MF Schematic Devices	392
Methods for Matching Circuit Elements	393
Circuit Comparison Results	394
Ambiguity Resolution	394
Device Reduction	396

Table of Contents

Parallel and Series Device Reduction	397
Generic Device Reduction	398
Parallel MOS Transistor Reduction	398
Series MOS Transistor Reduction	400
Semi-Series MOS Transistor Reduction	402
Split Gate Reduction	405
Series-Parallel Split Gate Reduction	407
Semi-Split Gate Reduction	409
Input Order Considerations	410
Error Tolerances	410
Mixed Component Types	410
Ambiguous Topologies and Split Gate Reduction	411
Parallel Capacitor Reduction	415
Series Capacitor Reduction	416
Parallel Resistor Reduction	417
Series Resistor Reduction	419
Parallel Diode Reduction	420
Parallel Bipolar Transistor Reduction	421
Missing and Unknown Property Values	423
Case Comparison of String Properties	423
Device Reduction Programs	424
Tolerance in Device Reduction	424
Device Filtering	428
Filtering Unused MOS Transistors	428
Filtering Unused Bipolar Transistors	429
Nets	430
Usage of Power and Ground Nets	430
Isolated Nets	430
Passthrough Nets	433
Feedthrough Nets	437
Trivial Ports, Pins, and Nets	438
Logic Gate Recognition	439
Tolerance-Based Logic Gate Recognition	440
Effects of Logic Injection	440
Recognition Processes	441
Regular CMOS Gates	442
Regular NMOS Gates	452
LDD Gates	457
X+ Transistors	461
Device and Pin Swapping Checks in Logic Gates	461
Pin Swapping	462
Device Property Tracing	467
Built-In Property Classification	467
Built-In W/L Partner Properties	468
Device Count Comparison After Reduction	469

Chapter 13	
Hierarchical LVS	473
Hcells	474
Hcell Correspondence	475
Hcells and Circuit Comparison	479
Calibre LVS Hcell Report Format	483
Hierarchy Preservation by Layer Operations	491
Cell Pushdown	495
Hierarchical LVS Comparison	496
Hierarchical Pins	496
Trivial Pin Swappability	496
LVS Box Cells	497
High-Short Resolution	498
SRAM Bit-Cell Recognition	501
Parameterized Cells	503
Logic Injection	505
Injected Components and the LVS Report	507
Injected Component Identification	507
Injected Component Instance Pin Identification	508
Missing Injected Instance Discrepancy	508
Unmatched Injected Instance	509
Logic Injection and Gate Recognition	509
Internal Net Matching	510
Logic Injection and Pin Swappability	510
Injected Component Naming Conventions	511
Bit Structure Core	512
Bit Structure	513
Bit Rows	515
Inverters	517
Inverter Chains	518
Series Gates	519
Parallel Gates	521
NAND and NOR Gates	523
Multiplexer Structure	525
XOR and XNOR Gates	527
Transmission Gate Multiplexers	532
Register File Bit	533
Chapter 14	
LVS Results	537
LVS Transcript	538
Hierarchical Circuit Extraction Transcript	539
Circuit Comparison Transcript	543
Hierarchical Comparison with Hyperscaling Transcript	543
Circuit Extraction Report	545
ERC, Short Isolation, and Softchk Results	546
LVS Summary Report	547
LVS Report	549

Table of Contents

Overall Report Structure	549
Analysis of the LVS Report	558
Errors and Warnings	560
Overall Comparison Results	562
Primary Messages	562
Secondary Messages	563
Overall SPICE Syntax Check Results	570
LVS Report Listing Conventions	570
LVS Discrepancy Types	575
Absolute Trace Property Tolerances Out of Range	584
Component Types with Non-Identical Signal Pins	585
Detailed Instance Connections	587
Errors in Names Given for Power/Ground Nets	588
Hierarchical Cells Forming a Cycle	588
Incorrect Substrate Connections	589
Input Errors	590
Instances of Cells With Non-Floating Extra Pins	591
Name Errors	594
Property Errors	595
Unmatched Objects	596
Information and Warnings	599
Detailed Error Analysis Reporting	603
Bad Instance Connections Error Detailed Reporting	604
Bad Instance Pair Error Detailed Reporting	604
Circular Connection Error Detailed Reporting	605
Cross-Connect and Pin-Swap Errors Detailed Reporting	606
Cross-Connect Error Detailed Reporting	608
Logic Gate Bulk Pin Error Detailed Reporting	608
Logic Gate Supply Error Detailed Reporting	609
Net Connection Error Detailed Reporting	610
Open Circuit Detailed Reporting	611
Pin Connection and Open Circuit Error Detailed Reporting	613
Pin Connection and Short Circuit Error Detailed Reporting	614
Pin Connection Error Detailed Reporting	615
Pin Swap Error Detailed Reporting	615
Short Circuit Detailed Reporting	616
SPICE Syntax Check Report	619
SVDB Cross-Reference Files	623
SVDB Header	623
Flat Instance Cross-Reference File	624
Flat Net Cross-Reference File	626
Hierarchical Instance and Net Cross-Reference Files	627
Source and Layout Placement Hierarchy Files	628
Binary Polygon Format (BPF) Database	629
Mask Results Database	631

Chapter 15	
SPICE Format	633
SPICE Syntax Conventions	634
Case Sensitivity	635
Characters Allowed	635
Numeric Value Types	636
Arithmetic Expressions	638
Comments	639
Comment-Coded Extensions	639
String Parameters	640
Inline Parameters in Subcircuit Calls	641
X Instantiated Devices	642
Dollar Signs in Cell Names	644
\$D Parameters	644
Cell Statistics	644
SPICE Syntax Checking	645
Control Statements	647
*.BUSDELIMITER	648
*.CALIBRE	651
*.CAPA	652
*.CONNECT	653
*.DIODE	656
.ENDL	657
.EDNS or .EOM	658
*.EQUIV	659
.GLOBAL	661
.INCLUDE	663
*.LDD	665
.LIB	666
*.MEGA	668
.OPTION SCALE	670
.PARAM	672
*.SEEDPROM	673
*.STRIP_TYPE_CHAR	674
*.XPINS	675
Ignored Control Statements	676
Element Statements	677
Capacitor Element	677
Junction Diode Element	680
JFET Element	681
Inductor Element	683
MOSFET Element	685
BJT Element	690
Resistor Element	692
Voltage Source Element	694
Subcircuit Definitions	695
Subcircuit Calls	699
Floating Pins in Subcircuit Calls	704

Table of Contents

Block-Level Chip Assembly Using Scoped SPICE Subcircuits	705
Multiplier (m) Parameters.....	707
Eldo Format Y Element	708
Chapter 16	
V2LVS	709
V2LVS Overview	709
Basic V2LVS Usage	712
Modules	714
Named Port Declarations in a Verilog Module	715
Port Selection, Bit Selection, and Array Mapping	715
Undefined Modules.....	716
User-Defined Primitives	718
Module Instantiations	718
Call Conventions.....	718
Bit Expressions	720
Unnamed Concatenation Expressions in Declarations	721
Unconnected Pins	722
Declarations	724
Port Declarations.....	724
Net Types	727
Power and Ground Connections.....	727
Power and Ground Signal Overrides	731
Addition of Pins	733
Management of Bulk Pin Connections	740
Net Assignment.....	742
Primitive Instances.....	742
Behavioral Statements	744
Specify Block.....	745
Expressions	745
Compiler Directives.....	749
Library Files.....	750
V2LVS Used With SPICE Library Files	751
Range Mode	752
Pin Mode	753
Enable Subcircuit Scoping	754
*.BUSDELIMITER Statement Handling.....	756
V2LVS Used Without Verilog or SPICE Library Files	758
Instances of Undeclared SPICE Primitive Modules with Named Ports.....	758
Instances of Undeclared SPICE Primitive Modules with Ordered Ports	759
Potential Errors When Using V2LVS Without Verilog Libraries	759
Simulation Output Generation.....	760
Calibre xRC Source Template File	762
Collecting SPICE .INCLUDE Netlists into a Single Netlist	763
Creating LVS Box Subcircuits	765
V2LVS Tcl Interface	766
Tcl Interface and Command Line Option Correspondence	766
Tcl Basics	770

Tcl Interface Command Encyclopedia	771
v2lvs::add_actual_port	772
v2lvs::add_formal_port	776
v2lvs::add_pin	779
v2lvs::combine_interface_info	781
v2lvs::connect_missing_ports	784
v2lvs::create_pex_template	786
v2lvs::do_source	787
v2lvs::find_module	788
v2lvs::generate_empty_subckts	789
v2lvs::generate_ordered_pins	790
v2lvs::get_includes	793
v2lvs::get_ports	794
v2lvs::help	795
v2lvs::include_dir	796
v2lvs::insert_port_instance	797
v2lvs::load_spice	799
v2lvs::load_verilog	802
v2lvs::number_unconnected_pins	804
v2lvs::override_assign	805
v2lvs::override_globals	807
v2lvs::preserve_back_slash	809
v2lvs::rename_duplicate_cells	810
v2lvs::set_array_delimiter	815
v2lvs::set_config_file	816
v2lvs::set_includes	817
v2lvs::set_prefix	818
v2lvs::set_unnamed_pin_prefix	819
v2lvs::set_verbose_level	820
v2lvs::set_warning_level	821
v2lvs::show_properties	822
v2lvs::substitute_chars	823
v2lvs::verilog_version	824
v2lvs::warning_as_error	825
v2lvs::write_file_info	826
v2lvs::write_output	827
v2lvs::write_pin_info	831
Error Handling for the V2LVS Tcl Interface	832
Supported Verilog Syntax	833
V2LVS Command Line Syntax	836
V2LVS Errors and Warnings	843
Chapter 17	
E2LVS	853
E2LVS Overview	853
E2LVS Command Line Syntax	855
Differences Between EDIF and SPICE	858
Identifiers	858

Table of Contents

Name Scope	858
Arrays and Bundles.	859
Untranslated EDIF Syntax	860
EDIF Cell Names Versus SPICE Subcircuit Names	861
Rename and Name Conflict	862
Translated EDIF Syntax Elements.	863
cell.	863
edif	864
instance	865
joined	866
net and netBundle	866
port and portBundle	867
rename.	868
status	869
Netlist Example	869

Chapter 18

Rule File Analyzer **875**

calibre -svrf	876
Rule File Analyzer Commands	878
CHECK.	880
CONN	881
DEV	882
FILE	884
FREEZE	885
HELP	888
LAYER	889
LIST ALIASES	891
LIST CHECKS	892
LIST CONNECTS	893
LIST DEVICES	894
LIST LAYERS	895
QUIT.	896

Index

Third-Party Information

List of Figures

Figure 3-1. Edge-Polygon Relationship.....	77
Figure 3-2. Layer Types.....	78
Figure 3-3. Layer Types and Data Flow in the DRC System	88
Figure 3-4. Hierarchical AND Operation	94
Figure 4-1. Calibre nmDRC Data Flow.....	110
Figure 4-2. Erosion Effects	129
Figure 4-3. Case 1	130
Figure 4-4. Case 2	130
Figure 4-5. Case 3	130
Figure 4-6. Traditional DRC Versus eqDRC.....	131
Figure 4-7. Wide Metal Spacing Output in Calibre DESIGNrev	133
Figure 4-8. DRC Process Design Rule Manual.....	134
Figure 4-9. eqDRC Process Design Rule Manual	134
Figure 4-10. Polygon Area to Perimeter Ratio	138
Figure 4-11. Number of Contacts in Source-Drain Regions	139
Figure 5-1. Hierarchical Error Suppression.....	144
Figure 5-2. Hierarchical Versus Flat Results.....	149
Figure 6-1. Measured Edges in the Dimensional Check Operations.....	154
Figure 6-2. Euclidean Generation of Output Edges.....	156
Figure 6-3. Measurement Region Formation.....	157
Figure 6-4. Edge Trimming Prior to Region Construction	158
Figure 6-5. Measurement Regions for Basic Metrics	160
Figure 6-6. Commutative Measurement	161
Figure 6-7. Non-Commutative Measurement	162
Figure 6-8. Output Adjustments for the Opposite Metric	162
Figure 6-9. OPPOSITE1	168
Figure 6-10. Square Orthogonal Measurement Region.....	169
Figure 6-11. SQUARE ORTHOGONAL Measurement Region (45-degree).....	169
Figure 6-12. SQUARE ORTHOGONAL Measurement Region (Skew Edge).....	170
Figure 6-13. Three-Edge Output Cluster	171
Figure 6-14. Trivial Edge Generation	173
Figure 6-15. Four-Edge Output Cluster.....	174
Figure 6-16. Point-to-Point Trivial Edge Generation	174
Figure 6-17. Suppressing Redundant Errors (part 1).....	176
Figure 6-18. Suppressing Redundant Errors (part 2).....	177
Figure 6-19. Edge Inside and Outside Planes	178
Figure 6-20. Appropriate Angles Between the Outsides of Edges	179
Figure 6-21. Full-edge Shielding.....	180
Figure 6-22. Partial-Edge Shielding	181
Figure 6-23. Edge Breaking in a Two-Layer Dimensional Check Operation	182

Figure 6-24. Measurements Prevented by Polygon Containment	183
Figure 6-25. Error Reduction Using Polygon-Directed Output.	188
Figure 6-26. False Notch Measurement (single polygon)	189
Figure 6-27. False Notch Measurement (two polygons)	190
Figure 7-1. Mapping Algorithm for Multiplexing of DRC Rule Check Output	197
Figure 8-1. ASCII DRC Results Database Example	236
Figure 8-2. Description of Cell Name CELL SPACE XFORM Property	240
Figure 9-1. Connected Shapes on a Single Layer	252
Figure 9-2. Connected Shapes from Different Layers.	253
Figure 9-3. Polygons Connected By Contact.	253
Figure 9-4. Shielding	254
Figure 9-5. Sconnect	256
Figure 9-6. Sconnect Operation.	257
Figure 9-7. Example of Virtual Connect Box Name	287
Figure 10-1. Gate with Shorted S/D Pins	313
Figure 10-2. Device Signature	316
Figure 10-3. Efficient Function Choice	338
Figure 12-1. Single-Step LVS Data Flow	366
Figure 12-2. Two-Step Calibre nmLVS-H Data Flow	367
Figure 12-3. Parallel and Series Device Reduction	397
Figure 12-4. Parallel MOS Transistor Reduction	398
Figure 12-5. Effective AS/AD Computation With Pin Swapping.	399
Figure 12-6. Series MOS Transistor Reduction.	401
Figure 12-7. Reduce Semi-Series MOS.	403
Figure 12-8. Split Gate Reduction	406
Figure 12-9. Series-Parallel Split Gate Reduction.	408
Figure 12-10. Reduce Split Gates Example.	409
Figure 12-11. SAME ORDER Option.	410
Figure 12-12. Layout Schematic	412
Figure 12-13. Source Schematic	412
Figure 12-14. Alternate Source Schematic	412
Figure 12-15. Layout with Property-Value Symmetry	413
Figure 12-16. Source with Property-Value Symmetry	413
Figure 12-17. Input Data with Property-Value Symmetry	414
Figure 12-18. Parallel Capacitor Reduction	415
Figure 12-19. Series Capacitor Reduction.	417
Figure 12-20. Parallel Resistor Reduction.	418
Figure 12-21. Series Resistor Reduction	419
Figure 12-22. Parallel Diode Reduction	420
Figure 12-23. Parallel Bipolar Transistor Reduction.	422
Figure 12-24. Unused MOS Transistors	429
Figure 12-25. Unused Bipolar Transistor	429
Figure 12-26. LVS Logic Gate Selection	442
Figure 12-27. INV	442
Figure 12-28. NAND _n	443

List of Figures

Figure 12-29. NOR n	443
Figure 12-30. AOI_3_2	444
Figure 12-31. OAI_3_2	445
Figure 12-32. SUP n	446
Figure 12-33. SDW n	446
Figure 12-34. SPUP_3_2	447
Figure 12-35. SPDW_3_2	448
Figure 12-36. SMP n , SMN n , SM n , S(TTT) n Series	449
Figure 12-37. SPMP_3_2, SPMN_3_2, SPM_3_2, SP(TTT)_3_2	450
Figure 12-38. SPMP((2+1+1)*1) Structure	451
Figure 12-39. SPMN(((3*1)+2)*(2+2)) Structure	452
Figure 12-40. INV	453
Figure 12-41. NAND n	453
Figure 12-42. NOR n	454
Figure 12-43. OAI_3_2	454
Figure 12-44. SDW n	455
Figure 12-45. SPDW_3_2	455
Figure 12-46. SMD n , SME n , S(TTT) n	456
Figure 12-47. SPMD_3_2, SPME_3_2, SP(TTT)_3_2	457
Figure 12-48. LDD AOI_3_2	459
Figure 12-49. SLDDP3	460
Figure 12-50. SPMN-LDDN(D)_3_1	461
Figure 13-1. Trivial Pin Swappability	497
Figure 13-2. SRAM Bit-Cell	502
Figure 13-3. Carrying Pin Swappability Up the Hierarchy	503
Figure 13-4. _bitcorev Structure	512
Figure 13-5. _bitv Structure	513
Figure 13-6. _bitrow2v Structure	515
Figure 13-7. _inv Structures	517
Figure 13-8. _invx3v Structure	518
Figure 13-9. _smn2v, _smp2v, _sup2v, _sdw2v Gates	520
Figure 13-10. _pmn2v, _pmp2v, _pup2v, _pdw2v Gates	522
Figure 13-11. _nand2v and _nor2v Gates	524
Figure 13-12. _mx2v Gate Multiplexer	526
Figure 13-13. _xra2v Multiplexer	527
Figure 13-14. _xr2v Multiplexer	527
Figure 13-15. _xor2v and _xnori2 Gate	529
Figure 13-16. _xnor2v and xor2	530
Figure 13-17. _tgmb Gate Multiplexer	532
Figure 13-18. _bitrfv Structure	534
Figure 14-1. Split Gate Property Ratio Error	584
Figure 14-2. Bad Instance Connections Error	604
Figure 14-3. Bad Instance Pair Error	605
Figure 14-4. Circular Connection Error	606
Figure 14-5. Cross-Connect and Pin Swap Errors	607

Figure 14-6. Cross-Connected Instances Error	608
Figure 14-7. Logic Gate Supply Error.....	609
Figure 14-8. Net Connection Error	610
Figure 14-9. Open Circuit Error	611
Figure 14-10. Pin Connection and Open Circuit Error	613
Figure 14-11. Pin Connection and Short Circuit Error	614
Figure 14-12. Pin Connection Error	615
Figure 14-13. Pin Swap Error	616
Figure 14-14. Short Circuit Error	616
Figure 16-1. Typical V2LVS Inputs and Output.....	711
Figure 17-1. E2LVS Flow	854

List of Tables

Table 1-1. Syntax Conventions	28
Table 2-1. Required DRC Rule File Statements	33
Table 2-2. Required LVS Rule File Statements	33
Table 2-3. Layout Database Formats	33
Table 2-4. Source Database Formats	34
Table 3-1. Support of Unmerged Derived Polygon Layers	86
Table 3-2. FDI Environment Variables	105
Table 4-1. eqDRC SVRF Statements	135
Table 5-1. Flat Versus Hierarchical Results Presentation	145
Table 8-1. Cumulative Runtime Statistics	218
Table 9-1. Connectivity-Dependent Layer Operations	251
Table 9-2. Text Processing Statements	258
Table 9-3. DEBUG Results Annotations	299
Table 10-1. Default Device Properties	326
Table 10-2. Value Array Listing	333
Table 11-1. Path Tracing	358
Table 11-2. Built-In Device Pin Names for Mapping	358
Table 12-1. Case-Sensitive Comparison Settings	376
Table 12-2. Built-in Device Types for LVS Comparison	379
Table 12-3. Capacitor Required Pin Names	380
Table 12-4. Diode Required Pin Names	382
Table 12-5. MOS Transistor Required Pin Names	384
Table 12-6. Resistor Required Pin Names	386
Table 12-7. Bipolar Transistor Required Pin Names	387
Table 12-8. JFET Transistor Recognized Pin Names	389
Table 12-9. Inductor Recognized Pin Names	390
Table 12-10. Voltage Source Recognized Pin Names	391
Table 12-11. List of Property Values Based on the Scan Direction	414
Table 12-12. Device Reduction Statements	424
Table 13-1. Correspondence Table Column Header Definitions	488
Table 13-2. BY REJECTION Report CORRESPONDENCES Subsections and Related BY HCELL Reason Codes	490
Table 13-3. Hierarchical Degradation Effects in Layer Operations	493
Table 13-4. High-Shorted Pin Resolution Examples	499
Table 14-1. Primary Messages	563
Table 14-2. Secondary Messages—Errors	564
Table 14-3. Secondary Messages—Warnings	568
Table 14-4. Power/Ground Net Errors	588
Table 14-5. Input Errors	590
Table 14-6. Information and Warnings	599

Table 15-1. SPICE Netlist Notational Conventions	634
Table 15-2. Handling of Slash Characters in Names	635
Table 15-3. X Instantiated Device Parameter Scaling	642
Table 15-4. Capacitor Element	678
Table 15-5. Junction Diode Element	680
Table 15-6. JFET Element	682
Table 15-7. Inductor Element	684
Table 15-8. MOSFET Element	686
Table 15-9. Specification Statements for SPICE M Elements	689
Table 15-10. BJT Element	690
Table 15-11. Resistor Element	692
Table 15-12. Voltage Source Element	694
Table 15-13. SUBCKT Statements	696
Table 15-14. Subcircuit Calls	699
Table 16-1. Mapping of 1'b1 Signals	728
Table 16-2. Gate-Level Primitives	742
Table 16-3. Behavioral Statements	745
Table 16-4. Tcl Command Correspondence	767
Table 16-5. Tcl Commands With No Corresponding Command Line Option	769
Table 16-6. Tcl Special Characters	770
Table 16-7. Verilog-2001 Constructs	833
Table 16-8. V2LVS Errors	843
Table 16-9. V2LVS Warnings	847
Table 16-10. V2LVS Tcl Shell Errors and Warnings	850
Table 16-11. v2lvs::set_verbose Debugging Messages	852
Table 18-1. General Commands	878
Table 18-2. Layer Listing Commands	878
Table 18-3. Connectivity, Device, and Rule Check Listing Commands	879

Chapter 1 Overview

Calibre verification applications assist you in checking the physical and electrical integrity of IC designs. The tools discussed in this manual consist of Calibre nmDRC, Calibre nmDRC-H, Calibre nmLVS, Calibre nmLVS-H, and some related utilities.

When there is no possibility of confusion between flat and hierarchical tools, the term DRC is used instead of Calibre nmDRC/Calibre nmDRC-H. Similarly, LVS is used instead of Calibre nmLVS/Calibre nmLVS-H.

For current release-specific information, see the current [Calibre Release Notes](#).

Calibre nmDRC and Calibre nmDRC-H	25
Layout Utilities.....	26
Calibre nmLVS and Calibre nmLVS-H	26
Related Calibre Verification Tools.....	27
Syntax Conventions	28

Calibre nmDRC and Calibre nmDRC-H

The Calibre® nmDRC™ and Calibre® nmDRC-H™ tools perform design rule checking of integrated circuit layout designs. This tool set can also be used for auxiliary layout generation tasks.

The [Calibre nmDRC and Calibre nmDRC-H Command Line](#) tools support the following run configurations:

- Flat — Calibre nmDRC performs design rule checking by reading the input layout database flat and operating on the geometry at a single level. This tool requires a Calibre nmDRC license or equivalent.
- Hierarchical — Calibre nmDRC-H performs design rule checking hierarchically, which minimizes redundant processing of layout data. The tool stores, analyzes, and processes data once per cell instead of once for every placement of the cell. This tool requires Calibre nmDRC and Calibre nmDRC-H licenses.
- Multithreaded — The multithreaded (MT) configuration of Calibre nmDRC-H allows you to use multiple CPUs on the same machine. This configuration also allows the use of hyperscaling through the -hyper command line option. See “[Hyperscaling](#)” on page 146. The MT mode of operation requires sufficient Calibre nmDRC and Calibre nmDRC-H licenses for the number of processors used during the run.

- Multithreaded, multi-host — The multithreaded, multi-host (Calibre® MTflex) configuration of Calibre nmDRC-H allows you to use distributed, parallel processing using multiple machines. This configuration also allows the use of hyperscaling through the -hyper command line option. The Calibre MTflex mode of operation requires sufficient Calibre nmDRC and Calibre nmDRC-H licenses for the number of processors used during the run.

Complete licensing information is provided under “[Licensing: Physical Verification Products](#)” in the *Calibre Administrator’s Guide*.

Layout Utilities

The Calibre tool set includes some utility programs that enhance the productivity of the tools.

- The Foreign Database Interface allows you to convert LEF/DEF and OpenAccess databases to GDS or OASIS®¹ formats using the fdi2gds and fdi2oasis utilities.
- DBdiff allows you to perform numerous database comparison tasks that include third-party layout formats.

For more information about these tools, refer to the [Calibre Layout Comparison and Translation Guide](#).

Calibre nmLVS and Calibre nmLVS-H

The Calibre® nmLVS™ and Calibre® nmLVS-H™ tools extract layout netlists and compare a layout to a schematic netlist. Connectivity extraction functions in flat, hierarchical, and multithreaded (MT and MTflex) configurations. Connectivity extraction supports device recognition, short isolation, electrical rule checking (ERC), soft-connection checking, and SPICE circuit extraction. Circuit comparison functions in flat and hierarchical configurations.

Here is a summary of the [Calibre nmLVS and Calibre nmLVS-H Command Line](#) run configurations:

- Flat — Calibre nmLVS performs flat geometric layout versus schematic netlist comparison. When the -flatten command line option is used, a flat layout netlist is extracted and compared against the schematic netlist. This tool requires a Calibre nmLVS license or equivalent. A Calibre nmDRC license is required for layer operations.
- Hierarchical — Calibre nmLVS-H performs hierarchical layout versus schematic SPICE netlist comparison. It also extracts a hierarchical SPICE netlist from the layout. This tool

1. OASIS® is a registered trademark of Thomas Grebinski and licensed for use to SEMI®, San Jose. SEMI® is a registered trademark of Semiconductor Equipment and Materials International.

requires Calibre nmLVS and Calibre nmLVS-H licenses. A Calibre nmDRC and Calibre nmDRC-H license pair is required for layer operations.

- Multithreaded — The MT configuration of Calibre nmLVS-H allows you to use multiple CPUs on the same machine when performing circuit extraction. Hyperscaling with the -hyper option is supported during circuit extraction. Sufficient licenses are required for all the processors enabled during the run.
- Multithreaded flex — The MTflex configuration of Calibre nmLVS-H allows you to use distributed, parallel processing using multiple machines when performing circuit extraction. Hyperscaling with the -hyper option is supported during circuit extraction. Sufficient licenses are required for all the processors enabled during the run.

Additional licenses are required for specialized functionality in the Calibre nmLVS tool set. Complete licensing information is provided under “[Licensing: Physical Verification Products](#)” in the *Calibre Administrator’s Guide*.

LVS Utilities

The Calibre nmLVS tool set comes with utility programs that enhance the productivity of the tools.

- [V2LVS](#) — Verilog-to-LVS is a converter that translates a Verilog structural netlist into a SPICE-like netlist for use as input to Calibre nmLVS/Calibre nmLVS-H.
- [E2LVS](#) — EDIF-to-LVS is a converter that translates an EDIF structural netlist into a SPICE-like netlist for use as input to Calibre nmLVS/Calibre nmLVS-H.

These tools require a Calibre license environment. Complete licensing information is provided under “[Licensing: Physical Verification Products](#)” in the *Calibre Administrator’s Guide*.

Related Calibre Verification Tools

Many Calibre tools are useful in the Physical Verification process. Refer to the related documentation for more information.

For information on solutions to common Physical Verification problems see the [Calibre Solutions for Physical Verification](#) manual.

- [Calibre® Auto-Waivers™](#) — A Calibre nmDRC extension that enforces DRC, ERC, Calibre PERC, and DFM CFA error waivers. Waiver shapes are generated either through Calibre RVE (the preferred method) or through batch commands. The waivers are enforced based upon geometric interactions throughout the hierarchy rather than just within a given cell. Additionally, the waivers are enforced based upon area overlap criteria and the numbers of waiver shapes that interact with a result. See the [Calibre Auto-Waivers User’s and Reference Manual](#).

- **FastXOR** — An application that employs both DBdiff and Calibre nmDRC-H dual-database comparison capabilities to perform database XOR comparisons that are often five times faster than typical XOR runs. See the [Calibre Layout Comparison and Translation Guide](#).
- **Calibre® PERC™** — A circuit reliability verification platform for validating source and layout designs using advanced ERC, Electrostatic Discharge (ESD), Electrical Overstress (EOS), voltage propagation, and cell-based (LEF/DEF) checks. The Logic Driven Layout (LDL) interface allows current density, point-to-point resistance, high-level, and topology-aware DRC checks of a layout. See the [Calibre PERC User's Manual](#).
- **Calibre Query Server** — A command-line-driven database server that is used to access LVS connectivity information in a Standard Verification Database (SVDB). See the [Calibre Query Server Manual](#).
- **Calibre® Interactive™** — Enables the configuration and execution of Calibre verification tools from a graphical environment. See the [Calibre Interactive User's Manual](#).
- **Calibre® RVE™** — A graphical interface that allows you to view Calibre results interactively with your layout editor. See the [Calibre RVE User's Manual](#).
- **Calibre® DESIGNrev™** — A fast layout database viewer used for analyzing large layout databases in the final finishing stages before tapeout. See the [Calibre DESIGNrev Layout Viewer User's Manual](#).
- **Calibre® YieldAnalyzer and Calibre® YieldEnhancer** — Used for improving chip yield. Included critical area and feature analysis, and customized fill tools. See the [Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual](#).
- **Calibre® YieldServer** — Used for specialized applications pertaining to yield improvement. Properties are assigned to layout objects during a batch run, and those properties are then used for layout analysis, among other uses, in Calibre YieldServer. See the [Calibre YieldServer Reference Manual](#).

Syntax Conventions

The command descriptions use font properties and several metacharacters to document the command syntax.

Table 1-1. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.

Table 1-1. Syntax Conventions (cont.)

Convention	Description
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
Underline	Underlining indicates either the default argument or the default value of an argument.
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.
‘ ’	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
or	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
...	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.

Example:

```
DEvice {element_name [‘(‘model_name‘)’]}
    device_layer {pin_layer [‘(‘pin_name‘)’] ...}
        [‘<’auxiliary_layer‘> ...]
        [‘(‘swap_list‘)’ ...]
    [BY NET | BY SHAPE]
```


Chapter 2

Setup and Tool Invocation

Calibre nmDRC and Calibre nmLVS require rule file and design database inputs in certain formats in order to perform verification tasks. The command lines for these tools have a large set of options for configuring the runs.

Ensure your Calibre installation and configuration are correct, including installation of all applicable licenses. The *Calibre Administrator's Guide* contains the relevant information.

Calibre tools require that the CALIBRE_HOME environment variable be set. See “[Setting the CALIBRE_HOME Environment Variable](#)” in the *Calibre Administrator's Guide* for details. For lists of Calibre environment variables, see “[Calibre Environment Variables](#)” in the *Calibre Administrator's Guide*.

Required Input Files	32
Calibre nmDRC and Calibre nmDRC-H Command Line.....	36
Calibre nmLVS and Calibre nmLVS-H Command Line	45
Calibre nmLVS Reconnaissance Command Line.....	66
Calibre CB	76
Calibre Version Information	76

Required Input Files

Before you invoke a Calibre Physical Verification tool, certain files must exist.

- **Rule File** — Standard Verification Rule Format (SVRF) or Tcl Verification Format (TVF) form.
- **Layout Databases** — A geometric database for Calibre nmDRC applications, flat LVS, and hierarchical connectivity extraction; a layout SPICE netlist for LVS comparison.
- **Source Databases** — A SPICE netlist used for LVS comparison.

Rule File

Calibre rule files are written in the Standard Verification Rule Format (SVRF) or in Tcl Verification Format (TVF). Typical practice is to use separate rule files for separate verification tools, but this is not required.

Rule files contain two main categories of commands:

- **Specification statements** — Except for invocation arguments on the command line, rule file specification statements control the functional environment for Calibre tools, such as describing the layout and source databases, and specifying where to store the results of a run. Specification statements also guide internal heuristics.
- **Layer operations** — Rule file layer 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 used for layer derivations or output to DRC-style results databases.

The “[Key Concepts](#)” and “[Summary of Rule File Elements](#)” chapters of the *SVRF Manual* are good places to begin to understand rule files. A sample rule file is shown in the “[Rule File Examples](#)” chapter of the *SVRF Manual*.

For a discussion of TVF, see “[Tcl Verification Format](#)” in the *SVRF Manual*.

Required Statements

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

[Table 2-1](#) shows the required rule file statements for Calibre nmDRC applications.

Table 2-1. Required DRC Rule File Statements

Statement	Specifies ...
Layout System	The format of the layout data.
Layout Path	The location of the layout data.
Layout Primary	The top-level cell within the layout data.
DRC Results Database	The results database pathname and format.

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

Table 2-2. Required LVS Rule File Statements

Statement	Specifies ...
Layout System	The format of the layout data.
Layout Path	The location of the layout data.
Layout Primary ¹	The top-level cell within the layout data.
Source System	The format of the source data.
Source Path	The location of the source data.
Source Primary ¹	The top-level cell within the source data.
LVS Report	The pathname of the LVS comparison report.

1. Layout Primary and Source Primary statements are not required if your Layout System and Source System statements are set to SPICE.

Layout Databases

A layout database contains the physical description of a circuit. The Calibre tools support a number of input database formats.

You specify the database format using the [Layout System](#) statement in your rule file. For a given tool set, the layout database must be one of the system formats shown in [Table 2-3](#).

Table 2-3. Layout Database Formats

Layout System	DRC	DRC-H	LVS	LVS-H
GDSII (version 6.0)	Y	Y	Y	Y
OASIS (version 1.0)	Y	Y	Y	Y
OPENACCESS (version 22.43 and 22.50)	Y	Y	Y	Y
LEFDEF (version 5.8)	Y	Y	Y	Y
ASCII	Y		Y	

Table 2-3. Layout Database Formats (cont.)

Layout System	DRC	DRC-H	LVS	LVS-H
SPICE			Y ¹	Y
CNET			Y	

1. When -flatten is used.

For details about geometric layout databases, see “[Supported Geometric Layout Formats](#)” on page 102.

SPICE Format for Layout

Calibre nmLVS-H netlist-to-netlist comparison uses a SPICE or HSPICE netlist for the layout.

You can specify the path to the SPICE file in a [Layout Path](#) statement or on the command line. See “[SPICE Format](#)” on page 633 for a description of how Calibre interprets and writes SPICE.

CNET Format for Layout

CNET stands for Compiled NETlist, a proprietary netlist format. This database type can be used for the layout in flat LVS comparison.

The path to the CNET database directory must appear in a [Layout Path](#) statement. CNET can only be used in flat LVS.

Source Databases

A source database contains the original source reference information of a circuit for LVS applications. This is also called a schematic netlist or source netlist. The source file is compared to the layout during an LVS (layout versus schematic) run.

You must use a source database when doing LVS checks. The [Source System](#) statement identifies the reference to be compared. [Table 2-4](#) shows the allowed database formats.

Table 2-4. Source Database Formats

System Format	LVS	LVS-H
SPICE	Y	Y
CNET database	Y	

You must also specify the [Source Path](#), which is the path to the source file.

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

- [V2LVS](#) (Verilog-to-LVS) — Translates a Verilog structural netlist into a Calibre SPICE-like netlist.
- [E2LVS](#) (EDIF-to-LVS) — Translates an EDIF structural netlist into a Calibre SPICE-like netlist.

Calibre nmDRC and Calibre nmDRC-H Command Line

Performs flat or hierarchical design rule checking.

Usage

Calibre nmDRC (flat)

```
calibre -drc [-cb]
  [-waiver waiver_setup_file]
  [-nowait | -wait n | -lmretry retry_args]
  [-lmconfig licensing_config_filename ]
  [-E svrf_output_from_tvf] [-tvfarg argument]
  [-validprop [report_options | all] [trace]]
  rule_file_name
```

Calibre nmDRC-H (hierarchical)

```
calibre -drc -hier [-recon [inverse]] [-rhdb { save | restore } filename] [-fx] [-analyze]
  [-waiver waiver_setup_file ]
  [{ {-turbo [number_of_processors] [-turbo_all]}}
   | {-turbo_litho [number_of_processors]}
   | {-turbo [number_of_processors] [-turbo_all]
      -turbo_litho [number_of_processors]}}
   [{ -remote host,host,... | -remotefile filename | -remotecmd filename count }
    [-remotedata [-recoveroff | -recoverremote]]
    [{ -hyper [connect] [remote]} | -hdbflex | -hdbflex_acquire]
    [-nowait | -wait n | -lmretry retry_args]
    [-lmconfig licensing_config_filename]
    [-E svrf_output_from_tvf] [-tvfarg argument]
    [-validprop [report_options | all] [trace]]
    rule_file_name
```

Remote Host Launch

calibre -mtflex host_connection

Arguments

To display the Calibre command line, enter “calibre” with no arguments on the shell command line.

- **-drc**
A required option that specifies to perform design rule checking. Specifying this option without -hier performs flat design rule checking.
- **-hier**
An option that specifies to perform design rule checking hierarchically.

- **-recon [inverse]**

An optional argument that specifies to use Calibre nmDRC Reconnaissance mode (commonly referred to as *Calibre nmDRC Recon*) when processing the job. This argument automatically selects a subset of DRC rulechecks that identify gross errors early in the design and assembly process. The -hier argument must be specified with -recon.

inverse — Specifies to reverse the behavior of the operation by selecting the checks that are unselected by -recon and unselecting the checks that are selected by -recon.

See “[Chip-Level Verification](#)” in the *Calibre Solutions for Physical Verification* manual for information on using -recon. See “[Calibre RealTime Digital](#)” in the *Calibre Administrator’s Guide* for licensing information.

- **-rhdb {save | restore} filename**

An option that specifies to save or restore an RHDB (Reusable Hierarchical Database) file. An RHDB is useful in some flows to save time by avoiding the overhead of repeatedly constructing a hierarchical database.

save *filename* — Saves a proprietary image of the hierarchical database, generated from all input files in the rule file, as an RHDB file to the specified *filename*. The RHDB file is saved with an embedded checksum. Calibre exits once the RHDB file is saved.

restore *filename* — Restores an RHDB file from the specified *filename*. A checksum is generated from all input files and compared with the checksum in the specified *filename*. A mismatch between the checksums generates an error.

The following statements and command line argument are not supported with -rhdb:

- [Layout Place Cell](#)
- [DFM Read](#)
- [Layout Property Audit](#)
- [Layout Property Text OASIS](#)
- [Inside Cell ... WITH PROPERTY](#)
- [-hdbflex](#)
- [-hdbflex_acquire](#)
- **-fx**

An optional argument that specifies to perform FastXOR layout versus layout comparison. The rule file specified with this command must be configured to perform a dual-database XOR. The -hier argument must be specified with -fx. See “[FastXOR Database Comparison](#)” in the *Calibre Layout Comparison and Translation Guide*.
- **-cb**

An optional argument that specifies to use Calibre Cell/Block. Refer to “[Calibre CB](#)” on page 76 for more information.

- **-turbo [number_of_processors]**

An optional argument that instructs Calibre nmDRC-H to use multithreaded, parallel processing. If you do not specify *number_of_processors*, Calibre nmDRC-H runs on the maximum number of CPUs available for which you have licenses. In general, it is recommended to avoid specifying *number_of_processors*.

number_of_processors — Specifies a positive integer for the number of CPUs to use. Calibre nmDRC-H runs on the maximum number of CPUs available if you specify a value greater than the maximum number of available CPUs. For example, if you specify -turbo 3 for a 2 CPU machine, Calibre runs on only two processors.

See “[License Consumption for Distributed Calibre](#)” in the *Calibre Administrator’s Guide* for additional information about license scaling with CPU count.

- **-turbo_all**

An optional argument, used with the -turbo argument, that specifies to stop the invocation of Calibre when the number of CPUs specified by -turbo are not available. When this argument is not specified, Calibre normally uses fewer threads than requested if the requested number of licenses or CPUs is unavailable.

- **-turbo_litho [number_of_processors]**

An optional argument that specifies the number of processors to use for RET and MDP applications. This argument may only be specified with -turbo. If you do not specify *number_of_processors*, Calibre runs on the maximum number of CPUs available for which you have licenses, regardless of the -turbo setting.

number_of_processors — Specifies a positive integer for the number of CPUs to use for RET and MDP processes.

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

- **-remote host[, host ...]**

An optional argument that enables multithreaded operations on the specified remote host(s) in a distributed network. You must specify at least one *host*. You can specify additional *host* names in a comma-delimited format. The following requirements apply when specifying this argument:

- This argument can only be specified with the -turbo option, and cannot be specified with the -remotefile or -remotecommand arguments.
- This argument applies only to hierarchical applications on a homogeneous set of hosts. That is, all machines are the same supported platform type and must have the same address mode (64-bit).
- You should avoid specifying the primary host as a remote host. Doing so can have an adverse effect on performance.
- You must have the required number of licenses for your job.

For more details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

- **-remotefile** *filename*

An optional argument that enables multithreaded operations on remote hosts defined in the specified configuration file.

filename — Specifies the pathname of a configuration file containing information for the local and remote hosts on which to run the Calibre job.

The following requirements apply when specifying this argument:

- This argument can only be specified with the **-turbo** option. It may not be specified with the **-remote** or **-remotecmd** arguments.
- You must have the required number of licenses for the number of processors you intend to use.
- Do not use any CPUs on the primary host as remote CPUs. If you choose to do so, use spare CPUs on the primary host by specifying the **LOCAL HOST COMPUTE** statement in the configuration file. When this statement is enabled, the operations consume licenses for the local CPU count.

The remote hosts can be heterogeneous when using this option. That is, they may be different platform types and address spaces. However, it is recommended that you use hosts with the same address space for your runs.

For more details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

- **-remotecmd** *filename count*

An optional argument used for MTflex runs in an LSF environment that allows you to specify values for the **REMOTE COMMAND** and **LAUNCH CLUSTER** statements from the command line directly.

This option can only be used with **-turbo** or **-turbo_litho**, and may not be specified with **-remotefile** or **-remote**.

- *filename* — Specifies the path to the file on the primary host containing the commands for invoking a remote server on the remote host.
- *count* — Specifies the total number of CPUs that Calibre will attempt to connect to for the run. The *count* must be greater than or equal to 2.

The **-remotecmd** argument is equivalent to using the **-remotefile** argument to specify a configuration file containing the following commands:

```
LAUNCH CLUSTER MINCOUNT 2 COUNT <count>
REMOTE COMMAND <filename> ARGUMENTS [ %H:%P %C ]
```

For more details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

- `-remotedata [-recoverremote | -recoveroff]`

An optional argument used to control the behavior of the Calibre Remote Data Server. This argument can only be used with `-remote`, `-remotefile`, or `-remotecommand`.

Refer to [-remotedata](#) in the *Calibre Administrator's Guide* for more information about this command and optional arguments.

- `-hyper [connect] [remote]`

An optional argument used to enable hyperscaling mode. Hyperscaling mode enables the concurrent, parallel execution of rule file operations, thereby increasing the utilization of CPUs and improving scalability in the multithreaded (MT) and MTflex environments. This option can only be specified with `-turbo` and is recommended.

Hyperscaling uses the same licensing environment as MT and MTflex modes. No additional licenses are required. For more details on configuration, see the [Calibre Administrator's Guide](#).

`connect` — Enables connectivity-related operations to be processed by hyperscaling.
Using this keyword can increase memory usage.

`remote` — Enable remote pseudohierarchical database technology, allowing pseudo HDBs to be distributed across remote host. This keyword must be used with the `-remotedata` argument.

For more details, see “[-hyper](#)” in the *Calibre Administrator's Guide*.

See also “[Hyperscaling](#)” on page 146 for more details on using hyperscaling.

- `-hdbflex`

An optional argument that causes most of the HDB (Hierarchical Database) construction to be performed by the local host in a MTflex run. By default, when you run a job in MTflex mode, the HDB construction process is distributed among the local and remote CPUs. When you enable `-hdbflex`, most of the HDB construction process occurs on the local host, which does not connect to remotes until the later stages of HDB construction. This reduces the idle time of any remotes that would normally be connected during HDB construction. This process is identified in the transcript as the HDBFLEX INPUT PHASE.

With `-hdbflex`, remotes are not connected early in the Calibre process, so fatal connections to remotes are not known until after HDB construction. To manage this, enabling `-hdbflex` changes the default behavior for the `MINCOUNT` argument (in [LAUNCH AUTOMATIC](#), [LAUNCH CLUSTER](#), or [LAUNCH MANUAL](#)) when a connection issue is encountered. By default, the `MINCOUNT` argument determines the minimum number of connections that must be established in order for the Calibre run not to fail. When `-hdbflex` is enabled, the default behavior for `MINCOUNT` is to generate a warning when an issue occurs and ignore the `MINCOUNT` value. You can use the `CALIBRE_HDBFLEX_MIN` environment variable to control the behavior of `MINCOUNT` when you specify `-hdbflex`. Refer to the *Calibre Administrator's Guide* for information on the [CALIBRE_HDBFLEX_MIN](#) variable.

- **-hdbflex_acquire**
An optional argument that acquires resources in parallel with the HDB (Hierarchical Database) construction process. This can be particularly useful if remote acquisition takes a long time. The -hdbflex_acquire argument differs from the -hdbflex argument, which does not connect to remotes until the latter stages of HDB construction.
- **-waiver waiver_setup_file**
An optional argument that specifies a waiver setup file necessary for running Calibre AutoWaivers. This argument enables automatic waiving of DRC-style errors after appropriate data preparation has occurred. The -waiver argument can only be specified with the -drc option and cannot be used with the -analyze argument.
- **-analyze**
An optional argument that specifies to generate an analyze database (*analyze.dfmdb*) that can be opened in Calibre RVE and used to visually review the distribution of errors and identify potential areas for enhancing the quality of the layout. This argument can be used with -recon to analyze the checks selected by Calibre nmDRC Recon. This argument cannot be used with -waiver. The -hier must be specified with this argument.

See “[Chip-Level Verification](#)” in the *Calibre Solutions for Physical Verification* manual for usage information and “[Calibre RealTime Digital](#)” in the *Calibre Administrator’s Guide* for licensing information.

- **-E svrf_output_from_tvf**
An optional argument used when your rule file is in compile-time TVF format (see the “[Tcl Verification Format](#)” chapter of the *SVRF Manual*). By default, the TVF rule file is echoed to the runtime logfile, but the equivalent SVRF code is not written. This argument causes the TVF pre-processor to write the equivalent SVRF rule file to the filename specified as the *svrf_output_from_tvf* argument.
- **-validprop [report_options | all] [trace]**
An optional argument that invokes the “property validator,” which checks a rule file for missing and unused DFM properties. You must specify -drc and other arguments necessary for a Calibre run with the specified rule file.

report_options — Specifies one or more of the following to control what properties are reported. Valid options are:

- **missing** — Report properties that are used but not defined.
- **unused** — Report properties that are not used.
- **outputonly** — Report properties that are used only in output.

For example, properties are included in output from a DFM RDB operation and a DRC Check Map statement with the PROPERTIES keyword.

- automatic — Report properties that are not used or output, but are always generated by an operation, regardless of any user-specified options. May be abbreviated as auto.
- all — Reports missing, unused, outputonly, and automatic properties. This is the default.
- trace — Includes a trace of the property analysis in the output.

The property validator examines a limited set of rule file operations; see “[Property Validator](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

- ***rule_file_name***
A required argument that specifies the pathname of the rule file.
- **-nowait**
An optional argument that causes Calibre to queue only briefly (approximately 10 seconds) before attempting to acquire substitute licenses. This option is equivalent to specifying “-wait 0”.
- **-wait *n***
An optional argument that sets the maximum amount of time 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. The maximum value for *n* is 45000. Refer to the section “[Using License Substitution With Calibre Classic Licensing](#)” in the *Calibre Administrator’s Guide* for more information on substituting licenses.

In the following example, the -wait option queues on a 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.
```

- **-lmretry *retry_args***
An optional argument specifying the licensing mode (classic or loop) to use for the Calibre job. See “[Calibre Licensing Command Line Options](#)” in the *Calibre Administrator’s Guide* for more information.
- **-lmconfig *licensing_config_filename***
An optional argument that enables a user-specified search for default or substitute licenses. This search is used rather than the default license acquisition behavior. The *licensing_config_filename* argument specifies a configuration file that controls how the license search occurs. Refer to the section “[Calibre Licensing Configuration File](#)” in the *Calibre Administrator’s Guide* for more information.

- **-tvfarg *argument***

An optional argument specifying an *argument* that is passed to a compile-time TVF script. The *argument* can contain no space characters. The *argument* is read by the tvf::get_tvf_arg command in the TVF rule file.

- **-mtflex *host_connection***

An optional argument used to launch a remote host specified by the *host_connection* argument. See the *Calibre Administrator's Guide* for more information about configuring and using MTflex.

Description

Calibre nmDRC or Calibre nmDRC-H perform either flat (calibre -drc) or hierarchical (calibre -drc -hier) design rule checking. Calibre nmDRC and Calibre nmDRC-H are also used for Resolution Enhancement Technology (RET), Mask Data Preparation (MDP), and Design For Manufacturability (DFM) tasks. For licensing and configuration information, see the *Calibre Administrator's Guide*.

The table “[Required DRC Rule File Statements](#)” on page 33 shows the required specification statements for Calibre nmDRC/Calibre nmDRC-H.

Integers are represented internally in sign-magnitude or twos-complement format. The maximum database coordinate precision range is +/- 2⁶¹.

In multithreaded modes of operation (-turbo used), the number of CPUs enabled for the run is based upon license availability. The “[Licensing for Multithreaded Operations](#)” section of the *Calibre Administrator's Guide* has details. When the rule file requires more license types (for non-Litho applications) than just DRC/DRC-H pairs, the number of enabled CPUs is determined by the fewest number of available licenses for a given product that is required for the run. For example, if the rule file uses Pattern Matching, AutoWaivers, and DRC-H, and the number of available licenses among the tools is unequal, then the number of CPUs enabled corresponds to the product with fewest available licenses.

Examples

Example 1

The following examples show how to run both the flat and single-threaded hierarchical modes:

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

Example 2

These show examples for MT (multithreaded, single host) and MTflex (multithreaded, distributed hosts) modes, respectively:

```
calibre -drc -hier -turbo rules
```

```
calibre -drc -hier -turbo -remote host1,host2,host3 rules
```

These modes assume you have a local host machine with multiple CPUs. If using Calibre MTflex mode, your remote hosts must be accessible from your network and must have multiple CPUs. The -hyper option is recommended with -turbo when you use at least four CPUs.

Example 3

To save the logfile from your run, use output redirection. For example:

```
calibre -drc -hier -turbo -hyper rules >&! logfile &
tail -f logfile
```

The “tee” command can cause a run to take longer in cases when the rule file is very long and you are running many checks against a smaller design.

Calibre nmLVS and Calibre nmLVS-H Command Line

Performs flat and hierarchical LVS circuit extraction and comparison.

Many LVS command line options are also relevant to Calibre PERC, where the behavior is occasionally somewhat different than in LVS. See the *Calibre PERC User's Manual* for details.

Usage

```
calibre [-lvs { { [ { -tl | -ts } cnet_file_name ]  
    [ -nonames ]  
    [ -dblayers "name1,..." ]  
    [ -bpf [ no-extents ] ] [ -nl ] [ -cb ] }  
| [ -hier [ -automatch | -genhcells[=qs_tcl_file_name | select] ] ]  
| [ -flatten ]  
}  
[ -ixf ] [ -nxm ]  
]  
[ -spice spice_file_name | -layout spice_file_name ]  
[ -hcell cell_correspondence_file_name ] [ -xcell xcell_file_name ]  
[ -siggen ]  
[ -nowait | -wait n | -lmretry retry_args ]  
[ -lmconfig licensing_config_filename ]  
[ { -turbo [number_of_processors] [ -turbo_all ]  
    [ -lvs_supplement [number_of_processors] ] }  
| { -remote host,host,... | -remotefile filename | -remotecommand filename count }  
    [ -remotedata [ -recoverremote | -recoveroff ] ]  
| -hyper [ cmp ] [ pathchk ] [ remote ] ] ]  
[ -cmp_host host | -cmp_remotefile filename ]  
[ -waiver waiver_setup_file ]  
[ -E svrf_output_from_tvf ] [ -tvfarg argument ]  
[ -validprop [ report_options | all ] [ trace ] ]  
rule_file_name  
  
calibre { -lvs | -perc } { -cs [message_limit] | -cl [message_limit] }  
[ -nowait | -wait n | -lmretry retry_args ]  
[ -cb ] [ -lmconfig licensing_config_filename ]  
[ -E svrf_output_from_tvf ] [ -tvfarg argument ]  
rule_file_name
```

Arguments

To display command line information, enter calibre on the command line with no arguments.

- **-lvs**

A required option to run LVS comparison. It is not required when using -spice.

When **-lvs** is specified by itself, no layout netlist or circuit extraction report is produced (circuit extraction messages appear in the LVS Report). Connectivity is stored in memory and applied to flat LVS comparison. You can see the SPICE representation of the flat connectivity model by specifying the **-nl** command line option. If running LVS comparison flat, it is generally recommended to use the **-flatten** option.

When used with **-flatten**, Calibre nmLVS compares a source netlist to a layout netlist flat while using the hierarchical netlist comparison engine.

When used with **-hier**, Calibre nmLVS-H compares a source netlist to a layout netlist hierarchically. If the Layout System is geometric, LVS circuit extraction is also performed.

Circuit comparison is always a single-threaded process.

- **-auto[match]**

An option that specifies automatic correspondence matching by name for cells in Calibre nmLVS-H comparison. When **-automatch** is used, Calibre compares cells with the same name in the layout and source (in addition to any specified by the **-hcell** option or the **Hcell** rule file statement) as corresponding hierarchical entities. Calibre pushes all other cells down to the next level of hierarchy.

This option may not be specified with **-genhcells**, which applies to both circuit extraction and comparison.

This option does not apply to hierarchical circuit extraction, such as with **-spice**.

The **-automatch** option should only be used if the layout cells actually contain the same devices as the source subcircuits of the same name. Excellent performance is generally obtained with a relatively brief, but carefully chosen list of hcells instead of using **-automatch**.

The **-automatch** option allows the hierarchy to be handled as Calibre determines is best for performance. All layout cells that are neither flattened nor expanded by the system, and that have a like-named counterpart in the source netlist, are compared.

Cell 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 cell 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, however.

Pseudocells having names of the form **ICV_n**, which are generated by that tool as a performance optimization, do not participate in **-automatch**. Such cells are generated at artificial levels of hierarchy within Calibre and are unsuitable as hcells.

To exclude hcells that might be selected by **-automatch**, use the **LVS Exclude Hcell** specification statement.

See “**Hcells**” on page 474 for details on hcell comparison.

Calibre PERC also uses the **-automatch** option, but it has different behavior than Calibre nmLVS-H. See “**calibre -perc**” in the *Calibre PERC User’s Manual*.

- **-bpf [no-extents]**

An option that generates a database consisting of binary polygon format (BPF) files, a layout cross-reference file, and a ports file. You use BPF databases with third-party tools. This option is rarely used.

You can use this option in both historical flat operation (**-lvs** without **-flatten**) and in CNET translation runs. When used, this option is typically used with **-nl**.

You cannot specify this option with the **-flatten**, **-hier**, or **-spice** options; Calibre returns an error if this occurs.

The 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.

By default, Calibre outputs all [Connect](#) and [Device](#) seed layers. You can use the **-dblayers** option to select a specific set layers.

See “[Binary Polygon Format \(BPF\) Database](#)” on page 629 for additional information.

- **-cb**

This option is discussed under “[Calibre CB](#)” on page 76.

- **-cl [*message_limit*]**

An option that causes LVS to read and verify the SPICE netlist specified in the [Layout Path](#) specification statement. Calibre nmLVS issues any applicable warnings or errors and writes them to the logfile. The overall results are written to the LVS Report. Calibre nmLVS reads the SPICE netlist hierarchically (as is done with the **-hier** option) but does not generate any LVS comparison structures.

The primary status message in the LVS report is SYNTAX OK if the check succeeded or SYNTAX CHECK FAILED if the check failed.

By default, the count of error or warning discrepancy messages issued by the tool is 256 of each type. No more messages are issued after that count is reached for each discrepancy type. The *message_limit* is a positive integer that specifies how many discrepancy messages to issue for each type.

This option cannot be used with input systems other than SPICE netlists.

Calibre nmLVS reserves a flat license when using this option.

Calibre nmLVS exits with a non-zero status if the syntax check fails. In most Linux shells, you can check this status by examining the \$? shell variable.

- **-cmp_host *host***

An option that specifies to run LVS-H comparison on a remote host. A Calibre nmLVS Advanced license is required to use this option. See the [Calibre Administrator’s Guide](#) for license information.

This option may not be specified with **-cmp_remotefile** and is disallowed if circuit extraction is not performed as part of the LVS run. In other respects, this option has no

direct dependencies on options controlling circuit extraction. Circuit comparison is single-threaded. See [Example 7](#).

The remote *host* machine must be able to access the current local working directory through the same namespace as the originating process. If this is not the case, or if the local host machine is unreachable, then the following message is written to the transcript, and the run stops:

```
ERROR: Failed to connect to the remote LVS host <hostname>.
```

- `-cmp_remotefile filename`

An option that specifies a file containing commands for running LVS-H comparison on a remote host. A Calibre nmLVS Advanced license is required to use this option. See the *Calibre Administrator's Guide* for license information.

This option may not be specified with `-cmp_host` and is disallowed if circuit extraction is not performed as part of the LVS run. In other respects, this option has no direct dependencies on options controlling circuit extraction. Circuit comparison is single-threaded.

The specified *filename* must contain appropriate **LAUNCH MANUAL** and **REMOTE COMMAND** commands for initializing a remote LVS-H comparison host. See the *Calibre Administrator's Guide* for command reference material.

For example, this command sequence could be in the referenced *filename*:

```
// default is 600 seconds
LAUNCH MANUAL WAIT 30
REMOTE COMMAND ./config.sh ARGUMENTS [ lvs_host_7 ]
```

REMOTE COMMAND references *config.sh*, which could contain this:

```
#!/bin/sh
ssh $1 ${CALIBRE_HOME}/bin/rcalibre `pwd` 1 \
    ${CALIBRE_HOME} ${CALIBRE_HOME} \
    -par_cmp_remote ${CALIBRE_LVS_REMOTE_CONNECTION}
```

The argument *lvs_host_7* is a remote host name and is referenced by \$1 in the *config.sh* script. This script passes the current working directory (*pwd*) as the destination for the MTflex log file (the default is */tmp*), the \${CALIBRE_HOME} tree value from the primary host (master) environment, and the primary host name and port number to the remote host. The path of a Calibre tree and the value of \${CALIBRE_LVS_REMOTE_CONNECTION} are required in order for comparison to be performed on a remote host.

The \${CALIBRE_HOME} value in the preceding script is from the primary's environment. If that does not resolve correctly in the remote's environment, then use a value that does resolve properly.

The REMOTE COMMAND must reference a command or script that uses this command line option to perform the LVS comparison:

```
-par_cmp_remote host:port
```

`-par_cmp_remote` is a required special option and only applies in the context of a remote LVS comparison job. (It is an error to use it in any other situation.) This option is used as the only argument to “calibre” if the calibre command is sent to remote host for execution. The option is used as an argument to “rcalibre” if rcalibre is used to launch the remote job from the primary host.

The `CALIBRE_LVS_REMOTE_CONNECTION` environment variable is a special usage in this context and is available to any program or shell script specified by REMOTE COMMAND (`config.sh` in the preceding example). Its value is “*host:port*”, where *host* is the primary host name that the remote host connects with, and *port* is the port number of the connection. See [Example 8](#) for related setup scripts.

The `-cmp_remotefile` file does not do environment variable substitution; however, it does do substitution of `%H:%P` constructs (*host:port*) as part of a REMOTE COMMAND specification, so something like this could appear in the `-cmp_remotefile` script:

```
REMOTE COMMAND ./config.sh ARGUMENTS [ lvs_host_7 %H:%P ]
```

And `config.sh` could have this:

```
#!/bin/sh
ssh $1 $CALIBRE_HOME/bin/rcalibre 1 \
    $CALIBRE_HOME $CALIBRE_HOME -par_cmp_remote $2
```

The remote host machine must be able to access the current working directory through the same namespace as the originating process. If this is not the case, or if the primary host is unreachable, then the following message is written to the transcript, and the run stops:

```
ERROR: Failed to connect to the remote LVS host <hostname>.
```

- `-cs [message_limit]`

An option that causes LVS to read and verify the SPICE netlist specified in the [Source Path](#) specification statement. Calibre nmLVS issues any applicable warnings or errors and writes them to the logfile. The overall results are written to the LVS Report. Calibre nmLVS reads the SPICE netlist but does not generate any LVS comparison structures.

The primary host status message in the LVS report is SYNTAX OK if the check succeeded or SYNTAX CHECK FAILED if the check failed.

By default, the count of error or warning discrepancy messages issued by the tool is 256 of each type. No more messages are issued after that count is reached for each discrepancy type. The *message_limit* is a positive integer that specifies how many discrepancy messages to issue for each type.

Calibre nmLVS exits with a non-zero status if the syntax check fails. In most Linux shells, you can check this status by examining the `$?` shell variable.

This option cannot be used with input systems other than SPICE netlists.

Calibre nmLVS reserves a flat license when you use this option.

- `-dblayers "name1, ..."` (or `-db "name1, ..."`)

An option used in Pyxis Layout flows. This option controls the layer shapes written to a [Mask Results Database](#), which is not used by Calibre RVE but is used in Pyxis Layout. 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 option, Calibre writes all relevant layers to the Mask Results Database. The relevant layers include those that appear in [Connect](#) and [Sconnect](#) operations, all [Device](#) seed and pin layers from the rule file, and all [Stamp](#) target layers. Possible exceptions are contact layers, as specified with the [Mask Results Database](#) specification statement.

May not be specified with `-flatten`.

- `-E svrf_output_from_tvf`

An option that causes the TVF pre-processor to write the equivalent SVRF rule file to the filename specified as the `svrf_output_from_tvf` argument. This option may be used when your rule file is in compile-time TVF format (see the “[Tcl Verification Format](#)” chapter of the [SVRF Manual](#)). By default, the TVF rule file is echoed to the runtime logfile, but the equivalent SVRF code is not written.

- `-flatten`

An option that causes circuit extraction to generate a flat SPICE netlist and LVS circuit comparison is performed flat. The run transcript is similar to when a flat design is input when using the hierarchical engine.

When specified with the `-spice` option, or when a layout netlist is implicitly extracted, the layout netlist is flat. A circuit extraction report is produced.

When specified with the `-lvs` option, netlist-to-netlist LVS comparison is performed (as with `-hier`), but the results are flat.

If you specify the `-lvs -flatten` options, you omit the `-spice` option, and the layout is a geometric database, then Calibre generates an extracted layout netlist and performs circuit comparison in a single run. When the [Mask SVDB Directory](#) statement is specified in the rule file, the extracted netlist is written to that directory with the name `layout_primary.sp`. The `layout_primary` name is the name of the primary cell. Otherwise, the netlist is written as `lvs_report_name.sp` in the current working directory. The `lvs_report_name` name is taken from the [LVS Report](#) specification statement in this case.

The `-bpf`, `-cell`, `-dblayers`, `-nl`, `-nonames`, `-tl`, and `-ts` options cause errors when used with `-flatten`. All hierarchical options are incompatible with `-flatten`.

The `-cb`, `-ixf`, and `-nxf` options may be used with `-flatten`.

- `-genhcells[=qs_tcl_file | select]`

An option that automatically generates an hcell list during circuit extraction based upon matching cell names. The generated hcell list is used during circuit extraction and LVS-H

comparison. This option may not be specified with -automatch, which only applies during comparison.

The [Source System SPICE](#) and [Source Path](#) statements must be active in the rule file when this option is used or an error is given.

When the option is used by itself (no suffix or “select” keyword) during circuit extraction, the behavior is essentially the same as the following Query Server Tcl script:

```
hcells::automatch -on
hcells::read -auto_expand preset -rules <rules>
hcells::add_matching_hcells
hcells::print_hcells -file $CALIBRE_GENHCELLS_OUTPUT
puts "\n\nGenerated HCELL Status:"
puts [ hcells::status ]
puts "GENERATED HCELL File: $CALIBRE_GENHCELLS_OUTPUT"
```

Note

 Total hierarchical instance count (THIC) is not used as an evaluation criterion by default. To employ it, specify hcells::select in a *qs_tcl_file* script or specify the “select” keyword.

The “select” keyword mimics the [hcells::select](#) command’s default thresholds for hcell selection criteria. An hcell evaluation report like what is generated by the Query Server is written to the Mask SVDB Directory as *<topcell>.hcells.selectreport*, where *<topcell>* is the primary layout cell. If Mask SVDB Directory is unspecified, then the report is written to the working directory as *<lvs_report>. <topcell>.hcells.selectreport*, where *<lvs_report>* is the LVS Report filename.

For performance reasons, the select keyword is generally preferred over using the *=qs_tcl_file* suffix.

If the *=qs_tcl_file* suffix is used during circuit extraction, then the *qs_tcl_file* is the name of a user-defined Query Server Tcl script, and that script is executed instead of the default one described previously. See “[Tcl Shell Hcell Analysis Commands](#)” in the *Calibre Query Server Manual* for details about script commands.

The [CALIBRE_GENHCELLS_OUTPUT](#) global Tcl variable specifies the output filename and location. This variable must be referenced in a user script to output the hcell list. The output file may be empty, in which case no hcells are contributed to the aggregate hcell list.

The generated hcell list is stored in the Mask SVDB Directory using the filename *<topcell>.hcells*. If Mask SVDB Directory is unspecified, then *<lvs_report>. <topcell>.hcells* is written to the working directory.

This file includes user provided hcells from an hcell file, Hcell specification statements, and hcells selected by the [-genhcells](#) command option. Cells that are eliminated by other features like unbalanced hcell expansion, seed promotion, high-cost hcell removal are not included in this list. The list contains exactly the list of hcells that appear in the LVS Report.

If the [-genhcells](#) option is used during a comparison-only run (that is, no circuit extraction is performed), then an hcell file having one of the automatically generated pathnames

described previously must exist or an error is given. That hcell list becomes part of the aggregate hcell list used for comparison. An hcell list is not generated during a comparison-only run, and the `=qs_tcl_file` suffix may not be used in this case.

If hcell list generation fails, a runtime error is given.

Expansion of hcells during circuit extraction is controlled by the [LVS Auto Expand Hcells](#) statement in the rule file. Cell names specified in [LVS Exclude Hcell](#) statements in the rule file do not participate as hcells during a run regardless of the method of hcell specification. As for hcells specified by any method, hierarchy is handled during comparison as Calibre determines is best for performance.

See “[Hcells](#)” on page 474 and “[Hcell and Hierarchical Analysis](#)” in the *Calibre Query Server Manual* for more information about constructing hcell lists. Also see “[Performing Hcell Analysis in Calibre nmLVS](#)” in the *Calibre Interactive User’s Manual*.

- `-hcell cell_correspondence_file_name`

An option that defines an hcell correspondence list. The specified file defines corresponding cells in layout and source, respectively. Cells from the layout list are preserved during circuit extraction. Cell pairs that are found in both layout and source are compared to each other during circuit comparison.

Expansion of hcells during circuit extraction is controlled by the [LVS Auto Expand Hcells](#) statement in the rule file. Subcircuits not specified as hcells get expanded during circuit comparison.

Excellent performance is generally obtained with a relatively brief but carefully chosen list of hcells. See “[Hcells](#)” on page 474 and “[Hcell and Hierarchical Analysis](#)” in the *Calibre Query Server Manual* for more information about constructing hcell lists. Also see “[Performing Hcell Analysis in Calibre nmLVS](#)” in the *Calibre Interactive User’s Manual*.

You can also define cell correspondence using the `-genhcells` option and the [Hcell](#) specification statements in your rule file. If `-automatch` is specified, then automatically matched subcircuit names are added to the internal hcell list during circuit comparison (but not circuit extraction). All lists of hcells are concatenated internally. If no hcells are identified using any of these means, then the comparison is performed flat.

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.

Double quotation marks around a `-hcell` list name are treated as literal, but this is not true for the [Hcell](#) specification statement.

To prevent hierarchy modifications during circuit extraction only (but not during LVS comparison), use the [Layout Preserve Cell List](#) specification statement instead of an hcell list.

- **-hier**

An option that runs SPICE netlist-to-netlist comparison hierarchically. This option can only be specified with the **-lvs** option.

In order to compare source and layout netlists hierarchically, the tool must be told how to match the subcircuit names by using an hcell list. This list is specified by using the **-hcell**, **-genhcells**, or the **-automatch** command line options, or the **Hcell** rule file specification statement. All of the hcells specified by any of these three methods are concatenated into a single list, which determines the subcircuits to compare in layout and source. If you have no hcell-identifying options specified, then the LVS comparison is performed flat.

If you specify **-spice** with **-hier**, then the Layout System must be geometric (such as GDSII or OASIS). The layout netlist is extracted to the filename associated with the **-spice** option and is then compared against the source netlist specified in the **Source Path** statement.

If you specify the **-hier** option, you omit the **-spice** option, and the layout is a geometric database, then Calibre nmLVS-H generates an extracted layout netlist and performs circuit comparison in a single run. When the **Mask SVDB Directory** statement is specified in the rule file, the extracted netlist is written to that directory with the name *layout_primary.sp*. The *layout_primary* name is the name of the primary cell. Otherwise, the netlist is written as *lvs_report_name.sp* in the current working directory. The *lvs_report_name* name is taken from the **LVS Report** specification statement in this case.

The name of the extracted netlist is reported at the end of the transcript. Note that the name of the extracted netlist is compatible with the standard naming convention of Calibre xRC.

Calibre PERC uses the **-hier** option, but it has different behavior than Calibre nmLVS-H. See “[calibre -perc](#)” in the *Calibre PERC User’s Manual*.

- **-hyper [cmp] [pathchk] [remote]**

An option that specifies hyperscaling mode, which improves hardware utilization and run times.

Hyperscaling mode enables the concurrent, parallel execution of rule file layer operations, thereby increasing the utilization of CPUs and improving scalability in the MT and Calibre MTflex environments. The **-hyper** option can only be specified with the **-turbo** option and is recommended when a machine has at least four cpus. Hyperscaling uses the same licensing environment as MT and MTflex modes. No additional licenses are required. There is increased memory usage on the primary host when using **-hyper**.

This **-hyper** option supports layer operations during connectivity extraction. For additional details, see “[Hyperscaling for LVS Connectivity Extraction](#)” on page 276. Hyperscaling is not applied to circuit comparison, which always runs on a single thread.

If **cmp** is specified when layout circuit extraction and **-lvs -hier** are performed in the same run, then the LVS comparison module performs tasks simultaneously with the extraction module that ordinarily would occur only after circuit extraction tasks are complete.

Therefore, using **cmp** can improve run times. (By default, LVS comparison runs on the local host in MTflex mode. This can be changed by using the **-cmp_host** or **-cmp_remotefile**

options.) The cmp option shows the greatest benefit in runs where ERC, soft connection checking, and short isolation are performed.

The pathchk option enables concurrency of [Pathchk](#) layer operations in HDBs during an ERC run. This option is useful if Pathchk operations take the majority of time in a hyperscaling run. This option increases memory usage.

The remote option triggers remote pseudohierarchical database technology. This causes pseudo HDBs to be distributed across remote hosts in an MTflex run. Having 16 or more remote CPUs is optimal. For more details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

Using cmp and remote together enables reading of the source netlist from the start of the LVS run, which improves performance. Using cmp alone enables the reading the source netlist after the hierarchical database constructor finishes. See “[Example 6](#)” on page 63.

- -ixf

On option that generates an instance cross-reference file. For additional information about net cross-reference files, refer to the “[SVDB Cross-Reference Files](#)” on page 623.

The generated filename is *lvs_report_name.ixf*, where *lvs_report_name* is specified by the [LVS Report](#) specification statement in the rule file. If you specify the -ixf option and your rule file includes the [Mask SVDB Directory](#) specification statement, Calibre nmLVS writes the instance cross-reference file to the SVDB directory. In this case, the output file is in the form *layout_primary.ixf*, where *layout_primary* is from the [Layout Primary](#) specification statement. If you do not specify the Layout Primary statement, ICV_UNNAMED_TOP is substituted for *layout_primary*.

The cross-reference files generated in flat Calibre nmLVS with the use of the -ixf option are not equivalent to those generated for Calibre nmLVS-H.

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

- -layout *spice_file_name*

An option that overrides the [Layout Path](#) and [Layout System](#) statements in the rule file (which specify a geometric layout for circuit extraction) and causes the *spice_file_name* netlist to be used as the layout netlist instead.

The option -layout can be specified only with the [-lvs](#) option but cannot be specified with the -spice option.

- -lmconfig *licensing_config_filename*

An option that enables a user-specified search for default or substitute licenses. The user-specified search is used rather than the default license acquisition behavior. The *licensing_config_filename* argument specifies a configuration file that controls how the license search occurs. Refer to the section “[Calibre Licensing Configuration File](#)” in the *Calibre Administrator’s Guide* for more information.

- **-lmretry *retry_args***
An option that specifies classic or loop licensing mode. See “[Calibre Licensing Command Line Options](#)” in the *Calibre Administrator’s Guide* for more information.
- **-lvs_supplement [*number_of_processors*]**
An optional argument that allows Calibre nmLVS-H license pairs (calibreqlvs and calibreqlvls) to enable processing threads when the -turbo option is specified with -perc. This option does not apply to -turbo_all. For -perc usage, see “[calibre -perc](#)” in the *Calibre PERC User’s Manual*.
- **-nl**
An option that produces a flat netlist from the layout. This option may not be used with -spice or -flatten. This option is rarely used.
- **-nonames (or -non)**
An option that prevents the CNET writer from generating net and instance name files in the CNET database. Used only with -tl or -ts. This option is rarely used.
- **-nowait**
An option that Calibre queues only briefly (approximately 10 seconds) before attempting to acquire substitute licenses. This option is equivalent to specifying “-wait 0”.
- **-nxr**
An option that generates a net cross-reference file. For additional information about net cross-reference files, refer to the “[SVDB Cross-Reference Files](#)” on page 623.

The generated filename is *lvs_report_name.nxr*, where *lvs_report_name* is specified by the LVS Report specification statement in the rule file. If you specify the -nxr option and your rule file includes the Mask SVDB Directory specification statement, Calibre nmLVS writes the net cross-reference file to the SVDB directory. In this case, the output file is in the form *layout_primary.nxr*, where *layout_primary* is from the [Layout Primary](#) specification statement. If you do not specify the Layout Primary statement, ICV_UNNAMED_TOP is substituted for *layout_primary*.

The cross-reference files generated in flat Calibre nmLVS with the use of the -nxr option are not equivalent to those generated for Calibre nmLVS-H.
- **-remote *host[, host ...]***
An option that specifies to use the MTflex multithreaded parallel processing architecture. It can only be specified with the -turbo option. It enables multithreaded operation on remote hosts of a distributed network. You must specify at least one host parameter. A list of hostnames is comma-delimited and specifies that multiple hosts participate in multithreaded operations. You must have the required number of licenses for your job.

This option applies only to hierarchical applications on a homogeneous set of hosts. That is, all machines are the same supported platform type and must have the same address mode.

You should avoid specifying the primary host as a remote host. Doing so can have an adverse effect on performance.

When using MTflex to run Calibre nmLVS-H on distributed processors, connectivity extraction modules run in multithreaded mode. The LVS comparison stage runs only on the primary host processor.

For more details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

- **-remotecmd** *filename count*

An option used in MTflex runs in an LSF environment that allows you to specify two arguments from the command line directly. This option can only be used with -turbo. It may not be specified with -remotefile or -remote.

The *filename* argument specifies the **REMOTE COMMAND** setup file, and the *count* argument specifies the **LAUNCH CLUSTER** command’s COUNT parameter. The *count* must be greater than or equal to 2.

The -remotecmd option is equivalent to using the following configuration file with the -remotefile option:

```
LAUNCH CLUSTER MINCOUNT 2 COUNT <count>
REMOTE COMMAND <queue_command> ARGUMENTS [ %H:%P %C ]
```

For more details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

- **-remotedata** [**-recoveroff** | **-recoverremote**]

An option that controls the behavior of the Calibre Remote Data Server. These options can only be used with -remote, -remotefile, or -remotecmd.

The -remotedata option allows memory data to be distributed across remote hosts, thus reducing the storage overhead for the primary host. This option is enabled when -hyper remote is specified.

The -recoveroff option disables disaster recovery of remote data and is the default. The -recoverremote option enables disaster recovery of data in the event of remote host failures. This option is recommended when using large numbers of remotes or if the run typically takes more than 12 hours.

Further details are discussed under **-remotedata** in the *Calibre Administrator’s Guide*.

- **-remotefile** *filename*

An option that specifies to use the MTflex multithreaded parallel processing architecture, which enables multithreaded operation on remote hosts of a distributed network. It can only be specified in conjunction with the -turbo option. 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.

The remote hosts can be heterogeneous when using this option. That is, they may be different platform types and address spaces. However, it is recommended that you use hosts with the same address space for your runs.

In general, do not use any CPUs on the primary host as remote CPUs. If you choose to do so, use spare CPUs on the primary host by specifying the **LOCAL HOST COMPUTE** statement in the configuration file. When this statement is enabled, the operations consume licenses for the local CPU count.

When using MTflex to run Calibre nmLVS-H on distributed processors, connectivity extraction modules run in multithreaded mode. The LVS comparison stage runs only on the primary host processor.

For more details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

- -siggen

An option that causes LVS to generate a device signature in the log file during the circuit extraction phase of the run. It is associated with the SIGNATURE keyword of the [Device](#) statement. For information about device signatures, see “[Device Signatures](#)” on page 315. This option requires a Calibre Advanced Device Properties license.

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

- -spice *spice_file_name* (or -spi *spice_file_name*)

An option that extracts a SPICE netlist from a geometric layout and directs output to *spice_file_name*. The extracted netlist is used in place of the original geometric layout for comparison against the source.

If *spice_file_name* ends with either the .Z or .gz suffixes, then the output file is compressed using the Linux compress or gzip commands, respectively. These utilities must be available in your environment in order for the compression to occur.

See the -hier and -flatten options for details of how to run circuit extraction implicitly without using the -spice option.

When you perform circuit extraction and circuit comparison in a single run, Calibre verifies that the Source Path and the LVS Report file are specified in the rule file and the paths accessible before executing the circuit extraction (-spice) step. This check is not performed when you specify -spice without **-lvs**.

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.

If you intend to run Calibre xRC using the extracted netlist, specify the Mask SVDB Directory XRC statement in your rule file. This writes the results of circuit extraction to a hierarchical database (HDB) and places it in the SVDB.

- **-tl *cnet_file_name***

An option that performs layout translation to a CNET netlist specified by *cnet_file_name*. You specify the layout in the [Layout Path](#) specification statement. This option may not be used with -flatten. This option is rarely used.

- **-ts *cnet_file_name***

An option that performs source translation to a CNET netlist specified by *cnet_file_name*. You specify the source in the [Source Path](#) specification statement. This option is rarely used.

Output nets are identified with numerical IDs only, not 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.

The generated netlist may be useful with some third-party tools; however, it is not intended for general circuit extraction or LVS debugging.

This option may not be used with -flatten.

- **-turbo [*number_of_processors*]**

An option that enables multithreaded parallel processing.

The optional *number_of_processors* argument is a positive integer that specifies the number of CPUs to use. If you do not specify a *number_of_processors*, Calibre nmLVS-H runs on the maximum number of CPUs available for which you have licenses. In general, you should avoid specifying *number_of_processors*.

The -turbo applies only to hierarchical circuit extraction, not to the circuit comparison (-lvs -hier). Also, -hyper applies to circuit extraction, and it is recommended to use -hyper for machines with at least four cpus. If -hyper cmp is specified with -lvs -hier -turbo, then LVS comparison processes run while circuit extraction processes are running, which is more efficient.

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

```
calibre -lvs -hier ... -turbo 3 ...
```

operates on two processors for a 2-CPU machine.

See “[Licensing for Multithreaded Operations](#)” in the *Calibre Administrator’s Guide* for additional information about license scaling with CPU count.

- **-turbo_all**

An option used with -turbo that halts the run if the tool cannot obtain the exact number of CPUs you specified using -turbo.

Specifying the `-turbo_all` option without *number_of_processors* is effectively the same as specifying the maximum number of CPUs on the machine. For example:

```
calibre -spice layout.net -turbo -turbo_all rule_file
```

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

```
calibre -spice layout.net -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 “[Licensing for Multithreaded Operations](#)” in the *Calibre Administrator’s Guide* for additional information about license scaling with CPU count.

- **`-tvfarg argument`**

An option that specifies an *argument* that is passed to a compile-time TVF script. The *argument* can contain no space characters. The *argument* is read by the `tvf::get_tvf_arg` command in the TVF rule file.

- **`-validprop [report_options | all] [trace]`**

An option that invokes the “property validator,” which checks a rule file for missing and unused DFM properties. You must also specify `-lvs` and other arguments necessary for a Calibre run with the specified rule file.

report_options — Specify one or more of the following to control what properties are reported on.

- `missing` — Report properties that are used but not defined.
- `unused` — Report properties that are not used.
- `outputonly` — Report properties that are used only in output.

For example, properties are included in output from a DFM RDB operation.

- `automatic` — Report properties that are not used or output, but are always generated by an operation, regardless of any user-specified options. May be abbreviated as `auto`.

For example, the Device Layer operation automatically generates DFM properties.

- `all` — Report missing, unused, `outputonly`, and `automatic` properties. This is the default.
- `trace` — Include a trace of the property analysis in the output.

The property validator examines a limited set of rule file operations; see “[Property Validator](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

- **-wait *n***

An option that 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. The maximum value for *n* is 45000. Refer to the section “[Using License Substitution With Calibre Classic Licensing](#)” in the *Calibre Administrator’s Guide* for more information on substituting licenses.

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

```
calibre -lvs -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.
```

- **-waiver *waiver_setup_file***

An option that specifies a waiver setup file necessary for running the automatic error waiver flow. This option enables automatic waiving of ERC errors after appropriate data preparation has occurred. This option only applies when connectivity extraction is performed. See “[Performing a Waiver Run](#)” in the *Calibre Auto-Waivers User’s and Reference Manual*.

- **-xcell *file_name***

An option that specifies an xcell file, which is primarily used by Calibre xRC. If a layout cell name appears in the -xcell file and in the LVS hcell list, then the cell is never expanded during circuit extraction. See “[Hierarchy Control with Xcells](#)” in the *Calibre xRC User’s Manual* for details about xcell files.

Xcells are immune from geometry pushdown from higher levels of the design, except for layers derived using the Extent or Extent Cell operations. To globally allow this pushdown, you can specify Mask SVDB Directory NOPEXPCELLS along with the CCI, XACT, or XRC option. To achieve a similar effect on a per-cell basis, you can draw a shape on the desired layer in an xcell, or you can derive a layer using Extent and the AND operation with the LVSCAREFUL keyword.

- ***rule_file_name***

An argument that specifies a rule file.

Description

Flat LVS comparison occurs when the -flatten option is used. The runtime behavior is similar to LVS-H in that layout SPICE netlist extraction occurs first, but extraction and LVS comparison are both flat. If you want to perform flat comparison, this is the recommended method.

Flat circuit extraction is also performed when the -lvs option is used without the -hier or -flatten options. In general, this mode is not recommended. Connectivity information is stored in memory and compared against a flattened view of the geometric layout database.

LVS-H performs hierarchical SPICE circuit extraction from a geometric layout and hierarchical LVS comparison of SPICE netlists. The -spice option extracts a hierarchical layout SPICE netlist, and the -lvs -hier options perform hierarchical circuit comparison. However, if the Layout System is geometric, then -lvs -hier performs circuit extraction automatically without using -spice.

Integers are represented internally in sign-magnitude or twos-complement format. The maximum database coordinate precision range is +/- 2⁶¹.

The table “[Required LVS Rule File Statements](#)” on page 33 shows the required rule file specification statements for Calibre nmLVS and Calibre nmLVS-H.

Calibre nmLVS applications exit with a non-zero status if they cannot complete any form of requested processing due to fatal error conditions. In most Linux shells, you can check this status by examining the \$? shell variable. For example:

```
#!/bin/sh
calibre -lvs -hier rules
if [ $? -eq 0 ]
then
    echo "No errors."
else
    echo "Error in LVS!"
fi
```

See “[Basic LVS Rule File Template](#)” in the *Calibre Solutions for Physical Verification* manual. Also see “[LVS Best Practices](#)” in the same manual.

Examples

Example 1

The following example extracts a hierarchical SPICE netlist from a geometric layout and writes the netlist to file *layout.net*.

```
calibre -spice layout.net rules.extract
```

The following two examples show how to do this using MT and MTflex modes with hyperscaling, respectively:

```
calibre -spice layout.net -turbo rules.extract
```

```
calibre -spice layout.net -turbo -remote host1,host2 rules.extract
```

The -hyper option is also recommended if you have at least four CPUs on the local host.

Example 2

The following example performs hierarchical LVS comparison between a SPICE layout and source. The file called *cells* specifies cell correspondence.

Rule file:

```
// layout is SPICE
LAYOUT SYSTEM SPICE
LAYOUT PATH "./layout.net"
LAYOUT PRIMARY top
...
```

Command line:

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

Example 3

The following example extracts a hierarchical SPICE netlist called *layout.net* from a geometric layout and then compares it to a SPICE source. Both circuit extraction and comparison are hierarchical. The file called *cells* specifies hcell correspondence and is used for netlist comparison. If the hcell list is not used and none are specified in the rules, then the comparison is flat.

Rule file:

```
// layout is geometric
LAYOUT SYSTEM GDSII
LAYOUT PATH "./design.gds"
LAYOUT PRIMARY "top"

SOURCE SYSTEM SPICE
SOURCE PATH "./top.spi"
SOURCE PRIMARY top
...
```

Circuit extraction and comparison command line:

```
calibre -spice layout.net -turbo -lvs -hier -hcell cells rules
```

If you omit the -spice option, then the netlist is written to the working directory as *lvs_report.sp*, where *lvs_report* is the name of the LVS Report specified in the rules. If you specify a Mask SVDB Directory, then the netlist is written there with the top-level cell name as part of the filename.

Performing netlist extraction and LVS comparison in the same run has a potential drawback in that if the circuit extraction phase has errors, the comparison phase almost certainly will also, so the comparison would be wasteful.

If you add “-hyper cmp” to the preceding command line, then connectivity extraction uses hyperscaling, and LVS comparison processes run during the extraction phase, which is more efficient. It is recommended to run -hyper when you have at least four CPUs on the local host.

Example 4

The following example extracts a flat SPICE netlist from a geometric layout system and then compares it to a SPICE source. Comparison is flat.

Rule file:

```
// geometric layout
LAYOUT SYSTEM GDSII
LAYOUT PATH "./design.gds"
LAYOUT PRIMARY "top"

SOURCE SYSTEM SPICE
...
```

Command line:

```
calibre -lvs -flatten rules
```

The extracted SPICE netlist is written in the same way in [Example 3](#).

Example 5

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

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

Note

 Use -automatch only if the layout cells actually contain the same devices as the source subckts of the same name.

If the rule file has a geometric Layout System, then using -turbo is also recommended, which benefits the circuit extraction phase.

Example 6

This runs LVS-H in MTflex mode with circuit extraction on four remote hosts using hyperscaling and circuit comparison modules running simultaneously on the local host. Remote Data Server distribution is enabled with catastrophic recovery from a lost remote host disabled.

```
calibre -lvs -hier -turbo -hyper cmp remote\
-remote host1,host2,host3,host4 -remotedata -recoveroff rules
```

If, in addition, you want to run LVS-H comparison on a remote host, you can add “-cmp_host host5” to the preceding command line. This makes using the cmp option to -hyper redundant, so cmp could be omitted in this case, and you would get the same effect.

Example 7

To run LVS-H comparison on a remote host whose name is known in advance and circuit extraction in MTflex mode, you can do this:

```
calibre -lvs -hier -turbo -remotedata -recoveroff \
-remotefile remote_script -cmp_host machine_name rules
```

The *remote_script* is a MTflex configuration script similar to what is used for a DRC run. Substitute the appropriate host name for *machine_name*. The *-hyper* argument set can be included and is useful for host machines with at least four CPUs, with 16 being optimal.

Example 8

The following setup allows LVS-H comparison to run on a remote host requested through a set of user job queuing scripts. Here, *remote.txt* contains MTflex commands that initiate the remote job. MT mode circuit extraction runs on the local host.

```
calibre -lvs -hier -turbo -hyper cmp -cmp_remotefile remote.txt rules
```

remote.txt file contents:

```
LAUNCH MANUAL WAIT 30
REMOTE COMMAND submit_script ARGUMENTS [ <arg> ... ]
```

The *submit_script* (not shown) is frequently a wrapper script for a network queue management application that launches Calibre jobs. This script is for the LVS comparison portion of the run. The ARGUMENTS specification forwards information to the remote host via the *submit_script*. This method is similar to how you might submit an MTflex DRC job.

Note

 The primary host's environment is used for dereferencing variable values in the REMOTE COMMAND script. It is important to confirm that any values passed by that script apply in the remote host's environment.

The *<arg>* arguments shown previously are passed to the *submit_script* for processing. For example, the name of the following script could be passed in the ARGUMENTS block to the *submit_script*:

```
#!/bin/sh
CALPATH="/software/CALIBRE_TREES/aoi/latest"
OPT="-par_cmp_remote $CALIBRE_LVS_REMOTE_CONNECTION"
```

The *-par_cmp_remote* option launches a remote comparison job. The path of the Calibre tree and the *host:port* value from the built-in variable *\$CALIBRE_LVS_REMOTE_CONNECTION* must resolve in the remote host's environment in order to establish a connection.

If the preceding information were not passed as part of the REMOTE COMMAND ARGUMENTS set, then the *submit_script* must provide the necessary information to a remote host as discussed in the [-cmp_remotefile](#) section previously.

The following setup does not assume a queue management wrapper script and could be used to launch a remote job directly. This command initiates the job on the primary host:

```
calibre -lvs -hier -turbo -cmp_remotefile remote.txt rules
```

(The -hyper argument set may be included.)

remote.txt file contents:

```
LAUNCH MANUAL WAIT 30
REMOTE COMMAND ./launch ARGUMENTS [ host_name %H:%P ]
```

The *host_name* is the name of a remote host and *%H:%P* is resolved as the primary host name and the port number for the connection. These are passed to *launch* as arguments. (*%H:%P* is used here because environment variables like \$CALIBRE_LVS_REMOTE_CONNECTION do not resolve in the *-cmp_remotefile* file.)

launch file contents:

```
#!/bin/sh
ssh $1 $CALIBRE_HOME/bin/rcalibre `pwd` 1 $CALIBRE_HOME $CALIBRE_HOME \
-par_cmp_remote $2
```

From the primary host, a secure socket connection (ssh) is established with *host_name* (\$1) to run LVS comparison remotely. “rcalibre” launches the Calibre job. The path to the run directory (pwd) becomes the destination for the MTflex log file. The number of environment variables passed to the remote host is 1, which corresponds to *CALIBRE_HOME*. Here, *\$CALIBRE_HOME* is assumed to resolve both in the primary’s and the remote’s environment. This may not necessarily be true, so be careful to check this. The *%H:%P* value (\$2) resolves automatically in the environment of the primary host and the value is provided to the remote to establish a port connection.

The most likely causes of failure for this sort configuration are passing an incorrect path to the Calibre tree in the remote environment or passing an incorrect *host:port* argument to the *-par_cmp_remote* option. So if the run fails, begin by checking those items.

Calibre nmLVS Reconnaissance Command Line

Performs stand-alone hierarchical ERC, Sconnect stamping violation (soft connection), and short isolation checks on the layout using a rule file or a Mask SVDB Directory.

The command line options for multithreading have essentially the same semantics as for LVS. Those options are fully described under the LVS command line section with just the differences discussed in detail here.

Usage

calibre -recon

```
{ { -erc=svrf_script | -softchk[=tcl_script] } -svdb svdb_dir [ top_cell ] } |
{ -erc [ -svdb svdb_dir [ top_cell ] ] { -exec tcl_script [args] } |
{ -si[={ALL | DB | IO | PG | tcl_script}]|
[ { { -remote host,host,... } | { -remotefile filename } |
{ -remotecommand filename count } |
[ -remotedata [ -recoverremote | -recoveroff ] ] [ -hyper [ -remote ] ] ]
{ rule_file_name | { -svdb svdb_dir [ top_cell ] } } }
[ -turbo [ number_of_processors ] [ -turbo_all ] }
```

Arguments

To display the complete command usage, enter “calibre” on the command line with no arguments.

- **-recon**

A required argument specifying to run Calibre nmLVS Reconnaissance mode. This argument enables certain verification tasks to be run stand-alone outside of an LVS run. Either the **-erc**, **-si**, or **-softchk** option must also be specified.

A Calibre nmLVS Reconnaissance license is required whenever this option is used.

- **-erc=svrf_script**

An option specified with **-recon** that operates on layer data in the **svdb_dir**. The **=svrf_script** suffix is required in this usage.

The **svrf_script** is the name of a file that contains SVRF code used for deriving layers and is executed using the layout data in the **svdb_dir**. The code that appears in the **svrf_script** is handled in a similar way to **dfm::new_layer** -svrf -keep_all_layers -rdb_as_files. The intent of the **-erc** option is to allow layer operations such as **Pathchk** to be performed on data in the PHDB. In order to do this, the rule file used to generate the **svdb_dir** should be the same as for LVS circuit extraction. LVS specification statements such as LVS Power Name and LVS Ground name are not executed if included in the **svrf_script**, but their settings are observed if they appear in the rules used to generate the **svdb_dir**.

A Calibre nmLVS/Calibre nmLVS-H license pair is required to use this option.

See [Example 1](#).

- **-softchk[=tcl_script]**

An option that performs soft-connection checks on the layout data (PHDB) in the *svdb_dir*. The Mask SVDB Directory SI keyword must have been active in the rules when the *svdb_dir* was created, or an error is given.

By default, all **Sconnect** stamped layers (the second layer arguments in Sconnect operations) for which there was a stamping conflict during the circuit extraction run are processed. Soft connection checking is then performed as if **LVS Softchk LOWER** were specified in a circuit extraction run. This default can be changed by specifying the *=tcl_script* suffix, where *tcl_script* is a Query Server Tcl shell script.

If *=tcl_script* is specified, the soft connection checking is performed based upon **qs::softchk** or **qs::softchks** commands in the script. Query Server Tcl shell and Calibre YieldServer commands are also supported.

A Calibre nmLVS/Calibre nmLVS-H license pair is required to use this option. If the *tcl_script* is specified, a Calibre Query Server license is also required.

See [Example 2](#).

- **-svdb svdb_dir [top_cell]**

An argument set that specifies a Mask SVDB Directory. The *top_cell* is an optional argument specifying a primary cell name. The *top_cell* is defined when there is data from more than one primary cell in the SVDB and you want to select the primary cell to use. This option requires that **Mask SVDB Directory** SI was used to create the *svdb_dir*, or an error is given.

This argument set is required when **-erc=srvf_script** or **-softchk** is specified. It is optional with **-erc -exec** or **-si**.

The remaining behaviors discussed here apply only when **-si** is specified.

When **-si** is specified, stand-alone short isolation is performed on the layout data in the *svdb_dir*. The **-si=** modal arguments have analogous effects to when a *rule_file_name* is specified.

Either a *rule_file_name* argument or the **-svdb** option must be specified with **-recon -si**. If **-svdb** is specified, then the following apply:

- If there are specification statements that apply to short isolation in the rules stored in the *svdb_dir*, then those settings are applied as they are when a *rule_file_name* is specified.
- If no **-si** modal argument is specified, then short isolation results are presented per the **short_db::isolate_shorts -by_cell -by_layer** options. A *tcl_script* modal argument allows custom control of results presentation.
- Only Connect and Sconnect operations that provide connectivity to Device pin layers are executed. This can be altered if the rules stored in the *svdb_dir* contain the **LVS SI Select Connects** statement, in which case, only the Connect and Sconnect statements corresponding to that statement's settings are executed.

Alternatively, if **-si=tcl_script** is specified, then the **short_db::select_connects** command may be used in the Query Server script to produce the same effect.

- At the conclusion of the run, the **svdb_dir** contains an interactive short isolation database that can be loaded into Calibre RVE or for further processing in the Query Server Tcl shell.
- If **-si=DB** is specified on the command line, the contents of the SVDB are searched for sufficient information to perform short isolation, but no short isolation is performed. If there is sufficient information, then the message “SVDB READY FOR RECON SI” appears.
- All of the command line options that enable multithreading may be used except **-hyper**.
- **-erc -exec tcl_script [args]**

An argument set that specifies Query Server Tcl shell or YieldServer commands in the **tcl_script** are executed on data in a Mask SVDB Directory database that was created using the SI keyword. When this argument set is used, **-exec** must be the final option on the command line. The optional **args** are arguments passed to the **tcl_script**.

In this usage, if the **-svdb** option is also specified, then the **svdb_dir** is the one the **tcl_script** operates on unless overridden by the script. If the **-svdb** option is omitted, then an SVDB must be opened through the **tcl_script**.

- **-si[={ ALL | DB | IO | PG | tcl_script }]**

An option specified with **-recon** that performs stand-alone short isolation on the layout.

Short isolation is performed as if the **LVS Isolate Shorts YES BY LAYER BY CELL** statement were in the rules. Additional LVS Isolate Shorts secondary keywords apply (except for NO, which is always ignored with **-si**) if they exist in the rules specified by the **rule_file_name** or are contained in the SVDB directory specified by the **-svdb** option.

The presence of an equals sign (=) suffix to the **-si** option indicates an optional run mode. The modal argument following “=” corresponds either to an LVS Isolate Shorts statement or to a Query Server Tcl Shell script consisting of short isolation commands. The modal arguments (case-insensitive, except for **tcl_script**) are as follows:

Modal Argument	Equivalent Specification Statement and Definition
ALL	LVS Isolate Shorts YES CELL ALL All shorts are isolated at all levels of hierarchy. Any LVS Isolate Shorts NAME keywords appearing in a rule file are ignored. CELL PRIMARY is ignored.

Modal Argument	Equivalent Specification Statement and Definition
DB	<p>Not equivalent to an LVS Isolate Shorts statement.</p> <p>All necessary data for running short isolation in subsequent runs with the -svdb option is saved to the Mask SVDB Directory.</p> <p>This option is useful when many shorts are expected such that numerous short isolation runs will be needed using data in the SVDB. In this case, it can be more efficient to store the layout data in one run and then run short isolation later using the -svdb option later, like this:</p> <pre>calibre -recon -si=db -turbo rules calibre -recon -si -svdb svdb</pre> <p>Running -turbo without any remotes is also the fastest way of running circuit extraction and populating the SVDB. So running that part of the flow separately can improve overall processing time for shorts.</p> <p>When the SVDB has all the necessary information for short isolation, the following message appears in the transcript:</p> <p>SVDB READY FOR RECON SI.</p>
IO	<p>LVS Isolate Shorts YES CELL PRIMARY && !NAME LVS_POWER_NAME LVS_GROUND_NAME</p> <p>Shorts are isolated only between non-supply nets. The PRIMARY keyword is the default behavior but may be overridden.</p>
PG	<p>LVS Isolate Shorts YES CELL PRIMARY && NAME LVS_POWER_NAME LVS_GROUND_NAME</p> <p>Shorts are isolated only for primary level nets specified in LVS Power/Ground Name statements. The PRIMARY keyword is the default behavior but may be overridden.</p>
<i>tcl_script</i>	<p>Not equivalent to an LVS Isolate Shorts statement. If there is an LVS Isolate Shorts statement in the rules, then it is ignored.</p> <p>Specifies a Query Server Tcl shell script. The script is executed using the SVDB contents. Both Short Isolation Commands and Short Repair Database Commands are supported. If short isolation is not performed in the script, then no short isolation occurs.</p> <p>(Using an extension like .tcl or .qs is a good practice because it avoids the possibility of matching future modal operator names.)</p>

If there is an LVS Isolate Shorts YES statement in the rules and one of the ALL, IO, or PG modal arguments is specified, then the following occurs:

- a. If the LVS Isolate Shorts statement contains additional arguments that conflict with the modal argument's behavior, then the conflicting rule file statement's arguments are ignored.

For example, if -si=IO is specified, and the rule file contains this:

```
LVS_ISOLATE SHORTS YES CELL ALL || NAME LVS_POWER_NAME  
LVS_GROUND_NAME
```

then the NAME specification is ignored because of a conflict, and the IO argument's behavior prevails.

- b. If the LVS Isolate Shorts statement contains arguments that do not conflict with the modal argument's behavior, then the rule file statement's arguments are applied to the modal argument's behavior.

For the example in part a, the CELL ALL specification is applied because that does not conflict with the IO option's behavior.

Connectivity text shorts, opens, and unattached label warnings are given in the transcript, just as they are in a calibre -spice run. However, a circuit extraction report is not written, nor is a *.shorts* database.

The tail of the transcript contains a section like this when shorts are detected:

```
Current short serial number is 1  
Short circuit (index: 0) (cell: TOPCELL) (net_id: 2) (text: PWR).  
    PWR at (1.3 82.4) (layer metal2)  
    top_short at (3.4 78) (layer metal2)
```

```
Warning: 1 short paths from 1 out of 1 shorts were selected in  
primary cell TOPCELL for short isolation.
```

Shorts from non-primary cells may also be shown depending on rule file settings. If shorts are filtered out due to use of a -si= modal argument, then this warning is given:

```
Warning: <i> short paths from <j> out of <k> shorts were selected  
for short isolation.
```

where k is the number of detected shorts, j is the number of isolated shorts, and i is the number of distinct net paths between text labels among the j shorts.

If no shorts are detected, then this is shown instead:

```
Short Isolation results are clean for ALL cells.
```

If an LVS Isolate Shorts NAME keyword is applied when the rules are read, or if the -name option is used for `qs::isolate_shorts` or `short_db::isolate_shorts` command when *tcl_script* modal argument is used, then there is an additional section of the transcript like this:

```
Name Pads Selected for Short Isolation for cell  "<cell>"  
-----  
  
WARNING:  Short circuit - Different names on one net:  
          Net Id:  <n>  
          (1)  name  "<label>"  at location  (<x>,<y>)  on layer  <n>  
          "<name>"  
          (2)  name  "<label>"  at location  (<x>,<y>)  on layer  <n>  
          "<name>"  
          The name "<label>" was assigned to the net.
```

which reads similarly to the usual Extraction Errors and Warnings report in the transcript. But the difference is that the shorts have been filtered by net name specifications.

When the SVDB directory produced by this option is loaded into Calibre RVE, the short isolation results presentation is the same as for the Mask SVDB Directory SI keyword. That is, interactive short isolation is always enabled.

Either a *rule_file_name* argument or the **-svdb** option must be specified with **-si**. If *rule_file_name* is specified, then the following apply:

- A Mask SVDB Directory statement that generates a PHDB must be specified in the rules. Some of the keyword options that do this are CCI, PHDB, QUERY, and SI.
- Only Connect and Sconnect operations that provide connectivity to Device pin layers are executed by default. This can be altered by specifying the **LVS SI Select Connects** statement, in which case only the Connect and Sconnect operations corresponding to that statement's settings are executed.

Alternatively, if **-si=tcl_script** is specified, then the `short_db::select_connects` command may be used in the Query Server script to produce the same effect.

- Only short isolation is performed during connectivity extraction. None of the other modules like device recognition or netlist extraction are performed, regardless of rule file settings.

The **-svdb** option behaviors are discussed separately.

By default, the maximum number of isolated shorts using this option is 500. This can be changed using the **LVS Maximum Short Results** statement.

A Calibre nmLVS Reconnaissance license is required. A Calibre nmLVS-H hierarchical license pair is required if *rule_file_name* is specified. A Calibre Query Server license is required if **-si=tcl_script** is used. See the *Calibre Administrator's Guide* for complete license information.

See [Example 3](#) and [Example 4](#).

- **-hyper [remote]**

Specifies hyperscaling mode, which improves hardware utilization and run times. This option may only be specified with **-si ... -turbo** and is incompatible with **-svdb**. In MTflex mode, this option is best used with 16 or more remote CPUs.

The remote option triggers remote pseudohierarchical database technology. This causes pseudo HDBs to be distributed across remote hosts in a MTflex run. There should be a minimum of four remote hosts when using this option, with 16 or more CPUs being optimal. For more details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

- **{ -remote *host,host,...* } | { -remotefile *filename* } | { -remotecommand *filename count* } [-remotedata [-recoverremote | -recoveroff]]**

Specifies to use the MTflex multithreaded parallel processing architecture. It can only be specified with **-si ... -turbo**. These options are defined under “[Calibre nmLVS and Calibre nmLVS-H Command Line](#)” on page 45, and the behaviors are largely the same for Calibre nmLVS Reconnaissance applications. Certain runtime details are discussed in the “[Description](#)” section.

For general MTflex details, see “[Command Line Options Reference Dictionary](#)” in the *Calibre Administrator’s Guide*.

- ***rule_file_name***

A required argument that specifies an LVS circuit extraction rule file with Device statements is assumed. [Mask SVDB Directory](#) must be specified, and the SI keyword is implicitly activated if it is not specified in the rules. This argument only applies to the **-si** option.

- **-turbo [*number_of_processors*]**

An option that enables multithreaded parallel processing. The *number_of_processors* is a count of CPUs to use on the primary host. In general, you should avoid specifying *number_of_processors*, and you should never specify 1.

See “[Licensing for Multithreaded Operations](#)” in the *Calibre Administrator’s Guide* for additional information about license scaling with CPU count.

- **-turbo_all**

An option used with **-turbo** that halts the run if the tool cannot obtain the exact number of CPUs you specified using **-turbo**. Without **-turbo_all**, the Calibre tool normally uses fewer threads than requested if the requested number of licenses or CPUs is unavailable.

Specifying the **-turbo_all** option without *number_of_processors* is effectively the same as specifying the maximum number of CPUs on the machine.

Description

Initiates a Calibre nmLVS Reconnaissance run. The supported run modes are stand-alone ERC, soft connection checking, and text short isolation. These runs are separate from typical LVS

runs. The reconnaissance modes can result in more efficient flows allowing various aspects of LVS verification to be run in parallel.

The **-erc=svrf_script** option causes the SVRF code in the *svrf_script* to generate derived layers in the SVDB. Using the [DFM RDB](#) operation in the *svrf_script* is the recommended way to output layer results.

The **-erc -exec tcl_script** argument set causes Query Server Tcl shell or YieldServer commands in the *tcl_script* to be executed on data in a Mask SVDB Directory (the SI keyword must have been specified when the database was created). In this case, the SVDB is either loaded with the **-svdb** option, or it is loaded using the [qs::open_svdb](#) command in the *tcl_script*. In this usage, the **-exec** option must be the final one on the command line. For more information about Query Server Tcl shell functions, see “[Query Server Tcl Shell Command Summary](#)” in the *Calibre Query Server Manual*.

The **-softchk** option initiates a stand-alone soft-connection checking run. Soft-connection checks are performed just like LVS Softchk, but on the data in the *svdb_dir*. By default, the results output is as if LVS Softchk LOWER had been specified in the rule file, and checks are performed on any stamped layers for which there was an Sconnect stamping conflict during circuit extraction, as indicated by this warning:

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

If the *=tcl_script* suffix is used, then the commands in the Tcl script are executed. The [qs::softchk](#) and [qs::softchks](#) commands are useful in this context. The output is a *softchk.rdb* database, which can be loaded into Calibre RVE like any other ERC database.

When the **-svdb** option is used, the *svdb_dir* must have been produced by a run that included the SI keyword of the Mask SVDB Directory statement. If this condition is not met, then this error is given:

```
Error 302: ERROR: SVDB not produced using Mask SVDB Directory SI.
```

The **-si** option initiates a stand-alone short isolation run. Addressing short isolation discrepancies before other verification tasks that depend on valid connectivity makes for a more efficient overall process. Therefore, running **-recon -si** early on after full chip assembly and fixing any shorts can improve overall turnaround time for ERC checks and LVS comparison.

If MTflex distribution is used in a **-si** run, the primary or local host distributes the shorts to the remote processes and then collects the results from the remotes. By default, each remote process is assumed to consume no more than 100 GB of memory. For example, a remote machine with 1 TB of memory will have at most 10 remote processes for distributed short isolation. This distribution strategy can be adjusted using the new Tcl Query Server command [short_db::set_distribution_configuration](#).

If a remote machine does not have the requested resources, it will be skipped and one of the following messages is written in the transcript:

```
<remote-host-name> has 3 CPUs (16 CPUs are requested) - It will not be  
used in distributed SI.
```

or

```
<remote-host-name> has 50000 MB of memory (100000MB minimum is requested)  
- It will not be used in distributed SI.
```

If no remote hosts can be found to satisfy the criteria, all processes are executed locally, and the following message is issued:

```
Remote adjustment failed: executing SI on primary.
```

If a remote process fails to complete the analysis of a short, the analysis is re-started on the primary host. The results of locally restarted shorts are in the following RDB files:

```
./<SVDB>/<TOP_CELL>.phdb/isi/<TOP_CELL>-default-sn<N>.shorts
```

The time and memory statistics of the overall run are printed to the transcript in the following format:

```
<process ID> <host name> <CPU time> <real time> <lvheap> <malloc> <elapsed  
time>
```

Because this mode generates multiple results in parallel, it is possible for the number of results to exceed the limit prescribed in [LVS Maximum Short Results](#). In addition, the short serial numbers found in the RDB files may not match between distributed and non-distributed runs.

Integers are represented internally in sign-magnitude or twos-complement format. The maximum database coordinate precision range is +/- 2⁶¹.

Calibre nmLVS applications exit with a non-zero status if they cannot complete any form of requested processing due to fatal error conditions. In most Linux shells, you can check this status by examining the \$? shell variable.

Examples

Example 1

This example demonstrates Calibre nmLVS Recon ERC.

Assume that Mask SVDB Directory with the SI keyword has produced a PHDB in a previous circuit extraction run. Also assume that LVS Power Name and LVS Ground Name appear in the

rules that generated the SVDB. Assume you have a file called *erc.svrf* that contains the following:

```
floating = PATHCHK !POWER && !GROUND
erc.rules {
    DFM RDB floating erc.db CHECKNAME "no_supply_path" ALL CELLS CELL SPACE
}
```

The following command line executes the *erc.svrf* rule script:

```
calibre -recon -erc=erc.svrf -turbo -svdb svdb
```

The floating layer is output to the svdb directory. The results are output to the *erc.db*, which can be opened in Calibre RVE.

Example 2

This example demonstrates Calibre nmLVS Recon Softchk.

Assume you have a Mask SVDB Directory that was produced using the SI keyword during circuit extraction. The following command line isolates soft connections on any Sconnect stamped layers (the second layer in the Sconnect syntax) for which there was a stamping conflict during circuit extraction:

```
calibre -recon -softchk -turbo -svdb svdb
```

This produces output that is the same as for LVS Softchk LOWER.

Example 3

This example demonstrates Calibre nmLVS Recon SI stand-alone short isolation using SVDB input.

Assume the LVS rule file contains a Mask SVDB Directory svdb QUERY SI statement. Then, the following is run:

```
calibre -lvs -hier -turbo -hcell hcells rules
```

If this is subsequently run:

```
calibre -recon -si -turbo -svdb svdb
```

then the SVDB is updated with interactive short isolation data from the top level of the design, which can be accessed in Calibre RVE or post-processed in the Query Server Tcl shell.

If this is subsequently run:

```
calibre -recon -si=all -turbo -svdb svdb
```

Then short isolation is run at all levels of the hierarchy of the PHDB design data.

Example 4

This example demonstrates Calibre nmLVS Recon SI stand-alone short isolation using rule file input.

Assume this command line and a rule file containing Mask SVDB Directory svdb QUERY:

```
calibre -recon -si=IO -turbo rules
```

Then this statement is applied during short isolation:

```
LVS ISOLATE SHORTS CELL PRIMARY && !NAME LVS_POWER_NAME LVS_GROUND_NAME
```

which evaluates non-supply nets at the top level of the design.

If the rules contain this:

```
LVS ISOLATE SHORTS YES CELL PRIMARY && NAME LVS_POWER_NAME LVS_GROUND_NAME  
CELL ALL BY LAYER BY CELL
```

Then CELL ALL BY LAYER BY CELL are additionally applied because they do not conflict with the command line, but NAME is overridden by !NAME because the command line behavior takes precedence.

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 Calibre CB license. You use this license to run DRC, LVS, Calibre RVE, or the Query Server. You can only run DRC and LVS in flat mode. This license cannot be used to form a hierarchical pair.

See the [Calibre Administrator's Guide](#) for licensing information.

Calibre Version Information

You can use the -version command line option to view the software version. The header of a Calibre run transcript also shows the software version.

The CALIBRE_HOME environment variable determines the location of your Calibre software tree (refer to “[Setup and Tool Invocation](#)” on page 31 for information on setting this variable).

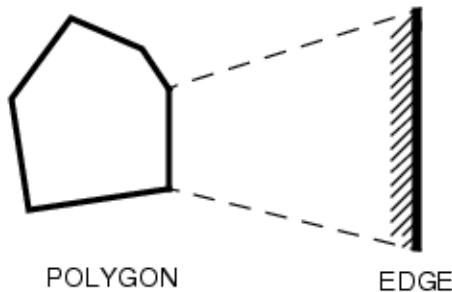
Chapter 3

Layout Processing Concepts

Layout designs consist of shapes, which are, in turn, comprised of edges. Calibre tools work primarily with edges. 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 layer connectivity in the rule file.

Figure 3-1 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.

Figure 3-1. Edge-Polygon Relationship



Layers	78
Layer Operations	89
Hierarchical Database Construction	93
Hierarchy-Specific Statements	94
Layout Input Control	96
Supported Geometric Layout Formats	102

Layers

A geometric layout database consists of layers corresponding to materials used in chip manufacturing. The layers contain various objects like polygons, paths, text objects, and properties. There are four types of layers that Calibre tools process.

Figure 3-2 shows examples of these layer types:

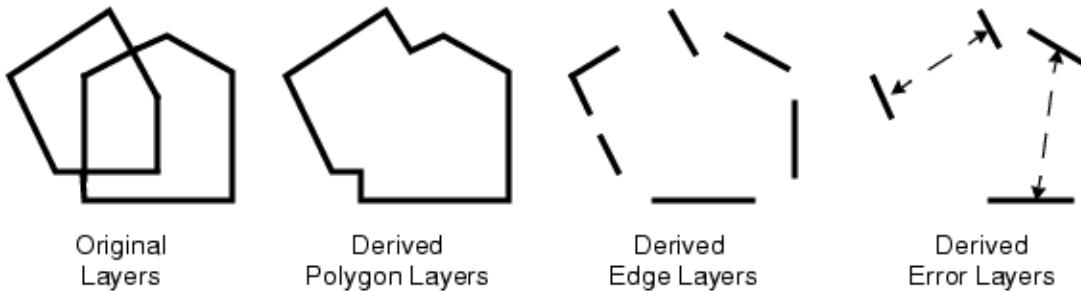
Original Layers (drawn layers)

Derived Polygon Layers

Derived Edge Layers

Derived Error Layers

Figure 3-2. Layer Types



Original Layers

Original layers (or drawn layers) are layers that represent original layout data. Original layers have assigned numbers, datatypes, and texttypes. In a rule file, layers may be referenced by numbers, but more frequently the numbers are mapped to names by using Layer statements.

A layer name to number mapping may be one-to-one, and when the rule file references the layer name, that layer is a *simple layer*.

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, the tool automatically merges the polygon data on an original layer before using that layer. In a merged-data representation, any original polygons that overlap or

share an edge have been merged 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. Merged data also reduces the number of objects that the Calibre database must store.

Datatypes and Texttypes

Calibre does not use datatypes or texttypes for its internal database layers. All layout database layers (with or without datatype or texttype) are represented using a layer number within the Calibre database.

If the layout database format uses datatypes, then all datatypes for a given layer are mapped into a single Calibre layer by default. For example, if you have layer 1 with datatypes 1 and 2 in your layout, these are all mapped to Calibre layer 1 internally. No datatype is used.

Processing of drawn geometry and text based on datatypes and texttypes is possible through [Layer Map](#) specification statements. As with all rule file elements, there are no restrictions on the relative ordering of the Layer and Layer Map specification statements. Using the previous example, you could map the datatypes this way in the rule file:

Example 3-1. Mapping Layer Datatypes to Calibre Layers

```
LAYER MAP 1 DATATYPE 1 1
LAYER MAP 1 DATATYPE 2 2
LAYER my_layer1 1
LAYER my_layer2 2
```

In this example, layer 1 datatype 1 is mapped to Calibre layer 1. Layer 1 datatype 2 is mapped to Calibre layer 2. Then the Calibre layer numbers are given the names my_layer1 and my_layer2.

When the DATATYPE keyword is present, then a Layer Map specification statement is called a datatype map. 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 non-empty. For example:

```
LAYER metal 100
LAYER MAP 6 DATATYPE 1 100
```

The intent is that any layout geometry on layer 6, datatype 1, is mapped to Calibre layer 100 and given the name metal. Problems can arise if layer 100 in the input layout database contains geometry that is not intended to be used, but which is placed on Calibre layer 100 per the mapping algorithm described previously. A simple solution is to map input objects on layer 100 to an unused layer as follows:

```
// map database layer 100 (all datatypes) to unused Calibre layer 999
LAYER MAP 100 DATATYPE >= 0 999
LAYER unused 999

LAYER MAP 6 DATATYPE 1 100 // layer 6 datatype 1 goes to Calibre layer 100
LAYER metal 100
```

Because layer maps are not always carefully designed, a warning is issued for any input layer that meets these criteria:

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

When the Layer Map TEXTTYPE keyword is present, it is called a texttype map, which is used for mapping text layers. A TEXTTYPE map works in a similar way to DATATYPE maps. Texttype maps apply to text objects that exist on a [Layout Property Text](#), [Layout Text](#), [Text Layer](#) (and in LVS, [Port Layer Text](#), and [Device](#) TEXT MODEL LAYER and TEXT PROPERTY LAYER parameters) in the same way that datatype maps work for datatypes and original layers defined in Layer specification statements.

Texttype attributes of text objects are retained throughout the Calibre flow, so if a text object's layer number is mapped in a TEXTTYPE map but not the texttype number, its original texttype attribute is maintained. The texttype can also be mapped to a different type using the Layer Map statement.

Related Topics

[Layer Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Layer Definitions

Derived layers are layers produced by layer operations in the rule file. *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 the layer's name.

A layer definition has the form:

layer_name=layer_operation

The *layer_operation* creates the derived layer and gives it the name *layer_name*. See “[SVRF Rule File Format](#)” in the *SVRF Manual* for a summary of element definitions that may appear in a rule file, such as layer operations and rule checks.

Here is a sample rule file excerpt illustrating layer definitions:

Example 3-2. Layer Definitions

```
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
p_tap = p_diff OUTSIDE n_well    // p-tap areas
```

These layer definitions are global because they do not appear in the context of a rule check. The names of global derived layers may be used anywhere in the rule file. You cannot declare a global layer name more than once in a rule file (or in rule files that appear in an [Include](#) statement).

A layer definition may also appear inside a rule check, like this:

```
rule1 {
    gate = poly AND diff      // layer definition
    LENGTH gate < min_gate_L // free-standing layer operation
}
```

A layer definition that is inside of a rule check statement is local to that statement and overrides a global layer definition of the same name. 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.

As shown in the previous example, a layer operation may be *free-standing* (that is, there is no equals sign and no assignment to a name) inside a rule check. A free-standing layer operation only sends output to the results database. Free-standing layer operations may not be used outside of a rule check.

Duplicate Layer Definitions

You can create duplicate layer definitions having different names. For example, you can do this (the layer operation is identical in both cases):

```
Alias1 = <operation> layerA
Alias2 = <operation> layerA
```

These layer definitions are duplicates when the layer operation and the input layer are the same.

The Calibre database constructor is efficient in that it uses only one layer internally for duplicate layer definitions. So in the case shown previously, Alias1 and Alias2 are actually the same layer within Calibre, but you can access that layer using either name.

This can have important implications for things like connectivity, because the connectivity model for all the aliases for the same layer definition is the same. See “[Node-Preserving Layer Operations](#)” on page 90 and “[Node Preservation and Duplicate Layer Definitions](#)” on page 91 for additional information on this topic.

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 specify 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:

Example 3-3. Implicit Layer Definitions

```
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.

Related Topics

[Layer Operations](#)

Derived Polygon Layers

Derived polygon layers represent merged polygons generated as the output of Boolean layer operations, topological polygon operations, and certain dimensional check operations that employ polygon output.

This is an example of a Boolean operation with derived polygon output:

```
gate = poly AND oxide
```

The gate layer is contains polygons that are the intersection of poly and oxide layer polygons.

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), the database constructor calculates the layer only once and uses it wherever needed during the run. Any derived layers that are determined to be identical, even though the layer names differ, are calculated and stored only once.

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

```
contact NOT INTERACT metal
```

with this being a full rule check construct:

```
bad_contact { contact NOT INTERACT metal }
```

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 or edge-directed dimensional check operations. Derived edge layers can also produce results output if they occur in a free-standing layer operation.

An example of a derived edge layer is this:

```
long_metal_edge = LENGTH metal > 5
```

Whether used globally or locally (or both), the database constructor calculates the layer only once and uses it wherever needed. Any derived layers that are determined to be identical, even though the layer names differ, are calculated and stored only once.

The free-standing operation:

```
LENGTH metal > 5
```

generates results output that is sent to the results database when this operation is placed in a rule check, such as here:

```
long_metal { LENGTH metal > 5 }
```

Derived Edge Orientation

Derived edges have inside-facing sides and outside-facing sides. An inside-facing side of an edge is the side that faces the interior of a polygon from which the edge originates. An outside-facing side of an edge is the side that is not inside-facing. The notion of inside-facing and outside-facing sides of derived edges is maintained by the Calibre layout database. This is

important for layer operations for which inside and outside are considered when processing edges.

This shows an example of where edge “facing” orientation is important:

```
// edges forming the width of layer "a"  
// m is a variable width  
a_width = INTERNAL [a] < m OPPOSITE  
// move edges left 0.25 um  
a_shift_L = GROW a_width LEFT BY 0.25
```

Edges on a_width occur in pairs. If such a pair is aligned vertically, for one edge of the pair, its exterior side faces to the right, and for the other edge, its exterior faces to the left. The layer a_shift_L contains only the outputs from edges whose exterior-facing side faces to the left.

This type of sensitivity to edge orientation can be manifested by any layer operation that processes derived edge layers.

The [OR Edge](#), [DFM OR Edge](#), and [DFM Copy](#) operations warrant particular care in this context because they can be used to merge edge layers. If these operations are used without regard for edge orientation, they can derive layers with unexpected output. This might only be noticed in the context of some downstream layer operation that uses the output of one of these three. If this occurs, using DFM OR Edge with the BOTH keyword to merge edge layers may provide the desired output.

See “[Appropriateness Criteria](#)” on page 177 for a detailed discussion of this topic in the context of dimension check operations.

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* layer operations for instantiation into the DRC results database; although other results databases can output derived error layers as well.

Following are some of the more commonly-used operations that can generate derived error layers:

[DFM Copy](#) (with the CLUSTER keyword)

[DFM DV](#)

[DFM Measure](#)

[DFM Property](#)

[DFM Property Select Secondary](#)

[DFM Shift Edge](#)

[DFM Space](#)

[Drawn Acute](#)

[Drawn Angled](#)

[Drawn Offgrid](#)

[Drawn Skew](#)

[Enclosure](#)

[External](#)

[Internal](#)

A free-standing error-directed layer operation produces a derived error layer that is sent to the DRC results database when the operation is placed inside a rule check:

```
min_ml_width { INTERNAL m1 < 0.08 }
```

This syntax generates a derived error layer that is stored in layer x:

```
x = INTERNAL m1 < .08 // derived error layers have limited used
```

Note

 Not all layer operations take error-directed results as input. Following are some of the more commonly-used operations that accept derived error layers as input:

- [DFM Analyze](#)
 - [DFM Copy](#)
 - [DFM DP](#)
 - [DFM Expand Edge](#)
 - [DFM MP](#)
 - [DFM OR Edge](#)
 - [DFM Property](#)
 - [DFM Property Select Secondary](#)
 - [DFM RDB](#)
 - [DFM Shift Edge](#)
 - [RET NMDPC](#)
-

Dimensional check operations use the [] and () operators around their layer arguments to generate derived edge layers instead of derived error layers. For example, this syntax generates a derived edge layer that can be used as input to any other layer operation:

```
y = INTERNAL [m1] < 0.08
```

Note the use of the brackets ([]) in this syntax. This derivation does not have the limitations of derived error layers.

The ENClosure, EXTernal, and INTernal operations use keywords that begin with the name REGION to generate derived polygon layers. Dimensional check operations are discussed in detail in “[Dimensional Check Operations](#)” on page 153.

Unmerged Polygon Layers

Several layer operations generate unmerged polygon layers or support unmerged derived polygon layers as input to the operation. This is not true for most layer operations.

The following table lists the operations that support derived polygon layers that are unmerged. The table notes whether unmerged layers are supported as input or output, or both. The table does not refer to handling of original polygon layers, edge layers, or error layers.

Table 3-1. Support of Unmerged Derived Polygon Layers

Operation	Input	Output	Comment
DFM Analyze	No	Yes	Output: The _detail layer written to the DFM database contains unmerged geometries.
DFM Copy	Yes	Yes	<p>Input: Specify UNMERGED for unmerged polygon layer input.</p> <p>Output: Unmerged polygon output is generated with REGION UNMERGED, or when the input layer is an unmerged polygon layer.</p>
DFM Expand Edge	No	Yes	Output: Specify UNMERGED to generate unmerged polygon output.
DFM OR Edge	Yes	No	Input: No keyword needed.
DFM Property	Yes	Yes	<p>Input: Specify UNMERGED for unmerged polygon layer input.</p> <p>Output: Unmerged polygon layer output is generated if an unmerged polygon layer is the primary layer.</p>
DFM Property Merge	Yes	No	Input: No keyword needed.

Table 3-1. Support of Unmerged Derived Polygon Layers (cont.)

Operation	Input	Output	Comment
DFM Property Select Secondary	Yes	Yes	<p>Input: Specify UNMERGED for unmerged polygon layer input.</p> <p>Output: If the SELECT layer is a unmerged polygon layer and no layer derivation is used on the SELECT layer, the output is also an unmerged polygon layer. If the :EXTENT layer derivation is used on the SELECT layer, the output is an unmerged polygon layer.</p>
DFM RDB	Yes	No	Input: No keyword needed.
DFM Read	No	Yes	Output: No keyword needed.
DFM Text	No	Yes	Output: No keyword needed.
Inside Cell	No	Yes	Output: Unmerged when PRIMARY ONLY is specified.
LVS Isolate Shorts	No	Yes	Output: When UNMERGED is specified in a hierarchical run, the output to the short isolation database is not merged.
Not Inside Cell	No	Yes	Output: Unmerged when PRIMARY ONLY is specified.
Ornet	No	Yes	Output: Overlapping polygons that are not on the same net are not merged.

Layer Type Summary

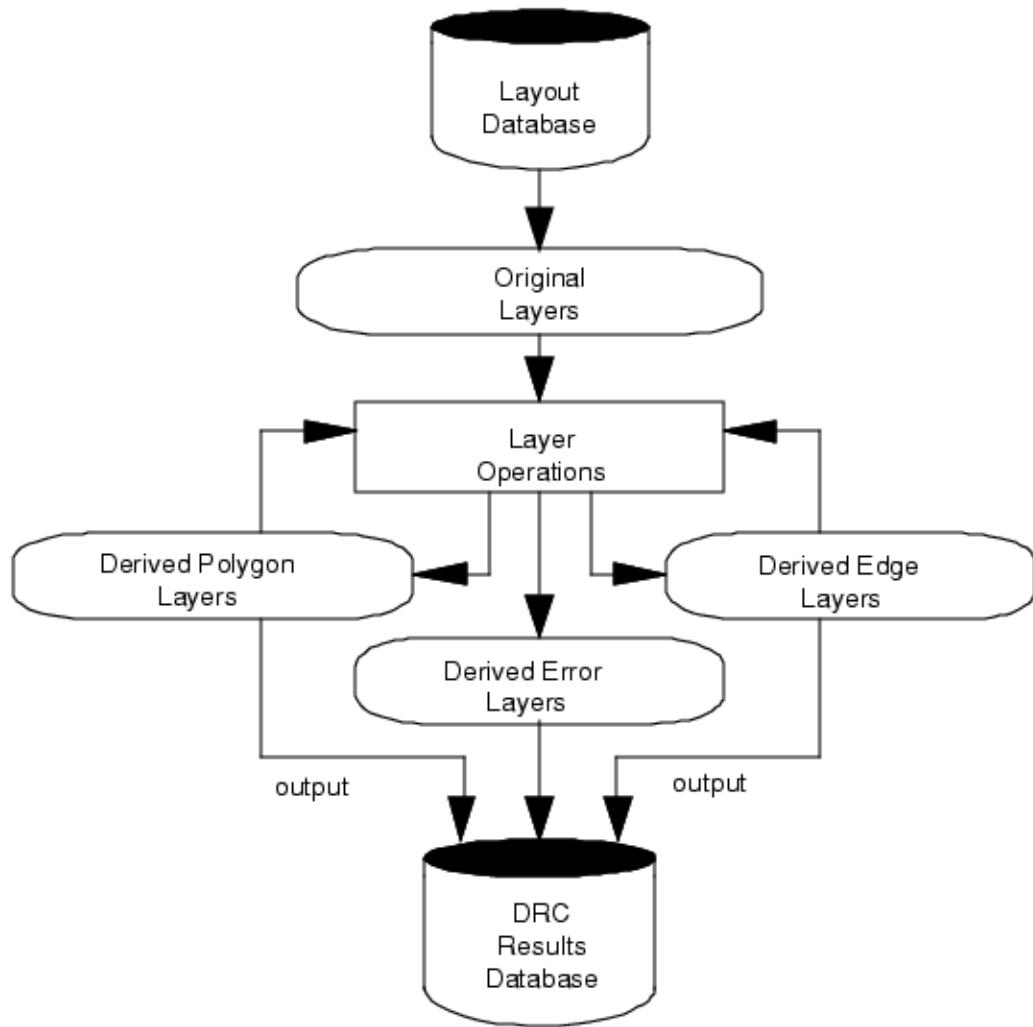
To summarize, derived polygon and derived edge layers can be used as output to the DRC results database or in layer definitions that are used for further processing in other layer operations.

The ENClosure, EXTernal, INTernal, DFM Measure, DFM Property, 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 (derived edge, derived polygon, or derived error) can represent output from the Calibre nmDRC system, although only the third type is actually called an *error layer*.

Figure 3-3 shows the global flow of data through the Calibre system, in terms of the layer types supported by the system.

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



Layer Operations

The basic unit within a Calibre 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 fundamental types of layer operations:

- **Dimensional Check Operations** — Dimensional check operations provide the core design-rule checking capability of the DRC system. The principal ones are ENClosure, EXTernal, and INTernal. The [Rectangle Enclosure](#), [\[Not\] With Neighbor](#), [TDDRC](#), and [DFM Measure](#) operations also fit into this category.

Dimensional check operations measure distances between various types of edges. Note that the ENCclosure, EXTERNAL, INTERNAL, and TDDRC operations are error-directed by default, but have special edge- and polygon-directed syntaxes that produce derived layer data that can be used as input by other operations.

- **Auxiliary Operations** — Any layer operation that is not a dimensional check operation 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 of operations are used to derive polygon or edge layers, depending on the function of the operation. Both types can be used to send data to a 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.

Layer Constructors and Selectors	89
Node-Preserving Layer Operations	90
Layer of Origin	92

Layer Constructors and Selectors

Layer operations that generate derived edge or polygon layers are either layer constructors or layer selectors. The following sections describe these layer classifications.

The “[Auxiliary Layer Operations](#)” section in the *SVRF Manual* shows which auxiliary operations are constructors and which are selectors. Dimensional check operations can function in either role, depending on the keywords you specify.

Layer Constructors

Operations classified as layer constructors create new polygon or edge data that do not exist in the original layout.

For example, a two-layer Boolean AND operation:

```
layer1 AND layer2
```

is a layer constructor because it creates new polygons from the intersections of both input layers.

The layers with a superscript “c” in the “[Auxiliary Layer Operations](#)” section in the *SVRF Manual* are layer constructors.

Layer Selectors

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

Here is an example:

```
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.

The layers with a superscript “s” in the “[Auxiliary Layer Operations](#)” section in the *SVRF Manual* are layer selectors.

Node-Preserving Layer Operations

The connectivity model for the design is constructed in a single step *before any operation that requires or passes connectivity*. (DRC Incremental Connect YES changes this behavior in DRC). The connectivity model is built from all Connect and Sconnect statements in the rule file. From this initial connectivity model, node IDs can be passed to derived layers through node-preserving layer operations.

A node-preserving layer operation (also called net-preserving) passes connectivity from the first input layer to the derived layer. This occurs when nodal information is available for the appropriate input layer. For example:

Example 3-4. Node Preservation by Layer Operation

```
CONNECT layer1
layer2 = layer1 AND layer3 // layer2 inherits layer1 node IDs
```

If nodal information is not available from an input layer on the right side of the equals sign, then no nodal information gets passed.

If a derived layer generated by a node-preserving operation appears in a [Connect](#) statement (but not a BY keyword parameter), or as the lower layer parameter of an [Sconnect](#) statement, then

the node-preserving operation passes no connectivity. This is because the operations must be performed first in order to establish the connectivity model. For example:

```
CONNECT a b
b = c AND d
```

The AND operation must be performed before the CONNECT statement, so the AND operation does not pass connectivity in this case. This also prevents layers from being shorted together unexpectedly.

All [Layer Selectors](#) are node-preserving because they select polygon or edge data from a *single* input layer, and the connectivity is unambiguous in this case. For example:

```
CONNECT metall poly BY cont
x = poly COINCIDENT EDGE metall // x receives the node ID of poly
```

Because poly is the first layer in the layer selector operation, its nodal information is passed to the output layer. For some layer operations, reversing the order of the input layers does not affect the geometric output. However, this does affect the nodal information that gets passed to the output layer.

Dimensional check operations (ENClosure, EXTernal, and INTernal) can pass connectivity information if edge-directed output is used (the default is error-directed). Again, node-preserving operations usually pass connectivity from the first input layer (or the only input layer in some cases) to the derived layer. However, the dimensional check operations can pass connectivity from the second input layer if the appropriate edge-directed syntax is used.

With few exceptions, [Layer Constructors](#) are not node-preserving. The [Ornet](#), two-layer [AND](#), and the two-layer [NOT](#) (all layer constructors) are node-preserving. The Ornet operation passes connectivity information from both input layers to the derived layer. The AND and NOT operations pass connectivity from the first input layer to the derived layer.

Node Preservation and Duplicate Layer Definitions

Duplicate layer definitions actually represent the same layer within the Calibre database. This has important implications for node preservation.

Assume the following rule file excerpt, where the operation could also be a single-input operation:

```
Alias1 = layerA <operation> layerB
Alias2 = layerA <operation> layerB // Alias1 and Alias2 are the same!

CONNECT Alias1 layerC
CONNECT Alias2 layerD // All of these layers are now shorted together

TEXT LAYER txt
ATTACH txt layerC
```

In this case, the layers Alias1 and Alias2 now have the same connectivity information and represent the *same layer* in the Calibre database. The consequence being, layerC and layerD are now connected.

This is a designed behavior because Calibre is efficient about storing duplicate layers. Since Alias1 and Alias2 are really the same layer, they are treated as names for the same thing.

If you want to avoid the unintentional shorting of layers caused by the preceding example, you can do this:

```
Alias1 = layerA <operation> layerB
Alias2 = layerA AND Alias1
```

Now the CONNECT operations shown previously will have the desired effect.

Related Topics

[Layer Definitions](#)

[Connect \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Incremental Connect \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Sconnect \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Layer of Origin

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 89 for a description of this class of operations.

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 N, then the layer of origin of N is the original layer from which it was initially derived.

For example, consider these layer definitions:

```
v = LENGTH metal > 10           // selector, metal is an original layer
w = WITH EDGE metal v           // selector
x = SIZE w BY 0.10 UNDEROVER   // constructor
y = AREA x < 4                 // selector
z = ENCLOSE y via              // selector
```

The layer of origin for layer z and layer y is layer x, because it is the last layer in the derivation chain generated by a layer constructor operation. The layer of origin for layers v, w, and x is metal.

An important point regarding layer selector operations (see “[Layer Selectors](#)” on page 90): these operations select data from the appropriate input layer. It may appear, for example, that these layer definitions:

```
x = a COINCIDENT EDGE b
x = b COINCIDENT EDGE a
```

produce the same data in the layer x. Another way of saying this is that it may appear that the Coincident Edge operation is commutative. Although the operation generates the same geometric edge data in both cases, it is also true that polygon association and node references are passing through the selector operation. Hence, seemingly identical layer data selected from ‘a’ and ‘b’ are not, in general, the same data because polygon association and node references likely differ.

This has important implications for dimensional check operations. For example, the following [Internal](#) dimensional check sequence:

```
x = metal coincident edge poly
ruleA {internal x metal < 3}
```

has an entirely different definition from this sequence:

```
x = poly coincident edge metal
ruleB {internal x metal < 3}
```

In the first case, layer x contains geometric data carrying polygon number information from layer metal. In the second case, this data carries polygon number information from layer poly. That is, in the first case, the layer of origin of the input layers to the INTernal operation is the same (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.

Hierarchical Database Construction

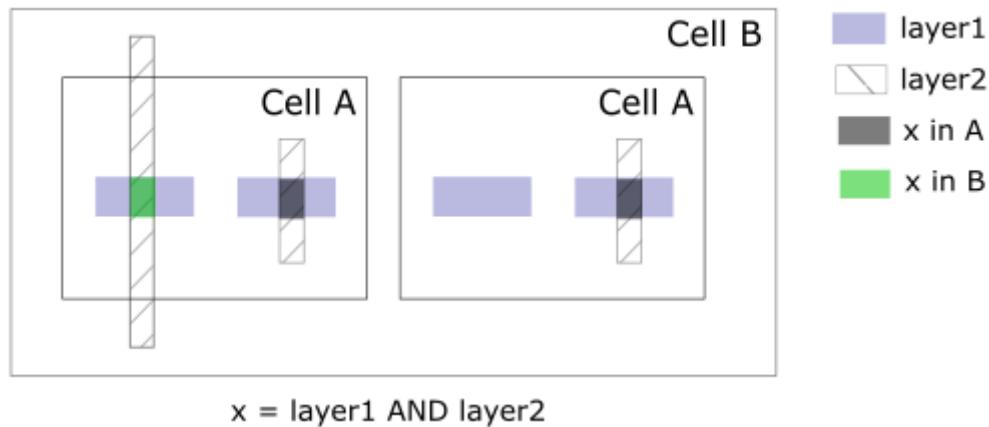
Flat verification applications work from a flat database representation only. For geometric layout input, flat Calibre tools expand the layout database to the top level at the start of a run and maintain no record of the original database hierarchy. Hierarchical database construction on the other hand leverages the hierarchy of the original design.

The Calibre hierarchical database constructor 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. The Calibre representation of a design’s hierarchy can have differences versus the original layout. Calibre hierarchy modifications optimize performance.

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 3-4 provides a simple illustration of the idea behind hierarchical processing.

Figure 3-4. Hierarchical AND Operation



Viewed flat, layer2 intersects layer1 three times. Viewed hierarchically, one of the intersections occurs exclusively in cell A, and the other occurs between shapes in cells A and B. The right-most shape on layer x is stored once in cell A, irrespective of the fact that cell A is placed twice. The left-most layer x shape is promoted to cell B to ensure an accurate AND operation in a hierarchical context.

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 hierarchical versions of the Calibre tools. 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. This has implications for the Calibre representation of design hierarchy versus the original layout data.

Hierarchy-Specific Statements

Certain specification statements only apply when the layout design has hierarchy. Such statements are ignored when the tool is run flat.

The specification statements [Expand Cell](#), [Flatten Cell](#), [Flatten Inside Cell](#), [Layout Base Cell](#), [Layout Base Layer](#), [Layout Top Layer](#), and [Push Cell](#), are used exclusively in hierarchical applications.

Expand Cell expands all placements of the specified cells one level into the hierarchy of their parents. Flatten Cell expands all hierarchy within placements of the specified cells into the hierarchy of their parents. Flatten Inside Cell expands all hierarchy within the specified cells, but placements themselves are not expanded. These statements have limited use, but can improve performance in (temporary) situations where lack of automatic expansion of certain placements can be identified as negatively impacting performance.

Layout Base Layer (or, alternatively, Layout Top Layer) is highly recommended for all hierarchical applications. 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 takes precedence over Layout Top Layer if both appear in the same rule file.

Be careful when specifying the layers for these statements, as mistakes can cause substantial runtime penalties.

Layout Base Layer enumerates a set of device-level original layer names. You should normally not specify other layers than in this example:

Example 3-5. Layout Base Layer Specification

```
LAYOUT BASE LAYER polysilicon diffusion contact
// do not use implant, via, or metal layers in most cases.
```

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).

Layout Input Control

Input layout data is pre-processed by the Calibre tools to optimize performance and results debugging efficiency. Some of the pre-processing is automatic, and a user has no control over it. Other pre-processing steps can be specified in the rule file.

Specification of the layout is made using the [Layout Path](#), [Layout Primary](#), and [Layout System](#) statements in the rule file.

Area-Based Filtering	96
Cell Exclusion	96
Cell Renaming	97
Duplicate Cell Processing	97
Database Pre-Merging	97
Input Layout Database Magnification	98
Wildcards in Layout Primary	100
Layout Read Error Tolerance Settings	100
Layout Database End Segment Warning	100
Layout Property Handling	101

Area-Based Filtering

Area-based filtering of the layout constrains the output of the tool to particular regions of the design.

The following specifications statements support area filtering of layout database objects: [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.

Cell Exclusion

All hierarchical 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.

You can control cell exclusion through the use of [Exclude Cell](#) specification statements. These statements 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 ASCII input layout databases. The wildcard character “*” may be used within the cell names.

Cell Renaming

You may rename cells as the input layout database is being read. This is especially useful in establishing cell correspondences for dual-database capability in hierarchical applications.

The [Layout Rename Cell](#) specification statement is used for renaming cells within the Calibre database.

Duplicate Cell Processing

By default, duplicate layout cells are not allowed. This condition causes a runtime error.

A frequent cause of duplicate cell errors is reading in multiple layout databases. You can specify the [Layout Path](#) specification statement with multiple filename parameters any number of times. Multiple input databases are treated as if all the structure records were embedded in the first file specified. Multiple cell records having the same name cause an error by default.

Whether or not you specify multiple files 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 when the database is split into multiple files by layer, not by cell.

Database Pre-Merging

A layout database that originates from a module generation program 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 $M \ll N$. 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 $M \ll N$, then this can be reduced considerably to MP. Merging of the flat layer is then much more efficient and memory is saved. You can accomplish this with the [Layout Merge On Input](#) YES specification statement.

The input layout database format must be GDSII or OASIS 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.

Layout Merge On Input allows database pre-merging for specified layers. This can be useful for certain advanced DRC checks that find inter-hierarchical overlapping of layer objects.

Input Layout Database Magnification

Magnification of the input layout database is dependent upon a number of factors including database and rule file precisions and magnifications specified in the rules.

The [Layout Magnify](#) statement specifies that all input layout databases are magnified by the given value as they are being read into the Calibre application. By default, no magnification occurs.

The magnification algorithm simply multiplies all coordinate data (including placement base points and array pitches for GDSII) by the given value, regardless of hierarchical position. This algorithm is 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, the need for greater precision in measurement necessitates a change in the rule file [Precision](#) statement versus the precision of the layout. In such usage, the rule file Precision is modified so that dimensioned quantities in the rule file agree with the new precision. When the rule file Precision does not match the physical precision of the layout database, the layout data is scaled up or down as it is read in, inversely to the change in the rule file Precision. For instance, if you increase your rule file Precision by a factor of 2, the layout data is scaled by 1/2 as it is read in.

When changing your rule file Precision, this usually requires the layout database to be magnified to compensate for the scaling caused by the change in Precision. For example:

Example 3-6. Basic Layout Database Magnification

```
// increase measurement precision by factor of 10
PRECISION 10000
LAYOUT MAGNIFY 10 // compensate for PRECISION change
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 1
```

Next are some formulas for determining the magnification factor to compensate for a given change in rule file precision. Define the following parameters as follows:

- Let P be the *original* rule file Precision
- Let D be the change in the rule file Precision
- Let M be the magnification factor to compensate for the Precision change

In the case of an increase in Precision, the formula for the compensating magnification is as follows:

$$M = 1 + D/P$$

and for a decrease in Precision, the formula is this:

$$M = 1 - D/P$$

M is the value you would use in a Layout Magnify statement.

Assume, for example, your original rule file Precision is 1000 and the physical database precision is 0.001. Assume your target measurement precision is 0.00025. This means your final rule file Precision would be 1/0.00025, or 4000. The change in Precision is 3000. Using the formula shown previously, your compensating magnification would be given by this:

$$M = 1 + 3000/1000$$

which is 4. This example is straightforward.

In some cases things are more complex. Assume your original rule file Precision is 4000 and the physical database precision is 0.00025. Assume your target measurement precision is 0.00018. This means your rule file Precision would be equivalent to 5555.6 rounded to five significant figures. So the change in precision is 1555.6. Using the formula shown previously, the compensating magnification factor is given by this:

$$M = 1 + 1555.6/4000$$

which is 1.3889. As a quick sanity check, 1.3889 multiplied by 0.00018 is 0.00025, which is the physical precision.

When more than one input layout database is used, you can magnify one of the input layout databases by using the MAG keyword for the [Layout Path](#) and [Layout Path2](#) statements. The MAG keyword behaves exactly the same as Layout Magnify with a value and without the PLACE keyword. The magnification applies only to the layout specified in the Layout Path[2] statement rather than to all layouts that are read in. For example:

```
LAYOUT PATH design.gds MAG 10 //magnify design.gds by a factor of 10
```

Layout Magnify has an AUTO keyword, which causes any layout that is read in to be magnified by the ratio of the rule file [Precision](#) statement and the physical precision of the layout database. This is a convenient feature to use in many cases where the rule file Precision differs from the physical precision.

The Layout Path and Layout Path2 statements also have an AUTO keyword, which behaves exactly as the Layout Magnify AUTO keyword. The AUTO keyword causes the layout specified in the Layout Path[2] statement to be magnified by the ratio of the rule file Precision and the database physical precision.

The [Layout Precision](#) statement is used to specify the expected physical precision of the layout. This statement has no scaling effect on the layout data and has no effect on measurement precision. It is used simply for checking the precision of the input database on read-in.

See “[Layout Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

Wildcards in Layout Primary

The Layout Primary (and Layout Primary2) 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, non-referenced) 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.

Layout Read Error Tolerance Settings

Calibre tools 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. The [Layout Input Exception RDB](#) statement allows you to write a database consisting of exception geometry encountered during database read-in.

The [Layout Error On Input](#) statement offers more coarse control over situations affecting database read-in than Layout Input Exception Severity. In some cases, Layout Error On Input may be sufficient to handle your needs.

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 and OASIS.

This behavior is controlled through the [Layout Input Exception Severity](#) PATH_ENDSEGMENT_SHORT rule file setting.

Layout Property Handling

Calibre automatically reads properties from a GDSII or OASIS input layout database to support. Properties are accessed from these commands:

- [Layout Property Text](#) specification statements.
- [Layout Property Text OASIS](#) specification statements.
- [Layout Property Audit](#) specification statements. (DRC-H only)
- [Inside Cell](#) operations which are required in the flow and which have WITH PROPERTY or NOT WITH PROPERTY keywords.
- [DFM Read](#) operations.

Properties are not read if they are not required as listed above.

Supported Geometric Layout Formats

Calibre tools support a variety of geometric layout formats.

GDSII Layout Format	102
OASIS Layout Format	104
Third-Party Layout Formats	104
ASCII Layout Format	107

GDSII Layout Format

Calibre processes GDSII version 6.0. Later versions are not supported.

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

When the layout database format is GDSII (specified as [Layout System GDSII](#)), you must specify the pathname to the file in a [Layout Path](#) specification statement and identify the highest-level cell name in a [Layout Primary](#) specification statement. The specified primary layout cell and the cells below it in the layout hierarchy are processed.

Per the original Calma specification¹, cell names may consist of the characters A-Z, a-z, 1-9, underscore (_), question mark (?), and dollar sign (\$), but Calibre is not limited to that. The default maximum cell name length for a GDS file is 32 characters. Calibre nmDRC can exceed this through the [DRC Maximum Cell Name Length](#) statement.

The Layout Path statement may be specified with multiple filenames 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.

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.

The following GDSII records are processed by Calibre:

ANGLE	DATATYPE	LIBNAME	STRNAME
AREF	ENDEL	MAG	SREF
BGNEXTN	ENDEXTN	PATH	TEXT

1. Calma Company, *GDSII™ Stream Format Manual*, Release 6.0 (February 1987).

BGNLIB	ENDLIB	PATHTYPE	TEXTTYPE
BGNSTR	ENDSTR	SNAME	UNITS
BOUNDARY	HEADER	STRANS	WIDTH
COLROW	LAYER	STRING	XY

The [Layout Input Exception Severity](#) rule file statement controls how Calibre handles layout input exceptions. The severities (fatal error, warning, or ignore) of most exceptions can be adjusted to your needs.

Calibre cannot preserve a non-orthogonal lattice read from a GDSII database, where non-orthogonal refers to the (X1,Y1)->(X2,Y2) and (X1,Y1)->(X3,Y3) coordinates of the lattice not being perpendicular. When reading GDSII, Calibre checks for the existence of non-orthogonal lattices and expands the array upon stream-in.

BOUNDARY and PATH records with zero vertices generate a fatal read error. PATH records of non-zero width are converted to polygons in the Calibre database and remain in that configuration for all processing. Derived edges are written to GDSII as 0-width paths.

Calibre can read GDSII polygons of up to 8191 distinct vertices (with the first vertex being the same as the last, the count is 8192). If a polygon has more than 8191 vertices, it generates a fatal error. If you have a polygon that exceeds this limit, then either segment it into smaller polygons whose individual vertex counts are less than 8191, or consider switching database formats to OASIS.

Calibre outputs GDSII polygons of up to 8190 vertices by default. However, it can output up to 2^{32} vertices. The [DRC Maximum Vertex](#) rule file statement controls vertex counts for DRC output.

By default, Calibre does not process BOX and BOXTYPE records because they are not intended to be mask data. The same is true of NODE and NODETYPE records. These elements are part of the GDSII specification, but since the advent of more capable computers with larger address spaces, they are infrequently used. Some modern layout tools can produce them, however.

Calibre batch tools 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

Calibre can process NODE and NODETYPE records as BOUNDARY and DATATYPE records, respectively. To do so, place the following statement in the rule file:

LAYOUT PROCESS NODE RECORD YES

The Calibre layout viewers do not process BOX, BOXTYPE, NODE, and NODETYPE records by default. The viewers can be configured to convert them similar to how the batch tools do so.

OASIS Layout Format

Calibre supports OASIS 1.0.

The handling of OASIS data is similar to GDSII in many respects, and most of the specification statements that apply to GDSII also apply to OASIS.

OASIS format has certain advantages over GDSII, the primary one being file size. An OASIS design is typically ten times smaller than the same design in GDSII format. OASIS does not have the same limitations for allowed cell name characters or cell name length as GDSII.

One important difference to note: Calibre reads OASIS polygons having up to 2^{32} vertices, while the GDSII read limit is 2^{12} vertices.

See the [Calibre for the Open Artwork System Interchange Standard](#) guide for more information about OASIS use in the Calibre tool set.

Third-Party Layout Formats

Calibre tools read LEF/DEF and OpenAccess layout formats.

The supported versions are shown in the table “[Layout Database Formats](#)” on page 33. LEF/DEF and OpenAccess are proprietary formats overseen by Silicon Integration Initiative, Inc. (Si2). For questions about these formats, contact Si2.

Processing of third-party layout formats is very similar to the processing of GDS and OASIS layout formats discussed under “[GDSII Layout Format](#)” and “[OASIS Layout Format](#).”

The third-party layout formats are specified in a rule file using the [Layout System](#) statement. The [Layout Path](#) statement needs to specify the design library for OpenAccess systems. See the *SVRF Manual* description of Layout Path for details regarding integration with third-party formats.

Calibre Interactive and the Calibre layout viewers all support these layout formats for input. Calibre tools do not output these formats, however.

[Table 3-2](#) shows the configuration environment variables that Calibre batch tools observe in conjunction with LEF/DEF and OpenAccess databases. Calibre Interactive also observes these environment variables, except for MGC_CALIBRE_LAYERMAP_FILE and MGC_CALIBRE_CELLMAP_FILE.

Table 3-2. FDI Environment Variables

Variable Name	Value	Description
ICC_PATH	<i>pathname</i>	Specifies the directory that includes the <code>icc_shell</code> executable. This environment variable is not required if <code>icc_shell</code> is defined in the current PATH variable.
LD_LIBRARY_PATH	<i>pathname</i>	If using Calibre 2008.3 or later on a Linux machine that is earlier than RHEL 4u6, this variable specifies the pathname of the <code>libstdc++.so.6.0.8</code> compiler libraries, which are required to run the tool. This is for OpenAccess database support. Ensure that the path includes the 64-bit libraries.
MGC_CALIBRE_DB_READ_OPTIONS	<p>“-optionName <i>args</i> [-optionName <i>args</i> ...]”</p> <p>The following options may not be used:</p> <ul style="list-style-type: none"> -design -def -lef -outFile -system 	<p>Sets options for controlling the read-in of third-party databases by Calibre tools that support them. These options are read in addition to any that may be specified on the FDI utility command lines. A list of options for batch use is shown under Layout System in the <i>SVRF Manual</i>.</p> <p>The option list should be quoted and the delimiter between multiple options is the space character.</p>
MGC_CALIBRE_LAYERMAP_FILE	<i>pathname</i>	<p>Sets the pathname of a layer mapping file for use during third-party design database read-in by the batch tool.</p> <p>MGC_CALIBRE_DB_READ_OPTIONS can perform the same function using the <code>-layerMap</code> option.</p> <p>MGC_CALIBRE_LAYERMAP_FILE overrides MGC_CALIBRE_DB_READ_OPTIONS when both are specified.</p>

Table 3-2. FDI Environment Variables (cont.)

Variable Name	Value	Description
MGC_CALIBRE_CELLMAP_FILE	<i>pathname</i>	Sets the pathname of a cell mapping file for use during the Layout System design database read-in by the batch tool. MGC_CALIBRE_DB_READ_OPTIONS can perform the same function using the -cellMap option. MGC_CALIBRE_LAYERMAPFILE overrides MGC_CALIBRE_DB_READ_OPTIONS when both are specified.
MGC_FDI_OA_VERSION	{22.43 22.60 22.50}	Sets the OpenAccess version for FDI and DBdiff utilities. OA 22.43 is the default. If you specify the OA version number in the FDI or DBdiff utilities, the environment variable value is ignored.
OA_HOME ¹	<i>pathname</i>	Calibre uses the OpenAccess library at <calibre_install_dir>/shared/pkgs/icv_oa. If additional plug-ins are needed, use the OA_HOME environment variable to specify the path to an OA library that contains the plug-ins. Note, that the library specified by the OA_HOME variable is not used. Calibre always uses the default OpenAccess library included with the Calibre installation. This variable applies only to dbdiff, fdi2gds, and fdi2oasis.

1. If this variable is set, then the specified OA library must be oa22.43p006 or later. libstdc++.so.6.0.8 must be available from the host (RHEL 4u6 and later have it by default) or through the LD_LIBRARY_PATH environment variable.

For information about translating LEF/DEF or OpenAccess layout formats to GDS or OASIS, see the [Calibre Layout Comparison and Translation Guide](#). For information on setting database options in Calibre Interactive, see “Third Party Database Options” in the [Calibre Interactive User’s Manual](#).

OpenAccess Support — OpenAccess is widely used in the EDA industry. As new versions are released by Si2, Siemens Digital Industry Software will qualify and adopt them with approximately a one-calendar-quarter lag from the Si2 release to the public. Because Cadence

provides the OA reference implementation, some Cadence tools may be released in parallel with Si2's release, thus placing the Calibre OpenAccess library up to one-quarter out of sync with the most recent Cadence tools.

Be sure all third-party pcell evaluators are compatible with the latest supported version of OA in Calibre. The suggested Virtuoso version is IC615.

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 number. 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. This is the form:

```
<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 a rectangle that is oriented orthogonally with respect to the database axes.

The ASCII database format does not support text labels for connectivity extraction. For such databases, connectivity extraction text can be specified using the [Text](#) specification statement in the rule file.

Chapter 4

Calibre nmDRC Concepts

This chapter describes fundamental concepts germane to Calibre nmDRC operation, rule file structures, and usage.

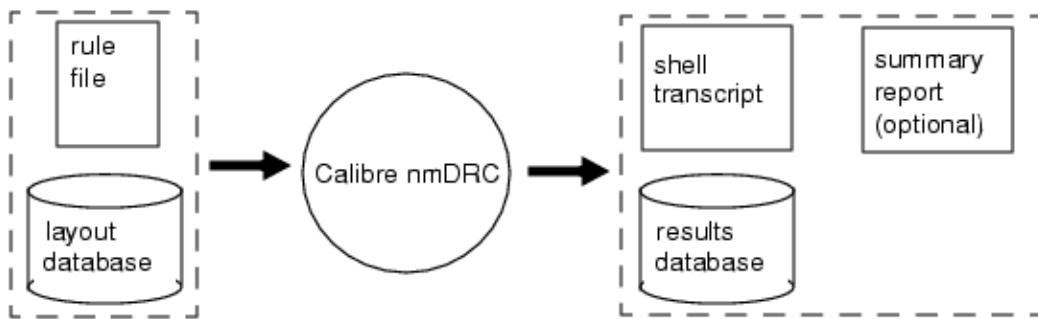
Data Flow in Calibre nmDRC	109
Invocation	110
Calibre nmDRC Specification Statements	111
Rule Check Statements	112
Rule Check Selection	116
Rule Check Result Limits	117
Control of Empty Rule Checks	118
Concurrency	118
Result Redundancy Elimination	119
Layer Operation Scheduling	119
Maximizing DRC Capacity and Minimizing Run Time	120
Polygon Segmentation	124
Rule File Compilation	126
Calibre nmDRC Exit Values	126
Advanced Calibre nmDRC Concepts	128

Data Flow in Calibre nmDRC

The inputs to Calibre nmDRC are a rule file and a layout database. The outputs are a run transcript and a DRC Results Database, with an optional DRC Summary Report. Certain layer operations generate their own specialized results databases, and these are also optional.

Figure 4-1 shows the Calibre nmDRC data flow.

Figure 4-1. Calibre nmDRC Data Flow



The rule file is either SVRF or TVF format. The input layout database can be GDSII, OASIS, LEF/DEF, OpenAccess, or ASCII. The latter is a proprietary format and only works in flat Calibre nmDRC. Certain third-party layout formats are supported through the Foreign Database Interface (FDI). The “[Setup and Tool Invocation](#)” chapter discusses the required inputs in detail.

The results database can be ASCII (default), GDSII, or OASIS. For general design rule checks, you should output an ASCII DRC results database. When you use Calibre nmDRC-H for mask layer generation, you should output a GDSII or OASIS DRC results database (see DRC Check Map in the SVRF Manual); you should also specify DRC Maximum Results ALL in this case. You should follow these guidelines because Calibre nmDRC-H requires extra internal overhead to generate mask layer results. The “[Calibre nmDRC Results](#)” chapter discusses DRC results databases.

Related Topics

[DRC Check Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Results Database \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Summary Report \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Invocation

Calibre nmDRC and Calibre nmDRC-H have some typical command line invocations.

The following command line invokes Calibre nmDRC:

```
# flat run  
calibre -drc rule_file
```

The following command lines invoke Calibre nmDRC-H:

```
# hierarchical run
calibre -drc -hier rule_file

# multithreaded hierarchical runs
calibre -drc -hier -turbo -hyper rule_file
calibre -drc -hier -turbo -remote host1,host2,host3 -hyper rule_file
```

Compilation errors are detected one-at-a-time, so if there are multiple errors in your rule file, you will find them iteratively by fixing each error after it appears and running the tool again.

Related Topics

[Calibre nmDRC and Calibre nmDRC-H Command Line](#)

[Setup and Tool Invocation](#)

[Rule File Compilation](#)

Calibre nmDRC Specification Statements

Specification statements control the overall behavior of a Calibre run. These statements control things like inputs, outputs, which rule checks are run, which regions of the layout are checked, and so forth.

Some statements are required rule file statements for DRC applications. Refer to “[Required Statements](#)” on page 32 for the minimum set of specification statements.

The section “[DRC Specification Statements](#)” in the *SVRF Manual* shows a summary of all specification statements for DRC applications.

The section “[Layout Database Specification Statements](#)” in the *SVRF Manual* summarizes the statements that control the handling of layout databases. Many frequently-used specification statements are discussed in this section.

Specification statements may not be used inside [Rule Check Statements](#) or within MACROs.

Many specification statements are supported from within Calibre Interactive—DRC. Options that you specify in the GUI appear as specification statements in the control file generated by Calibre Interactive. See “[Using Calibre Interactive to Perform DRC](#)” in the *Calibre Interactive User’s Manual*.

Rule Check Statements

Rule check statements are used in many Calibre applications (as well as other Siemens EDA tools that use rule files, like ICVerify). They are used in design rule checks (DRC), optical and process checks (OPC), mask data preparation (MDP), and electrical rule check (ERC) operations performed in LVS. Rule checks specify layer operations that send the resulting derived layers into the 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 or it will not compile (that is, it must have at least one free-standing layer operation that outputs results data).

Rule check statements take the following form:

```
name {  
    @ rule check comment  
    layer_definition | layer_operation  
    ...  
    layer_definition | layer_operation  
    ...  
}
```

where “name” is the name of the rule check, and each line consists of a comment, layer definition (see [Layer Definitions](#)), or a free-standing layer operation.

Rule check names must be unique. 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 DRC application executes a rule check statement, it places the output layers in the DRC results database (see [Layer Types and Data Flow in the DRC System](#)). All derived layers created by all free-standing layer operations (not layer definitions) within the rule check statement are placed in the DRC results database. Refer to “[Calibre nmDRC Results](#)” on page 205 for information on the DRC results database.

For example, the following rule check statement generates layer definitions and derived error layers:

```
METAL_WIDTH {  
    // Metal width check. Metal width must be greater than or equal to  
    // 3 microns except where metal length exceeds 5 microns; in that case,  
    // metal width must be greater than or equal to 4 microns.  
    long_metal = metal LENGTH > 5           // Layer definition;  
                                         // no output to results db.  
    INTERNAL long_metal < 4                 // Output to results db.  
    short_metal = metal NOT LENGTH > 5      //Layer definition.  
    INTERNAL short_metal < 3                //Output to results db.  
}
```

The layer operation “metal LENGTH > 5” defines layer long_metal; this layer is then used within the next operation. There is no output to the DRC results database corresponding to the metal LENGTH > 5 operation because it is part of a layer definition (it has an = sign). Similarly, the third operation is a layer definition. The second and fourth operations are not layer definitions; these operations generate results *data*, which is output to the DRC results database under the name METAL_WIDTH.

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. The Copy operation is a useful debugging tool when used in this way. Be sure to comment out any such debugging statements when you finish with them.

Rule Comments	113
Check Text	114

Rule Comments

You can use three types of comments within a rule file. 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 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:

Example 4-1. Rule Comments

```
METAL_SPACING {  
    @ Metal to metal spacing errors.  
    @ Do not confuse with notch errors;  
    @ the spacing rule is 4 microns.  
    metal EXTERNAL < 4 space // Check spacing between different polygons.  
}  
  
VARIABLE notch_delta 3.5  
  
METAL_NOTCH {  
    @ Metal notch errors.  
    @ Do not confuse with spacing errors;  
    @ the notch rule is ^notch_delta microns.  
    metal EXTERNAL < notch_delta 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. Calibre RVE also recognizes certain rule check comments as configuration commands when it loads a DRC results database.

The ^ symbol used in an @ rule check comment causes a [Variable](#) value (only the first value for an array) to appear in DRC result reports. Otherwise, the variable's name appears.

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 this comment style can lead to unexpected results, especially when it is not clear where a comment block begins or ends.

Related Topics

[DRC Rule Check Comments for Calibre RVE \[Calibre RVE User's Manual\]](#)

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 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. Check text comments can be displayed in Calibre RVE for DRC.

The DRC Check Text specification statement controls mapping of check text to the DRC results database. The default is to map the rule file pathname, title, and rule check comments.

Related Topics

[DRC Check Text \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Rule Check Selection

You can select one or more rule check statements from a rule file, which DRC runs as a unit. This unit is called the *check set*. A rule check name is specified immediately before an open brace in a rule check.

For example:

```
rule_check_name { ...
```

By default, Calibre nmDRC 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 nmDRC 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 nmDRC selects all rule checks. Otherwise, Calibre nmDRC selects only those rule checks specified in DRC Select Check specification statements.
2. Calibre nmDRC does not select any rule checks specified with DRC Unselect Check specification statements in the rule file.

Both the selected and unselected Calibre nmDRC rule checks are shown in the transcript after rule file compilation in a Calibre run.

The [DRC \[Un\]Select Check By Layer](#) specification statements provide additional control over which rule checks are executed, based upon layers you specify.

Rule Check Selection and Mask Results Databases [116](#)

Rule Check Selection and Mask Results Databases

Calibre nmDRC and related applications only read in layers from the layout database as needed for the check set that is active for the run. If a layer is not required for the run, it is not read in. Similarly, layers are only derived as necessary to satisfy the data requirements of the check set.

When producing a mask (geometric) results database, the data that is output is affected by the check set. The structure of the output database, including hierarchy, cell extents, and output layers is affected by the check set that is used for the run.

Mask results database differences between runs due to the check set can be tracked by running the complete DRC rule file using statements like Layout Window, Layout Window Layer, and Layout Window Cell for areas of the design that are of particular interest. The results windows from the complete DRC run can be used for comparison when you run with a smaller check set on the complete design.

Differences in results database hierarchy due to the check set can be mitigated somewhat by using the DRC Results Database options USER and USER MERGED. The PSEUDO option tends to show the greatest differences in output hierarchy due to differing check sets.

Related Topics

[DRC Results Database \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Layer Operation Scheduling](#)

[Layout Window \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Layout Window Layer \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Layout Window Cell \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Maximizing DRC Capacity and Minimizing Run Time](#)

Rule Check Result Limits

You can limit the number of DRC results written to the DRC results database for any given rule check by using the DRC Maximum Results specification statement, or by using the MAXIMUM RESULTS parameter of 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. If you are producing mask data results, then you should specify the keyword ALL. Calibre issues a warning whenever the results limit is reached.

As a performance optimization in flat Calibre nmDRC, this maximum result limit is internally passed to the [External](#), [Internal](#), and [Enclosure](#) operations. These operations stop the generation of DRC results when the results to be written *only* to the DRC results database will exceed the result limit. This mechanism is not used where more than one operation is used in a rule check. 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 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 ENClosure, EXternal, and INTernal 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.

Related Topics

[DRC Maximum Results \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Check Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

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 execution of the rule check that is important, whether or not there actually are results.

In Calibre nmDRC, 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. The DRC Keep Empty statement has no effect upon DRC Check Map databases.

Related Topics

[DRC Keep Empty \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Check Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Concurrency

Calibre nmDRC performs many of the layer operations concurrently. This means that they run simultaneously when present in a rule file. Whenever Calibre nmDRC performs layer operations, it locates all required layer operations within the set that it can run concurrently and executes them as a single group. This offers significant performance advantages.

Operations that have been executed concurrently are shown in the logfile as grouped together. Here is an example:

```
min_spacing_metal1::<1> = EXT metal1 < 1
min_metal1_width::<1> = INT metal1 < 2
-----
min_spacing_metal1::<1> (HIER TYP=3 HGC=0 FGC=0 HEC=0 FEC=0)
min_metal1_width::<1> (HIER TYP=3 HGC=0 FGC=0 HEC=0 FEC=0)
CPU TIME = 0 REAL TIME = 0 LVHEAP = 0/3/3 OPS COMPLETE = 5 OF 20
ELAPSED TIME = 7
```

The two operations in this example have been executed concurrently.

Operations that are not performed concurrently are shown this way:

```
min_contact_width::<1> = INT contact < 1.75
-----
min_contact_width::<1> (HIER TYP=3 HGC=0 FGC=0 HEC=0 FEC=0)
CPU TIME = 0 REAL TIME = 0 LVHEAP = 0/3/3 OPS COMPLETE = 10 OF 20
ELAPSED TIME = 7
```

Result Redundancy Elimination

Calibre nmDRC 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 nmDRC 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 DRC applications, you do not need to use the global scope to avoid duplicating layer operations.

Layer Operation Scheduling

Calibre nmDRC 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 nmDRC 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 nmDRC must evaluate to produce A. The process stops when Calibre nmDRC retrieves data for the original layers from the database.

Calibre nmDRC schedules all involved layer operations for concurrency and to undergo redundancy elimination, as described under “[Result Redundancy Elimination](#).” If a layer operation is required to evaluate a check set, Calibre nmDRC does not perform that operation more than once during execution of the check set. Calibre nmDRC performs only those operations necessary to satisfy the data requirements of the check set.

In non-multithreaded mode, Calibre nmDRC 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 DRC Capacity and Minimizing Run Time

Calibre nmDRC 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 nmDRC attempts to maximize capacity by delaying data production until it is required and by deleting data when it is no longer required.

Calibre nmDRC 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 nmDRC 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 nmDRC generated the layer earlier according to [Concurrency](#).
- All layers, original or derived, persist only until they are no longer required, at which point Calibre nmDRC 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	120
Efficient Width and Spacing Checks	121

Conjunctive Checks

Calibre nmDRC performance depends on the amount of data Calibre nmDRC must use at any given time. In designing certain conjunctive checks, you can reduce data generation overhead and run time.

Consider the following design rule:

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

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

```
metal_contact_check {
    x = poly OR diff
    y = x AND metal
    contact NOT inside y
}
```

The problem with this approach is that, in a large design, Calibre nmDRC 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. 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:

```
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 data overhead for 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 performs faster.

In general, topological operations perform faster than measurement operations (where a dimension measurement constraint is used). If you can use a topological operation rather than a measurement operation, this often improves performance.

Efficient Width and Spacing Checks

In general, it is best to avoid using a dimensional check operation when other operations would suffice. Dimensional check operations are slower than other layer operations because of the computational overhead of checking distances between numerous edges.

One example where this applies is contact checking. For example, consider the design rule:

Contacts must be rectangles of width greater than or equal 0.10 and less than or equal to 0.15.

One way to check this is the following:

```
contact_size {  
    INT(contact) >= 0.10 <= 0.15 ABUT < 90  
    // Output the bad edges
```

This is functionally correct, but relatively slow because of the number of contact edges that must be measured.

A faster rule is this one:

```
contact_size {  
    NOT RECTANGLE contact >= 0.10 <= 0.15 BY >= 0.10 <= 0.15  
}
```

Tip The [Not] Rectangle operation is optimized to look for [non-]rectangular polygons and to filter them according to edge length criteria.

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:

```
Rule_5 {  
    EXT pad metal2 < 32 // this is a huge constraint distance  
}
```

The constraint used in this check is huge in comparison to typical feature sizes, and the number of metal2 polygons in the design is probably large. Calibre nmDRC does not 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. Therefore, this check is very slow.

Tip To avoid the performance penalty of using large constraints, *reduce the layer data* that the dimensional check operation must consider whenever this is possible. You can do this by filtering out non-essential objects using well-constructed layer derivations.

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:

Example 4-2. Optimized Spacing Check for Large Measurement Distance

```
Rule_5 {
    X = SIZE pad BY 32.1           // make pads larger by 32.1 units
    Y = metal2 NOT OUTSIDE X
    // any metal 2 touching or within X? probably not much.
    EXT pad Y < 32                //now do the external check
}
```

For this check, note the following things:

- The choice of 32.1 for the **Size BY** operation is somewhat arbitrary. You want to grow the pads just a bit more than the constraint distance you want to check in order to ensure all metal2 polygons that could cause errors are considered.
- All metal2 polygons inside or touching layer X are output as layer Y. There are likely few of these polygons, so the end result of deriving layers X and Y is to filter out all the metal2 you are not concerned about.

For the remainder of this discussion, assume you have the following layer derivation:

```
x = EXT layer1 > 30.0 < 90.0 OPPOSITE REGION
```

The intent of this sort of a derivation is to store the regions between layer1 polygons that are spaced > 30.0 and lie within a cell boundary. These regions are then used for further checks.

Assuming layer1 is rather dense, this derivation is inefficient because the lower bound of this constraint is huge, and the constraint interval is likewise huge.

Tip

 When you want to take EXternal-type measurements and the measurement interval is large, you can use either **Expand Edge** or **Size** to write a more efficient rule. Also consider using **DFM Space**.

For the derivation of layer x above, layer1 could either be a derived edge layer or a polygon layer. The following derivations cover these two cases and are far more efficient from a runtime standpoint than using EXternal.

Case 1 — Input layer is derived-edge type.

Example 4-3. Using Expand Edge Instead of EXTernal

```
w = EXPAND EDGE layer1 OUTSIDE BY 15.0
// Merge layer1 shapes that are <= 30.0 user units apart.
// If they are spaced > 30, there will be more than one shape on w.

y = EXTENT DRAWN ORIGINAL // get the extent of the chip

z = y INTERACT w > 1
// If there is more than one shape on w, then z = chip extent.
// Otherwise, z is empty.

layer1_big = w INTERACT z
// Oversized layer1. Use this for further derivations as needed.
```

This derivation avoids using a dimensional check operation completely. It also uses techniques like reducing the layer data that layer operations must consider. The Expand Edge operation merges numerous polygons together, thus making the [Interact](#) operation very fast.

Tip

 Interact is a versatile layer operation and quite fast. It is useful for filtering polygons that overlap or touch each other.

Case 2 — Input layer is polygon type.

The only difference between Case 2 and Case 1 is Size is used instead of Expand Edge.

Example 4-4. Using Size Instead of EXTernal

```
w = SIZE layer1 BY 15.0
y = EXTENT DRAWN ORIGINAL
z = y INTERACT w > 1
layer1_big = w INTERACT z
```

One drawback to using this type of derivation is the results presentation is not as refined as for EXTERNAL because the output more closely represents the use of the SQUARE metric as opposed to the default Euclidean metric. However, if you look at the results from these two cases in the context of the original layout shapes, you should be able to determine which edges satisfy the constraints of the original rule.

To get a better approximation of the Euclidean measurement, you can remove small convex corners from layer1 polygons, thus leaving small 45-degree edges at the corners instead.

Polygon Segmentation

By default, Calibre nmDRC segments any DRC-generated polygon whose vertex count exceeds 4096. You can control the maximum vertex count for polygon segmentation with the DRC Maximum Vertex specification statement.

Note that if the value specified in DRC Maximum Vertex exceeds 4096, Calibre nmDRC uses the specified value even though some DRC results databases (ASCII, for example) may contain no polygons having more than 4096 vertices. Therefore, for some output database formats, you should limit the vertex count of polygon results. For example:

- Calibre has a limit of 8191 vertices for reading in GDSII polygons. For OASIS there is no practical limit, but try to keep the number of input vertices per polygon below 1E06 for optimum performance.
- The Pyxis Layout template database has a vertex limit of 4096.
- The Calma GDSII database format specification has a vertex limit of 200, which some tools may enforce, but Calibre does not.

DRC Maximum Vertex ALL denotes a limit of 2^{32} vertices for OASIS output and 8190 for GDSII output. However, 4096 should be used as the limit for ASCII results databases. For GDSII, Calibre can read up to 8191 vertices on input. For OASIS, the read limit is 2^{32} vertices.

Calibre nmDRC uses a polygon segmentation algorithm that breaks up result polygons whose vertex count exceeds the DRC Maximum Vertex value. This produces polygons with sufficiently small vertex counts to meet the DRC Maximum Vertex limit. These are called polygon segments. *The merged representation of the segments is equal to the original polygon.*

Calibre nmDRC computes DRC result counts after applying any polygon segmentation. These result counts appear in the DRC results database or in various DRC transcript messages.

Whenever a polygon result is segmented, a warning is issued—once per polygon—as segmentation can produce potentially confusing results for certain operations. When a large-vertex-count polygon is segmented, the segments can have edges that are off grid (the polygon segmentation algorithm does not obey the user grid). This is can be potentially confusing in mask results DRC output if you expect the segmented edges to be on grid. This is not a problem, however, because all the polygon segments are on the same layer, and all the segments get merged on any future input to Calibre. Any segmented edges that are off grid get merged away on input, so the original large-vertex-count polygon is on grid. Most DRC tools perform such merging.

Another potentially confusing situation in mask output is, polygon segments can have unexpected angles in them. Large-vertex-count polygons are segmented without regard to angle measure criteria because angles in polygon segments that might be considered DRC errors (when viewed out of context from the adjacent polygon segments) get merged on any future input, so the entire large-vertex-count polygon has no unusual angles after merging of its segments.

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.

Related Topics

[DRC Maximum Vertex \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Rule File Compilation

You must specify a rule file name to invoke Calibre nmDRC applications. Calibre nmDRC automatically compiles the rule file as the first step of executing the command. Compilation errors stop the program and error codes are issued to the run transcript along with descriptions of the error.

Compilation errors are presented in the “[Error Messages](#)” chapter of the *SVRF Manual*.

Here is a simple script to check rule file syntax:

```
#! /bin/csh -f
calibre -svrf $argv[1] << EOF
FILE outfile
FREEZE
QUIT
EOF
```

Make the script executable and then call the script with your rule file name as an argument.

If compilation succeeds, the transcript shows this:

```
--- STANDARD VERIFICATION RULE FILE COMPILATION MODULE COMPLETED.
```

The *outfile* contains the currently active rule file code. Any conditional compilation instructions in the rules are taken into account.

If compilation fails, an error message is returned that corresponds to the first problem encountered:

```
ERROR: Error INP1 on line 6 of rules - superfluous or invalid input
object: RC.
```

Calibre nmDRC Exit Values

Calibre nmDRC applications exit with a non-zero status if they cannot complete any form of requested processing due to fatal error conditions. In a Linux shell, you can check this status by examining the \$? shell variable.

For example, here is a script that returns messages based on exit status:

```
#!/bin/sh
calibre -drc -hier rules
if [ $? -eq 0 ]
then
    echo "No errors."
else
    echo "Error in DRC!"
fi
```

Advanced Calibre nmDRC Concepts

Historically, physical verification has relied on independent, single-dimensional rule checks to identify layout features most sensitive to failure during manufacturing. As technology nodes shrink and manufacturing variability increases, checks must be written to address a new variety of complex design features.

Equation-based DRC (or eqDRC) is a set of Calibre SVRF operations that enable you to analyze complex interactions that are difficult or impossible to verify using traditional DRC methods.

These areas can benefit from eqDRC capabilities:

- **Area and yield** — Increase the accuracy of DRC checks to avoid over-constraining or under-constraining a design.
- **Productivity** — Simplify debugging with the ability to view the modification values needed to adjust the layout in the immediate context of affected design geometries.
- **Check complexity** — Reduce the number of rules in DRC checks by using continuous functions instead of piece-wise linear interval values (sometimes referred to as “buckets”).
- **Check capability** — Verify complex design features that are extremely difficult or impossible to check using traditional DRC.
- **Restrictive Design Rules (RDR)** — These rules trade design flexibility for enhanced printability. As technology nodes shrink below 32 nm, lithography costs rise dramatically. Restrictive design rules help to mitigate cost by enforcing geometric configurations that are known to print successfully. Examples of restrictive design rules include pitch checks, grid checks, and pattern matching.

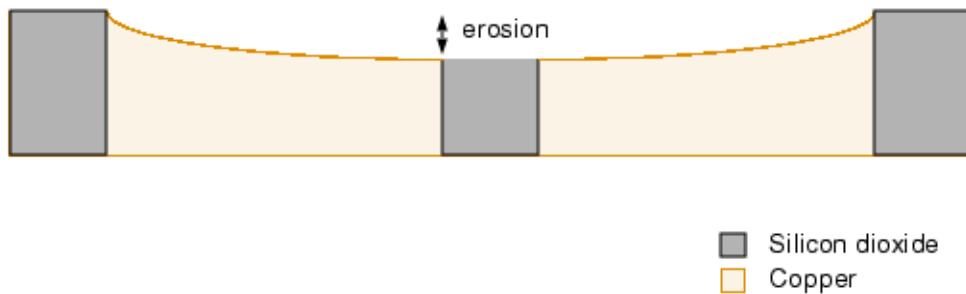
The following sections provide examples and additional information on advanced DRC concepts:

Wide Metal Spacing Rules	128
Advanced DRC Toolset	135

Wide Metal Spacing Rules

To demonstrate the advantages of eqDRC, consider the example of wide metal spacing. Wide metal spacing refers to a design challenge where wide metal bands separated by small areas of silicon dioxide can lead to performance and yield problems. Since copper is softer than oxide, it is removed more easily during the Chemical Mechanical Polishing (CMP) process, which can lead to metal erosion.

[Figure 4-2](#) illustrates the effects of erosion from the CMP process.

Figure 4-2. Erosion Effects

If the copper bands are too wide relative to the area of oxide between them, there is increased susceptibility to erosion because the oxide does not create enough resistance against the polisher, and the two copper bands essentially behave as a single large area of copper. Increased erosion can lead to undesirable changes in resistance and capacitance. It can even cause open circuits. Designers must take care to avoid erosion by ensuring a minimal separation distance for a number of metal widths.

Traditional DRC Implementation

The common method of addressing the erosion problem involves a series of single-dimensional design rules. A minimum space value is associated with each range of widths for a single band and both bands.

For example:

```
Rule1: 0.09 minimum M1 Space if both bands >= 0.09 wide
Rule2: 0.25 minimum M1 Space if one band    >= 0.36 wide
Rule3: 0.30 minimum M1 Space if both bands >= 0.36 wide
Rule4: 0.40 minimum M1 Space if one band    >= 1.44 wide
Rule5: 0.50 minimum M1 Space if both bands >= 1.44 wide
Rule6: 0.55 minimum M1 Space if one band    >= 4.50 wide
Rule7: 0.60 minimum M1 Space if both bands >= 4.50 wide
```

However, the following examples show that the effectiveness of this approach can limit designers in reducing erosion effects.

Figure 4-3 shows a case where both wires are 1.44 units wide, and the separation distance is 0.49 units. This design fails rule 5.

Figure 4-3. Case 1

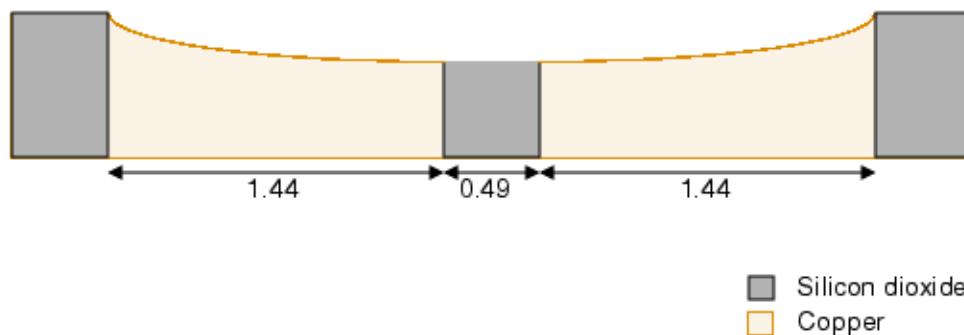
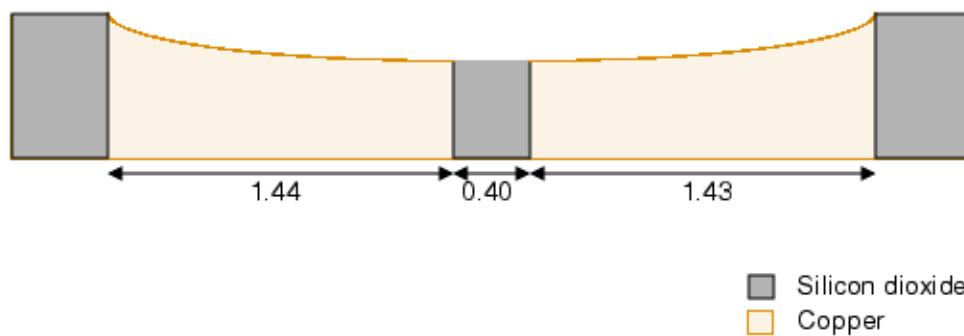


Figure 4-4 shows a case where the bands are 1.44 and 1.43 units wide, and the separation distance is 0.4 units. Rule 4 now applies instead of rule 5, and the design passes.

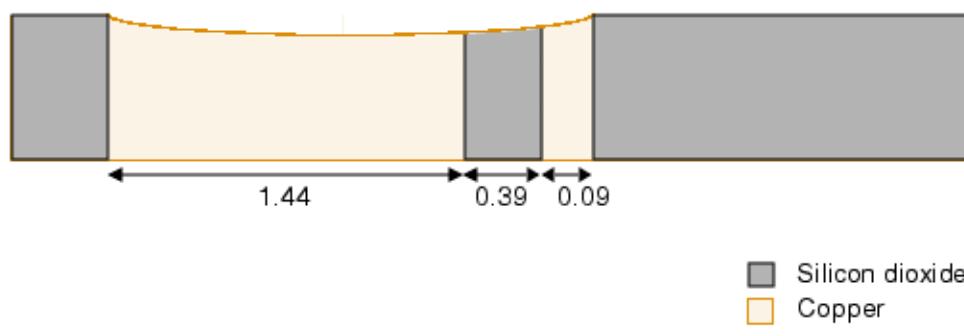
Figure 4-4. Case 2



However, from a standpoint of erosion susceptibility, the failing case shown in Figure 4-3 is actually better than the passing case shown in Figure 4-4. In the passing case, since the oxide area is smaller, more erosion will occur.

In Figure 4-5, one wire has a width of 1.44 units, but the other wire has been dramatically reduced to a width of 0.09 units. Since one wire is 1.44 units, the design fails rule 4 and the designer must increase the separation distance. Again, from a standpoint of susceptibility to erosion, Figure 4-5 is much better than Figure 4-4.

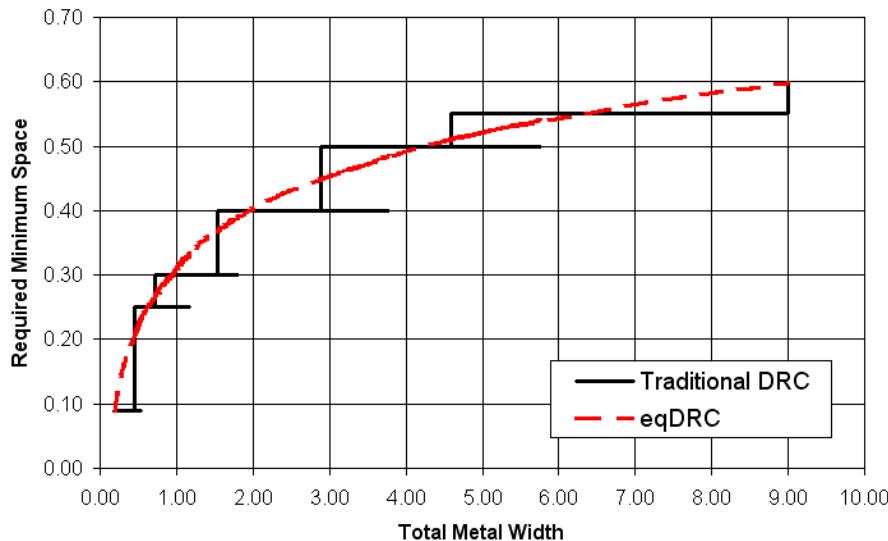
Figure 4-5. Case 3



eqDRC Implementation

Although the resolution of the width ranges can be increased by adding more rules, a better approach is to create a continuous function of the width to space values. To do this, find a curve of best fit to the discrete values defined by the traditional DRC rules as shown in [Figure 4-6](#). (In this case, the stepwise linear graph of the traditional DRC measurement intervals is roughly logarithmic. So an application that can perform logarithmic curve fitting is useful.)

Figure 4-6. Traditional DRC Versus eqDRC



Note the artifacts that extend to the right of the stair step pattern for each minimum space value on the traditional DRC plot. Since there is no upper constraint on width for one band or both bands (only that the widths must be greater than or equal to a minimum value), for each space value, the total metal width can continue to increase without violating any of the traditional DRC rules.

[Figure 4-6](#) shows how the traditional DRC rules can over-constrain designers (sacrificing area), or under-constrain designers (sacrificing yield). The interval values that extend above the eqDRC curve represent over-constraining the geometry, because these intervals imply exclusions of geometric dimensions that should be valid. The intervals that extend below the eqDRC curve represent under-constraining the geometry, because these intervals permit geometric dimensions that should be forbidden.

An equation can be obtained from the red curve in terms of the two variables—the separation distance (*space*) and the total width of the bands (*width1* plus *width2*). The following equation fits the curve:

$$space = \frac{0.09}{\ln(2)} \cdot \ln\left(\frac{totalwidth}{0.09}\right)$$

You can then create a DRC-style check by dividing both sides of the equation by *space*, then setting 1 as the cutoff value for DRC—if the result is less than or equal to 1 the condition passes; if the result is greater than 1 the condition fails.

$$\frac{0.09}{\text{space} \cdot \ln(2)} \cdot \ln\left(\frac{\text{totalwidth}}{0.09}\right) > 1$$

For failing conditions, it is useful to display the changes in separation distance or width needed to pass the condition. To do this, first derive additional equations by solving for each variable. Since the isolated variables represent the values needed to pass the condition, you can represent them as *value* + *delta(value)*, where *delta(value)* is the amount by which the original value needs to be changed to pass the condition.

$$\text{space} + \Delta\text{space} = \frac{0.09}{\ln(2)} \cdot \ln\left(\frac{\text{totalwidth}}{0.09}\right)$$

$$\Delta\text{space} = \frac{0.09}{\ln(2)} \cdot \ln\left(\frac{\text{totalwidth}}{0.09}\right) - \text{space}$$

$$\text{totalwidth} + \Delta\text{totalwidth} = 0.09 \cdot e^{\left(\frac{\text{space} \cdot \ln(2)}{0.09}\right)}$$

$$\Delta\text{totalwidth} = \left(0.09 \cdot e^{\left(\frac{\text{space} \cdot \ln(2)}{0.09}\right)}\right) - \text{totalwidth}$$

These equations can then be used as DFM expressions to define DFM properties, which can be saved to a DFM RDB or DFM database and displayed for each geometry in a layout viewer.

The following steps show how you can accomplish this in a rule file:

1. Create a derived error layer where the edge clusters mark the separation distance between the bands. This layer is used as the primary input layer to the DFM Property operation in Step 3.

```
mtlspc = EXT MET1 <= 0.60 OPPOSITE
```

2. Create a derived error layer where the edge clusters mark the width of each band.

```
mtlwid = INT MET1 <= 5.00 OPPOSITE
```

3. Create a derived error layer eqdrc_err with three properties attached: error_value, change_space_by, and change_sum_width_by. The eqdrc_err layer is the subset of the primary layer mtlspc for which the property constraints are met. The EW measurement function returns the separation distance of the input error layer. Because mtlwid is a secondary layer, the EW values are automatically summed over secondary geometries in the cluster, so that EW(mtlwid) returns the total width of the two metal bands that abut the mtlspc error cluster

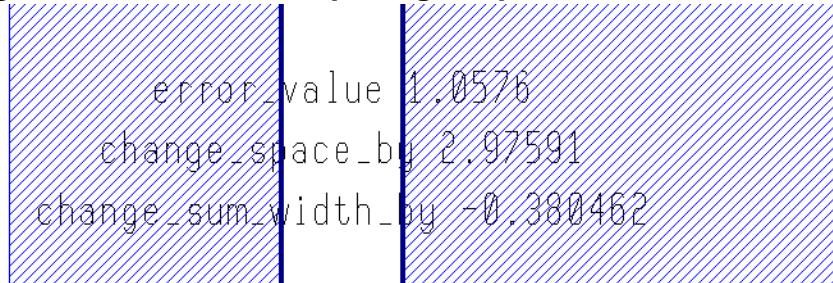
```
eqdrc_err = DFM PROPERTY mtlspc mtlwid OVERLAP ABUT ALSO MULTI
[error_value = 0.09/(EW(mtlspc)*LOG(2)) * log(EW(mtlwid)/0.09)] > 1
[change_space_by = (0.09/LOG(2))*(EW(mtlwid)/0.09)-EW(mtlspc)]
[change_sum_width_by = 0.09*EXP((EW(mtlspc)*LOG(2))/0.09)
-EW(mtlwid)]
```

4. Output the eqdrc_err layer (with properties) to a DFM RDB file named *eqDRC.rdb*.

```
eqDRC_rule {
    DFM RDB eqdrc_err eqDRC.rdb
    COMMENT "Wide metal spacing Rule: Use the property values to
            determine the error magnitude and adjustments needed to
            fix the violation"
}
```

[Figure 4-7](#) shows the output in Calibre DESIGNrev for a pair of failing metal bands.

Figure 4-7. Wide Metal Spacing Output in Calibre DESIGNrev



The property values shown in this figure indicate that in order to pass the check, you can either increase the spacing by approximately 2.98 units, or decrease the total width by approximately 0.38 units.

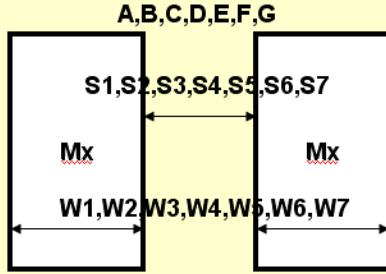
Note on Process Design Rule Manuals

Once a suitable equation is derived from the set of data, the equation can be integrated into a process design rule manual.

[Figure 4-8](#) and [Figure 4-9](#) show a comparison between an example DRC and eqDRC page.

Figure 4-8. DRC Process Design Rule Manual

Process Design Rule Manual				
Rule Name	Description	label	Rule	
MxSpc1	Space (S1) if both metal wire widths (W1) ≥ 0.09	A	\geq	0.09
MxSpc2	Space (S2) if one metal wire widths (W2) ≥ 0.36	B	\geq	0.25
MxSpc3	Space (S3) if both metal wire widths (W3) ≥ 0.36	C	\geq	0.30
MxSpc4	Space (S4) if one metal wire widths (W4) ≥ 1.44	D	\geq	0.40
MxSpc5	Space (S5) if both metal wire widths (W5) ≥ 1.44	E	\geq	0.50
MxSpc6	Space (S6) if one metal wire widths (W6) ≥ 4.50	F	\geq	0.55
MxSpc7	Space (S7) if both metal wire widths (W7) ≥ 4.50	G	\geq	0.60



A,B,C,D,E,F,G

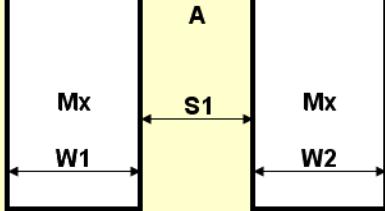
S1,S2,S3,S4,S5,S6,S7

Mx

W1,W2,W3,W4,W5,W6,W7

Figure 4-9. eqDRC Process Design Rule Manual

Process Design Rule Manual				
Rule Name	Description	label	Rule	
MxSpc	Min space (S1) is dependent on the sum of width W1 + W2	A	$>$	$\frac{0.09}{\ln(2)} * \ln\left(\frac{W1+W2}{0.09}\right)$



Examples	S1	W1	W2
Example 1	0.09	0.09	0.09
Example 2	0.21	0.09	0.36
Example 3	0.39	0.36	1.44
Example 4	0.45	1.44	1.44
Example 5	0.54	1.44	4.50

By incorporating equations into design rule manuals, foundries can express actual process variation metrics as functions of multidimensional design variations to prioritize and flag potential issues in ways that were previously not possible. Furthermore, these equations eliminate the need for threshold tables entirely, resulting in potentially more accurate checks that are easier to specify and debug. As new process or design requirements are identified, new or modified equations can be made to maintain yield and performance goals.

In cases where modifying the design rule manual is not possible, the traditional “bucketed” rules can still be combined with eqDRC to take advantage of reduced code complexity and the ability to show correction values in a layout viewer. Refer to the *Calibre Solutions for Physical Verification* manual for more information.

Advanced DRC Toolset

You can implement rule checks that take advantage of eqDRC using a set of SVRF statements.

The following table identifies the eqDRC SVRF commands. Most operations can use a Calibre nmDRC-H license in a calibre -drc run.

Table 4-1. eqDRC SVRF Statements

Statement	Description
DFM Classify	Groups geometries on a layer by one or more specified DFM properties and outputs a representative geometry from each group. Each output geometry is annotated with a numeric property named “Class” that contains the group number.
DFM Copy	Copies input layers of any type to a new layer. The type of input layer can be changed in the output. DFM properties are included in the output for single layer operation.
DFM Defaults	Specifies default options for certain operations.
DFM Function	Creates a user-defined function for use in DFM Property expressions.
DFM OR	Performs an OR operation on the input polygon layers and includes DFM properties from the input layers on the output layer.
DFM Property	Creates a derived layer with the specified DFM Properties attached. Objects on the output layer must meet specified criteria.
DFM Property Merge	Accepts unmerged layers as input and converts them into regular merged layers.
DFM Property Net	Creates an empty polygon layer containing DFM properties calculated from one or more input layers. Properties are reported per net in the output layer.
DFM Property Select Secondary	Similar to DFM Property, but outputs geometries from a secondary layer or from an edge or extent layer derivation on one of the input layers.
DFM Property Singleton	Creates a polygon layer with no physical geometries and containing user-defined DFM properties calculated from one or more input layers.
DFM RDB	Writes original or derived layer data, including properties, to the specified output format.

Table 4-1. eqDRC SVRF Statements (cont.)

Statement	Description
DFM Read	Reads layout properties in the input database and converts them to DFM properties. The output layer is an unmerged polygon layer with attached DFM properties.
DFM Space	Provides efficient measurements of edge-to-edge distances for large distance constraints.
DFM Stamp	Assigns node numbers to a layer based on DFM properties.
DFM Text	Converts text objects to DFM properties and outputs an unmerged layer.

Short introductions to these commands are in these sections:

Introduction to DFM Property	136
Introduction to DFM Function	139
Introduction to DFM RDB	140
Introduction to DFM Copy and DFM OR	141

Introduction to DFM Property

DFM Property is the keystone of eqDRC. It is a layer operation that accepts one or more layers. In most cases the input layers can be a combination of any layer type: original, derived polygon, derived edge, and derived error. It outputs a layer of the same type as the primary (first) input layer with any user-specified properties (name/value pairs) attached.

Layout objects on the output layer are a subset of the layout objects on the primary input layer. Objects on the primary layer may be excluded from the output layer based on constraints specified in the operation. When property calculations involve multiple input layers, the operation clusters objects on the primary layer with objects on the secondary layers based on specified keywords.

DFM Property provides these capabilities:

- **Measurement Functions** — DFM Property supports several measurement functions that calculate geometric attributes, such as length and area. The measurement results are attached to layer shapes as DFM properties. See “[Measurement Functions](#)” and “[DFM Properties](#)” in the *SVRF Manual*.
- **Object Clustering** — When more than one input layer is provided to DFM Property, the layer shapes are grouped in clusters, where each cluster has one shape from the first input layer (the *primary* layer) and zero or more shapes from the other layers (the *secondary* layers). The basic clustering modes are INTERSECTING, OVERLAP, and

NODAL (by node number). By default, measurement function results for secondary layer objects are summed over a cluster. See “[Clustering Methods in DFM Property](#)” in the *SVRF Manual*.

- **Object Filtering** — Filtering is provided by specifying a constraint for a property expression. Layer shapes that do not satisfy the constraint are not included in the output layer.
- **Expression Support** — The expressions used to calculate property values support many standard math functions, conditional expressions, and expression chains (used to provide an alternate property definition if an expression fails to evaluate). See “[Rules for Writing DFM Expressions](#)” in the *SVRF Manual*.
- **Property Value Access** — The DFM Property operation can read DFM property values that were attached to layer shapes by another DFM Property operation. See “[Property Access Function Summary](#)” in the *SVRF Manual*.

There are several operations that are similar to DFM Property; these are summarized in “[Advanced DRC Toolset](#)” on page 135. [DFM Property Select Secondary](#) is useful in certain cases because the output layer consists of shapes from a secondary layer rather than the primary layer.

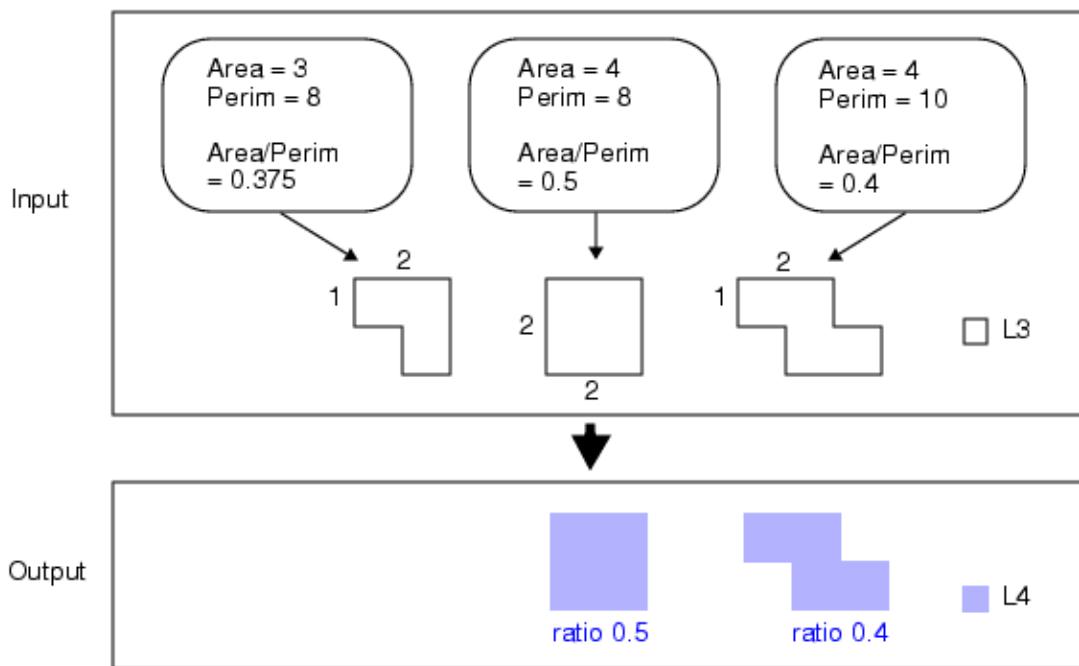
Example 1: Find Ratio of Polygon Area to Perimeter

This example demonstrates the use of measurement functions and object filtering. It takes a derived polygon layer L3 and uses the [measurement functions](#) (AREA and PERIM) to calculate the area divided by the perimeter for each geometry. Only geometries with an AREA/PERIM value greater than 0.4 are written to the output layer.

```
L4 = DFM PROPERTY L3 [ratio = AREA(L3)/PERIM(L3)] >= 0.4
```

A property named “ratio” is attached to each geometry on the output layer L4. The constraint “ ≥ 0.4 ” filters the output geometries to those with a ratio property value greater than or equal to 0.4.

Figure 4-10. Polygon Area to Perimeter Ratio



The left-most L3 polygon has a ratio property value less than the constraint value of 0.4, so it is excluded from the output layer L4.

Example 2: Counting Contacts

This example uses a multilayer DFM Property operation to return the number of contacts in each source-drain region.

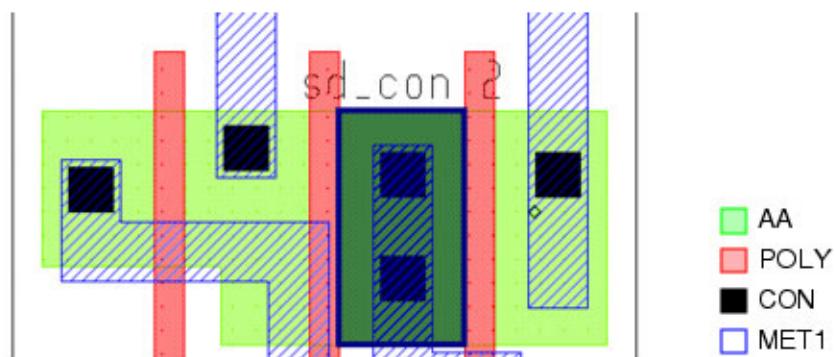
```
LAYER CON
gate = POLY AND AA
imp = AA NOT POLY
sd = imp INTERACT POLY

sd_count = DFM PROPERTY sd CON INTERSECTING [sd_con = COUNT(CON)]
```

The next figure shows an output result in Calibre DESIGNrev where the count is 2.

Note

To enable display of property names and values when highlighting results, from Calibre RVE, choose **Setup > Options > Highlighting** expand the DRC/DFM Highlighting area, and ensure the **While highlighting, show ... Properties** option is checked.

Figure 4-11. Number of Contacts in Source-Drain Regions

The preceding example is also in “[Counting Contacts in Source-Drain and Gate Regions](#)” in the *Calibre Solutions for Physical Verification* manual. That example also counts contacts in the gate region, and uses the property access function PROPERTY() to read the property sd_con on the sd_count layer.

Introduction to DFM Function

The DFM Function statement allows you to create a custom mathematical expression or lookup table for use in DFM Property statements.

Math Expression Examples

Here are examples of how to code math expressions.

Example 1

A DFM Function statement defines a function named Ratio that accepts a single layer and returns the area divided by the perimeter for that layer. The function is then called in a DFM Property expression.

```
DFM FUNCTION [Ratio(LAYER L1) AREA(L1)/PERIM(L1)]  
M1_ratio = DFM PROPERTY M1 [area_perim_ratio = Ratio(M1)]
```

Example 2

This example is an extension of “[Example 2: Counting Contacts](#)” on page 138. It uses a DFM Function statement to calculate the ratio of contact area to source-drain area. The DFM Function statement defines the CON_AREA function, which has two inputs.

```
LAYER CON // contact layer
gate = POLY AND AA
imp = AA NOT POLY
sd = imp INTERACT POLY

DFM FUNCTION [CON_AREA(LAYER L1,LAYER L2) AREA(L2)/AREA(L1)]

sd_count = DFM PROPERTY sd CON INTERSECTING
[sd_con = COUNT(CON)]
[contact_area = CON_AREA(sd,CON)]
```

Example 3

In the following code, if AREA(MET1) is 0.1, the function returns 12. If AREA(MET1) is 0.2, the functions returns 16. If the value of AREA(MET1) falls between values specified in the table, the function returns a value based on linear interpolation. In this case, if AREA(MET1) is 0.15, the function returns 14.

```
DFM FUNCTION [ F(NUMBER X) TABLE LINEAR { 0.0 10 0.1 12 0.2 16 0.3 22 0.4
30 0.5 46 0.6 54 0.7 60 0.8 66 0.9 72 1.0 76 } ]
met1_f = DFM PROPERTY MET1 [A2 = F(AREA(MET1)) ]
```

Function F returns a value from a specified lookup table that corresponds to the input value. The table values occur implicitly in pairs. The initial value of a pair is the input, or lookup, value. The second value of a pair is what the input value is mapped to by the function. The interpolation method for lookup values not defined in the table is defined by a keyword.

Interpolation may also be done using a bicubic algorithm by replacing the LINEAR keyword with SPLINE.

Related Topics

[DFM Function \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Introduction to DFM RDB

DFM RDB is a layer operation that writes original or derived layer data to ASCII RDB format, including any attached properties. The DFM RDB operation must be contained inside a rule check. In a calibre -dfm run, the output RDB is stored in the DFM database and is not stand-alone as it is in a calibre -drc run.

In the following code, the eqdrc_err layer, which is a derived layer created with a DFM Property operation, is output to a DFM RDB file named *eqDRC.rdb*. The layer is contained in a check named via_count_gate_width.

```
eqDRC_rule {
    DFM RDB eqdrc_err eqDRC.rdb CHECKNAME via_count_gate_width
    COMMENT "Wide metal spacing Rule: Use the property values to
            determine the error magnitude and adjustments needed to fix
            the violation"
}
```

You can load the *eqDRC.rdb* file into Calibre RVE to browse the results of the check. Calibre RVE displays DFM expression properties associated with the check in the results pane.

Related Topics

- [Using Calibre RVE for DRC \[Calibre RVE User's Manual\]](#)
- [DFM RDB \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Introduction to DFM Copy and DFM OR

DFM Copy is more flexible than the Copy operation in its handling of input layers. DFM OR performs an OR operation on the input polygon layers and includes DFM properties from the input layers on the output layer.

DFM Copy

DFM Copy is a layer operation that copies one or more input layers of any type (original, derived polygon, derived edge, or derived error) to a new layer. This is more flexible than the Copy operation. In addition, the output layer can be a different layer type than the input layer.

The operation can process derived error layers in one of several ways:

- Convert error clusters into polygons and output a polygon layer.
- Convert error clusters into individual edges and output an edge layer.
- Convert error clusters that consist of two edges into a single edge and output an edge layer that is the centerline between the error cluster.

DFM Copy copies DFM properties on the input layer to the output layer for single input layer operation without the CLUSTER keyword.

DFM OR

DFM OR performs an OR operation on the input polygon layers and attaches DFM properties from the input layer shapes to the output shapes. Each shape on the output layer includes the complete set of property names that exist on the input layers. Optional keywords specify the method for merging properties.

A common application of DFM OR is dividing a polygon layer into component layers according to some criteria, deriving properties separately for the component layers, then using DFM OR to combine the component layers. This process is useful when the property derivations are different for the component layers. See the command reference for details and examples.

Related Topics

- [DFM Copy \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

DFM OR [Standard Verification Rule Format (SVRF) Manual]

Chapter 5

Hierarchical Calibre nmDRC

Calibre nmDRC-H exploits the hierarchy in a design to reduce processing time, memory use, and DRC result counts when compared to flat DRC applications.

Note

 Other Calibre tools also have hierarchical versions. For layer derivation and rule checking, information in this section applies to those tools also.

Calibre nmDRC-H can use the same rule file as its flat counterpart, Calibre nmDRC. Calibre nmDRC and Calibre nmDRC-H behave similarly when it comes to finding design rule errors. However, Calibre nmDRC-H uses hierarchical processing algorithms that handle cell and placement structures that reduce the overall error count by reporting errors per-cell wherever possible rather than across the entire flat design. There are some SVRF specification statements that apply only to Calibre nmDRC-H. As with any rule file structure, these statements are ignored by the tool when the run mode does not use them.

Calibre nmDRC-H imposes no design restrictions concerning geometry overlapping cell placements or overlaps of cell placements.

The basic idea behind processing layers hierarchically is discussed under “[Hierarchical Database Construction](#)” on page 93.

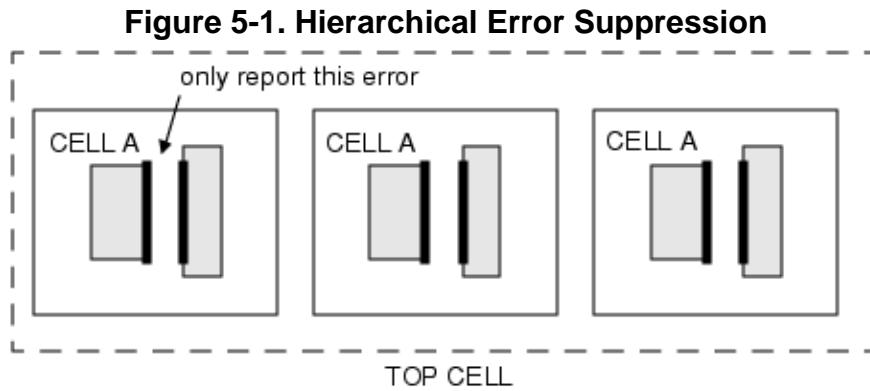
Calibre nmDRC-H Results Data Storage	143
Flat Versus Hierarchical Results Presentation	145
Multithreaded Modes of Operation	146
Hierarchical Operation Efficiency	147
Flat Instantiations	148
Path Length Variances	148
Layer Area Printing.....	150
Text Objects in Mask Results Output	150
Calibre nmDRC-H Use of Hcells	151

Calibre nmDRC-H Results Data Storage

Calibre nmDRC-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 shapes up the hierarchy 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 Calibre nmDRC-H incorporates a natural error suppression device.

For example, consider the design in [Figure 5-1](#).



The design consists of three placements of cell A, which contains a spacing error between the two shapes. Calibre nmDRC-H generates the error when analyzing cell A; the error is independent of context. Therefore, Calibre nmDRC-H stores the error geometry in the cell template of A and reports the error only once (at a location corresponding to the hierarchically lowest, left-most placement of the cell). 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 nmDRC-H implicitly recognizes that two errors are repeated errors and does not report them.

If it is not possible to fully examine edge data within the context of a certain cell, the shapes are promoted up the hierarchy until the cell context is sufficient to completely contain the data under examination. When this is achieved, promotion of the shapes stops, and the edge data is processed in the promoted location in a similar way to what is described in the preceding paragraph.

By default, Calibre nmDRC-H generates a DRC results database with the same format as Calibre nmDRC does. Calibre nmDRC-H also transforms all DRC results into the top-level cell space, consistent with the flat applications (the results can be presented at the containing cell level, if desired). 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 nmDRC-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 nmDRC-H chooses one placement of cell A throughout the entire hierarchy. Calibre nmDRC-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 hierarchically lowest, leftmost placement.

By contrast, flat DRC runs cause all geometry to be evaluated at the uppermost level of the design and results are reported there, thus multiplying the number of results by the number of flattened placements of cells.

Near the end of a hierarchical run transcript, the total results are reported from the hierarchical standpoint first, with the calculated flat count in parentheses:

```
--- TOTAL RESULTS GENERATED = 2 (6)
```

Related Topics

[Calibre nmDRC Results](#)

[Hierarchical Calibre nmDRC Results Database](#)

Flat Versus Hierarchical Results Presentation

Calibre nmDRC presents flat and hierarchical results differently in the DRC Summary Report and results databases. Hierarchical presentation offers a variety of settings to assist in result interpretation.

Table 5-1. Flat Versus Hierarchical Results Presentation

Output	Flat Run	Hierarchical Run
DRC Summary Report	Flat counts	Shows hierarchical result counts first, followed by estimated flat result counts in parentheses. Cell-specific counts are not shown by default.
		To show per-cell statistics use DRC Summary Report HIER option.
DRC Results Database and RDBs	Flat results	Stores each DRC result only once. Location is inside the top-level boundary of the hierarchically lowest, leftmost placement of the error cell. No cell-specific data is stored.
		To store cell-based results use DRC Cell Name YES CELL SPACE XFORM

Table 5-1. Flat Versus Hierarchical Results Presentation (cont.)

Output	Flat Run	Hierarchical Run
Calibre RVE results presentation	Flat view	Default shows results at the top level.
		To show cell-based results, select Highlight > Highlight in Context .

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

Related Topics

[DRC Cell Name \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Summary Report \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Using Calibre RVE for DRC \[Calibre RVE User's Manual\]](#)

Multithreaded Modes of Operation

Calibre nmDRC-H has multithreaded modes of operation that greatly reduce runtimes. Multithreaded (MT) mode uses multiple CPUs on the same host and is enabled through the -turbo command line option. MTflex mode uses multiple CPUs on a network of hosts and is enabled with the -remote or -remotefile command line options. These modes apply to other Calibre tools as well.

Hyperscaling

Hyperscaling is a mode of operation that allows parallel processing of layer operations in an MT or MTflex run. It is enabled using the -hyper command line option and requires no additional licenses than the usual MT or MTflex requirements. Hyperscaling typically results in twice the savings in real time over a multithreaded run and improved scaling of processor utilization. If you have hardware that supports MT or MTflex run modes, you should use hyperscaling unless you have compelling reasons not to (a minimum of four CPUs is recommended).

Example command line invocations are these:

```
calibre -drc -hier -turbo -hyper rules
calibre -drc -hier -turbo -remotefile config.txt -hyper rules
```

Note

 When using hyperscaling it is imperative that you use a correctly-specified Layout Base Layer statement in the rule file. (Layout Top Layer may be used, but Layout Base Layer is preferred.) Failure to do so often results in reduced performance, which is true for any Calibre nmDRC-H run.

Hyperscaling counts hierarchical objects differently from traditional hierarchical runs. Specifically, hyperscaling flattens internally-recognized, very small cells (vias and contacts). Hyperscaling also flattens internally-recognized top-level cells that are identified based upon your Layout Base Layer statement (or Layout Top Layer statement, if applicable). This means object counts can differ between hyperscaling and traditional hierarchical runs. This is an expected behavior. If results counts differ between a hyperscaling run and a non-hyperscaling run, this does not mean one run has missed results that the other has found. Hyperscaling selectively flattens some data and this can affect results counts.

Hyperscaling can perform rule checks in a different order than appears in the rule file. Non-hyperscaling runs follow the rule file order.

Related Topics

[Distributed Calibre Overview \[Calibre Administrator's Guide\]](#)

[Calibre nmDRC and Calibre nmDRC-H Command Line](#)

[Licensing for Multithreaded Operations \[Calibre Administrator's Guide\]](#)

[Layout Base Layer \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Hierarchical Operation Efficiency

Layer operations differ in their performance. There are some behaviors to be aware of when choosing various layer operations and options.

- Dimensional check operations [Enclosure](#), [External](#), and [Internal](#) are slower than most other operations because of the computational overhead of having to measure between many edges. This can be especially noticeable 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 nmDRC-H performs the one-layer Boolean operations [AND](#) (with a constraint other than ≥ 1 or > 1) and [XOR](#), in addition to [Magnify](#) and [Rotate](#), in a flat manner. This results in an exclusive flat instantiation of the output layer, and is costly for performance.
- Calibre nmDRC-H performs the [Angle](#) operation flat whenever the measurement constraint includes zero (0) but not 90, and vice-versa. This is costly for performance.
- You can perform the [Shift](#), [Grow](#), and [Shrink](#) operations in a cell if all of the placements of that cell have a consistent rotation or reflection 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.

- [Net Area Ratio](#) accumulation layers are flattened prior to printing in [Net Area Ratio Print](#).
- The [Rectangles](#) operation is more complex in hierarchical mode as compared to flat mode and is one of slower hierarchical operations.

Flat Instantiations

Calibre nmDRC-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 nmDRC-H creates flat instantiations for layers in one of three ways:

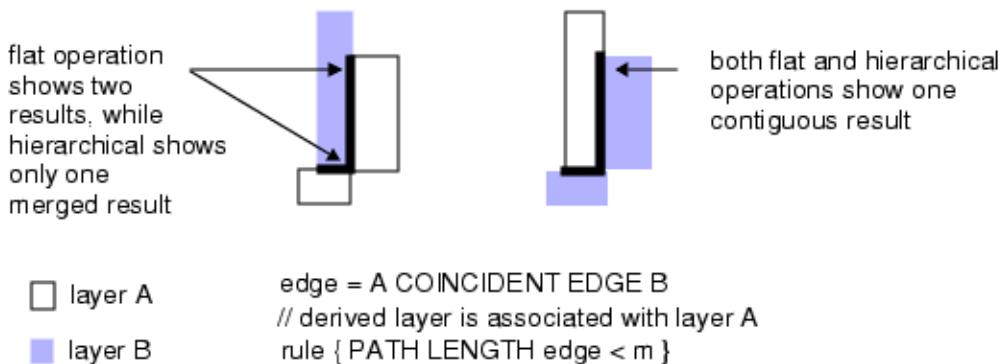
- The result layer of the [Flatten](#) operation has an exclusive flat instantiation.
- The result layer of a layer operation that is not supported hierarchically (or that has an input layer with a flat instantiation) has an exclusive flat instantiation.
- If an input layer to an operation is not supported hierarchically, or another input layer has a flat instantiation, Calibre nmDRC-H converts it to a dual instantiation. Another possibility is that Calibre nmDRC-H may create the input layer as a temporary flat copy.

Any flattening required to support non-hierarchical layer operations or co-existence of flat and hierarchical instantiations is completely automatic.

Path Length Variances

In certain uncommon situations, the results from Path Length operations in hierarchical runs can differ from flat runs. Such cases involve a singularity point where polygon corners meet.

Figure 5-2. Hierarchical Versus Flat Results



In [Figure 5-2](#), there are two layer A polygons on the left that touch at corners. Flat DRC shows two results, which is expected. Hierarchical DRC shows only one result because the derived edge layer is a merged layer. The hierarchical engine does not maintain the original layer A polygon associations with the derived edge layer, so there is only one result in this case. The difference in result count only occurs where there is a singularity point for layer A polygons, as shown in the diagram.

The diagram on the right shows behavior that is expected and is simply for comparison. You would see a similar result if the A and B layers were reversed in the Coincident Edge operation. This fact is used in the layer derivation shown next.

To reconcile the differing result counts for situations such as in the diagram on the left, you could do something similar to the following:

```

multi_A = A TOUCH B > 1
// multiple A polygons touch a B polygon
// possible singularity point exists on multi_A

single_A = A NOT TOUCH B > 1 // no singularities on this layer

edge1 = multi_A COINCIDENT EDGE B
// flat operation shows 2 results at a singularity

edge2 = B COINCIDENT EDGE multi_A
// input layers reversed; flat operation shows 1 result at a singularity

edge1 { PATH LENGTH edge1 < m }
edge2 { PATH LENGTH edge2 < m }
// result counts may differ but output is the same

edge = single_A COINCIDENT EDGE B
// no special treatment for singularities needed for layer single_A

edge { PATH LENGTH edge < m }

```

Related Topics

[Path Length \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Layer Area Printing

When first generating the layer, flat applications print the total area of a polygon-type layer in the transcript with other related statistics. Hierarchical applications do not do this by default because of the time it requires. However, you can specifically request this information with the DRC Print Area specification statement.

This statement directs hierarchical Calibre applications to print the flat area of a layer when generating the layer. This prints the area, along with other relevant statistics for the layer.

Related Topics

[DRC Print Area \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Print Perimeter \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Text Objects in Mask Results Output

You can use the DRC Map Text specification statement to cause Calibre nmDRC-H to transfer all text objects in the input layout database to a mask data results database. The text objects have the same hierarchy in both the input and results databases, unless DRC-H expands or flattens a placement during hierarchical processing. In that case, DRC-H moves the text up the hierarchy, as appropriate.

You can use the DRC Map Text Layer statement in conjunction with DRC Map Text YES to specify the layers from which to write input text objects to the output.

Calibre applications retain TEXTTYPE attributes from geometric input layout databases by default even if a text layer gets mapped to a different layer. If Calibre maps text objects with Layer Map TEXTTYPE specification statements, then text in the geometric DRC results database is on the target layer(s) of the mapping, but the original texttype attributes are maintained. For example:

Example 5-1. Mapping Text Objects for Output

```
LAYER MAP 10 TEXTTYPE 1 10      // layer 10.1 to Calibre layer 10
LAYER mapped_text 10
TEXT LAYER 10                  // Calibre layer 10 is a text layer

DRC MAP TEXT YES              // Output text objects to mask results
DRC MAP TEXT LAYER mapped_text // Output text layer maintains
                                // original datatype 1
```

Here, text objects on layer 10 texttype 1 are mapped to Calibre layer 10. The text objects in the output retain their texttype of 1 in the output. If you specify a different target texttype from the input texttype using the Layer Map statement, then the target texttype is used for output.

Related Topics

[DRC Map Text \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Map Text Layer \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Layer Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Calibre nmDRC-H Use of Hcells

The Hcell statement is used by Calibre nmLVS-H to denote the correspondence between cells in the layout and cells in the source. In circuit extraction, hcells in the layout are preserved. A similar effect occurs in hierarchical DRC runs.

The Hcell or Layout Preserve Cell List specification statement should only be in a rule file for Calibre nmDRC-H if the desired flow is as described herein, that is, future presentation of mask results layers to LVS. These statements should not be used simply to prevent automatic cell expansion by Calibre nmDRC-H. Calibre nmDRC-H generates a warning if Hcells are present in a layout to let you know that performance may be severely impacted.

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 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 nmLVS-H. Therefore, the hierarchical database construction phase for Calibre nmLVS-H connectivity extraction does not automatically expand any layout cell in a rule file Hcell statement, hcell list, or Layout Preserve Cell List statement regardless of the performance cost.

In many application flows for Calibre nmDRC-H, the goal is database modification, not necessarily traditional DRC checking. The output database(s) created by Calibre nmDRC-H may be subject to future LVS verification. In that event, it is not desirable that the DRC portion of the process expand cells (which essentially means they are removed from the hierarchy) that may later be designated in the LVS portion as hcells. Therefore, Calibre nmDRC-H also inhibits expansion of any layout cell in a rule file Hcell or Layout Preserve Cell List specification statement, regardless of the performance cost.

This does not mean that a DRC operation will not move data across an hcell boundary. If you perform data scaling operations in Calibre nmDRC-H, it is very likely that data will be promoted from some cells. Hcells are preserved in the hierarchy, but data can still be promoted out of an hcell.

Related Topics

[Hcell \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Layout Preserve Cell List \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Chapter 6

Dimensional Check Operations

Dimensional check operations represent the core design rule checking capability of Calibre nmDRC. 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.

Primary Dimensional Check Operations	153
Secondary Keywords	155
Edge Measurement	156
Edge Cluster Generation	171
Interval Constraints for Output Suppression	175
Appropriateness Criteria	177
Edge Shielding	179
Intersection Criteria	181
Edge Breaking	182
Polygon Containment Criteria	183
Output Modes	185
False Measurement Reduction	189
Measurement Tolerances	191

Primary Dimensional Check Operations

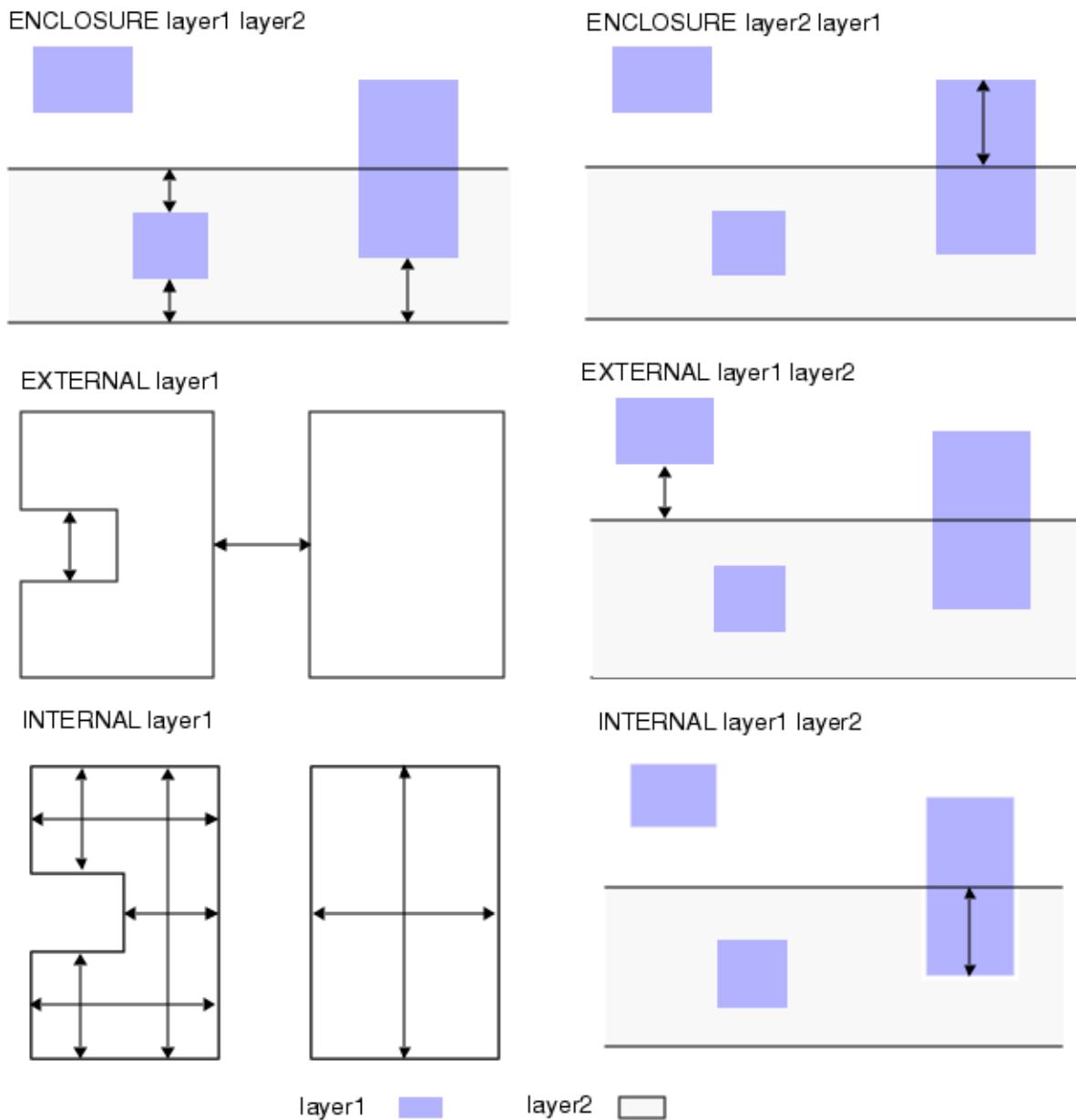
There are three primary dimensional check operations: ENClosure, EXTernal, and INTernal.

- The Enclosure operation is for enclosure or extension checks.
- The External operation is for spacing checks.
- The Internal operation is for width or overlap checks.

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

[Figure 6-1](#) 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. In these examples, the full rule check syntax is not used.

Figure 6-1. Measured Edges in the Dimensional Check Operations



The [Rectangle Enclosure](#), [\[Not\] With Neighbor](#), [TDDRC](#), and [DFM Space](#) operations are based upon the three primary dimensional check operations.

Related Topics

[Enclosure \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[External \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Internal \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

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 this chapter.

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.

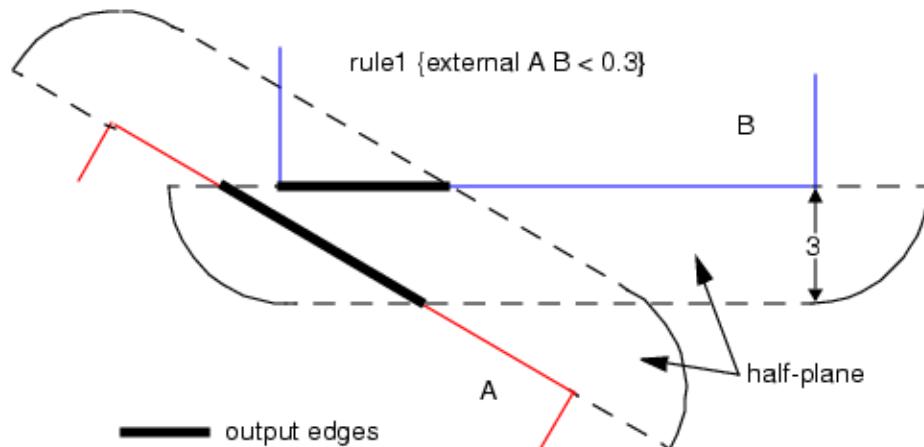
The edge pairs of dimensional check operations consist only of those portions of measured edges that conform to the measurement constraint. This edge-measurement method proceeds as follows (refer to [Figure 6-2](#)).

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 0.3 microns are output. Using the Euclidean metric (refer to “[Metrics](#)” on page 159), 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 0.3 microns of edge A; a similar region consists of all such points around edge B. (The tool 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 0.3 microns but any other type of allowed constraint or value works similarly.

Figure 6-2. Euclidean Generation of Output Edges



Measurement Region Construction	156
Metrics	159

Measurement Region Construction

To construct the region about an edge, a boundary forms on either the outside or inside of the edge, as specified by the operation. This area is referred to as a half-plane. The shape of this

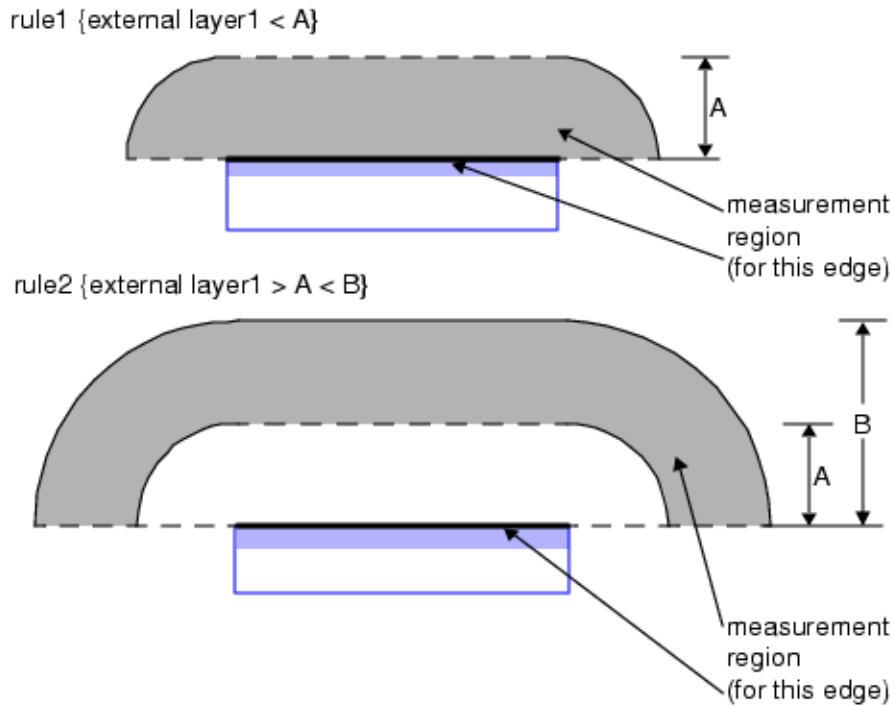
boundary is determined, in part, by the numeric value within the operation's constraint. The boundary consists of all points in the half-plane whose distance from the edge is the numeric value of the constraint.

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 separation A and strictly inside the boundary with separation 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 separation distances A and B .

Figure 6-3 shows region construction about the outside of an edge. The first region assumes the constraint $x < A$ and the second region assumes the constraint $A < x < B$. No points on the line containing the edge are considered to be within the region.

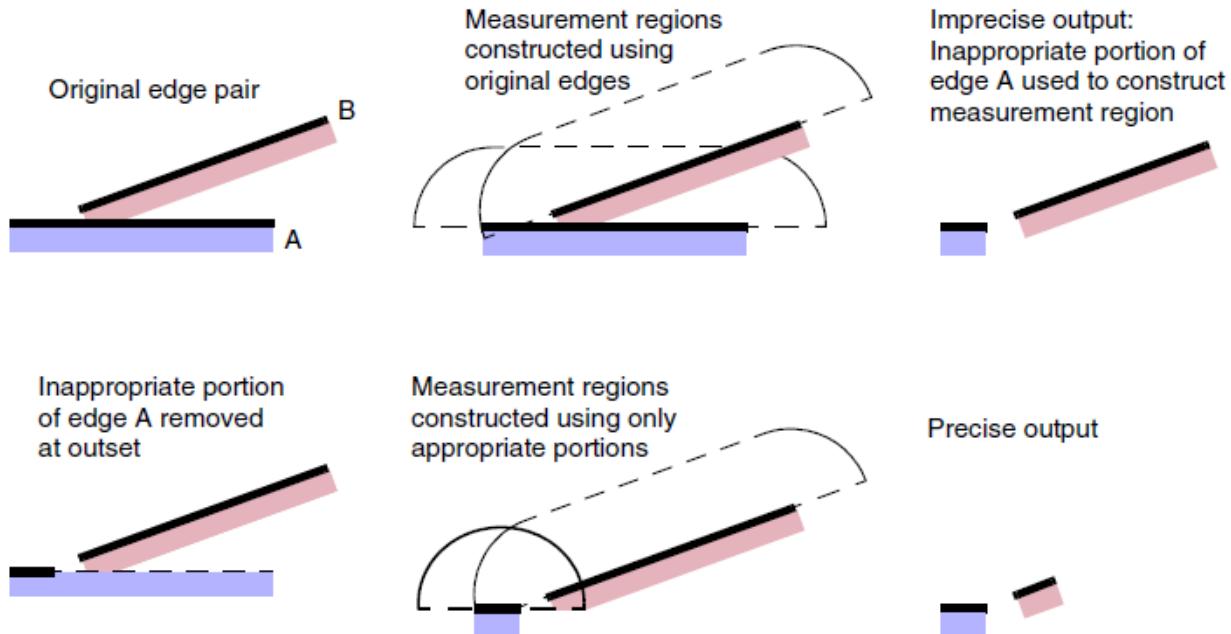
Figure 6-3. Measurement Region Formation



The measurement and output generation for two edges A and B are calculated by determining the portions of each edge which intersect the region about the other edge. The portions become the output of the dimensional check operation. There is no output if neither edge intersects the region about the other edge.

Prior to region construction, the edges in a measurement pair are trimmed, if necessary, so that any portion of an edge which does not actually face the other edge is discarded. This ensures that any portion of an edge in the measurement pair which, by itself, is not appropriate for measurement with the other edge in the pair, does not contribute to the region construction about its edge. The measurement region is then constructed about the remaining portion of the edge. [Figure 6-4](#) illustrates the difference in output when original edges versus trimmed edges are used to construct measurement regions.

Figure 6-4. Edge Trimming Prior to Region Construction



Metrics

There are four standard forms of metrics used with dimensional check operations: Euclidean, Square, Opposite, and Opposite Extended *value*.

Euclidean — 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 — Forms a region with right-angle boundaries that extend past the corners of the selected edges.

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

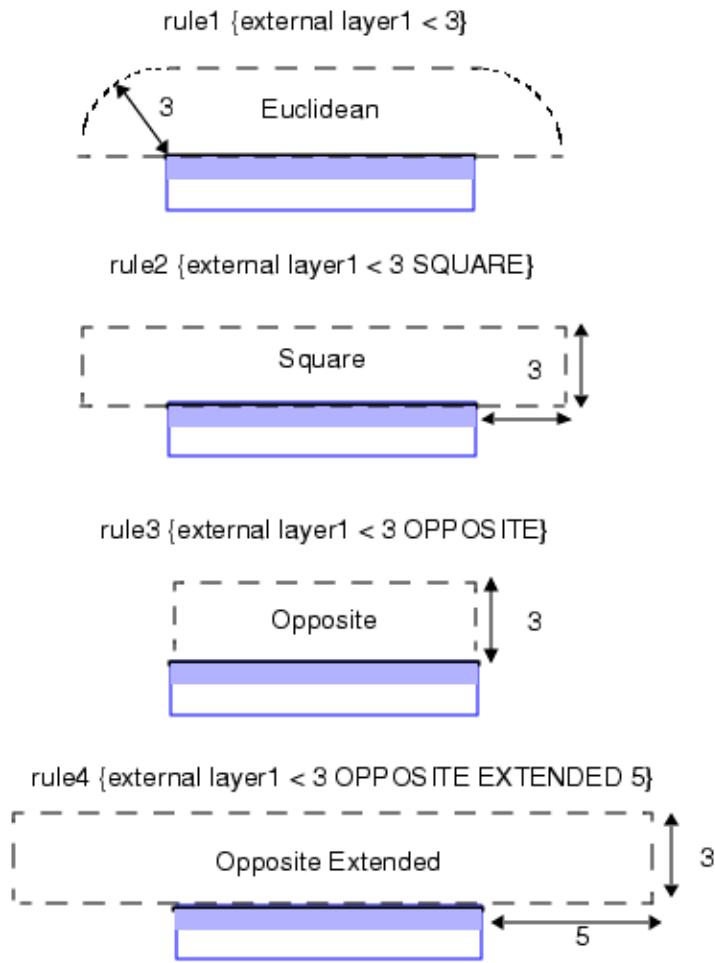
Opposite Extended *value* — Forms a region with right-angle boundaries that can extend past the corners of the selected edges, dependent upon an extension value you specify. This metric allows non-commutative measurements, where the measurement from edge A to edge B does not produce the same output as 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. The metrics (except for Euclidean) 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 6-5 shows measurement region construction for the four standard metric types using the EXTernal operation.

Figure 6-5. Measurement Regions for Basic Metrics



The following sections provide additional information about metrics:

Special Considerations for the Opposite Metric	160
Advanced Metrics for Specialized Applications	163
Opposite Symmetric Metrics	164
Unidirectional Metrics	166
Square Orthogonal Metric	168

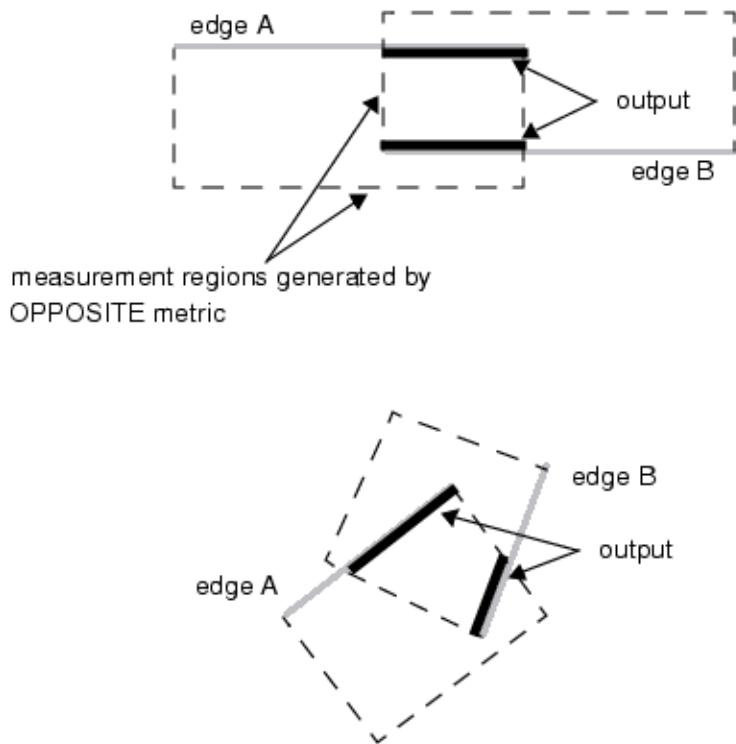
Special Considerations for the Opposite Metric

The main 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 output (REGION keyword).

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, and the measurement region from edge B to edge A intersects A in more than one point, as shown in [Figure 6-6](#).

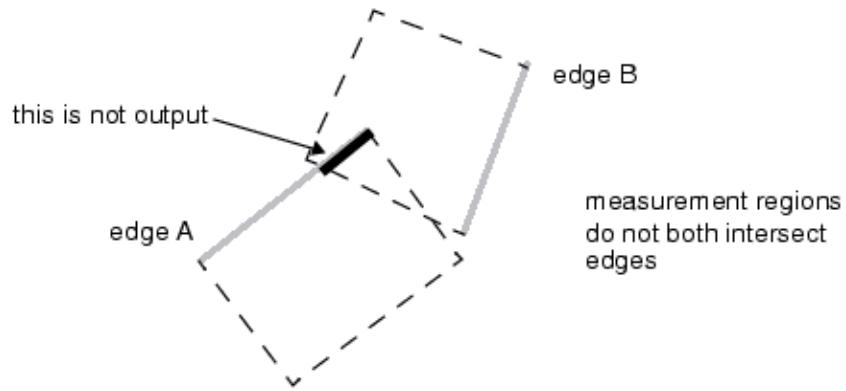
When both measurement regions for the Opposite metric have non-trivial intersections with opposing edges, this is defined as commutative measurement.

Figure 6-6. Commutative Measurement



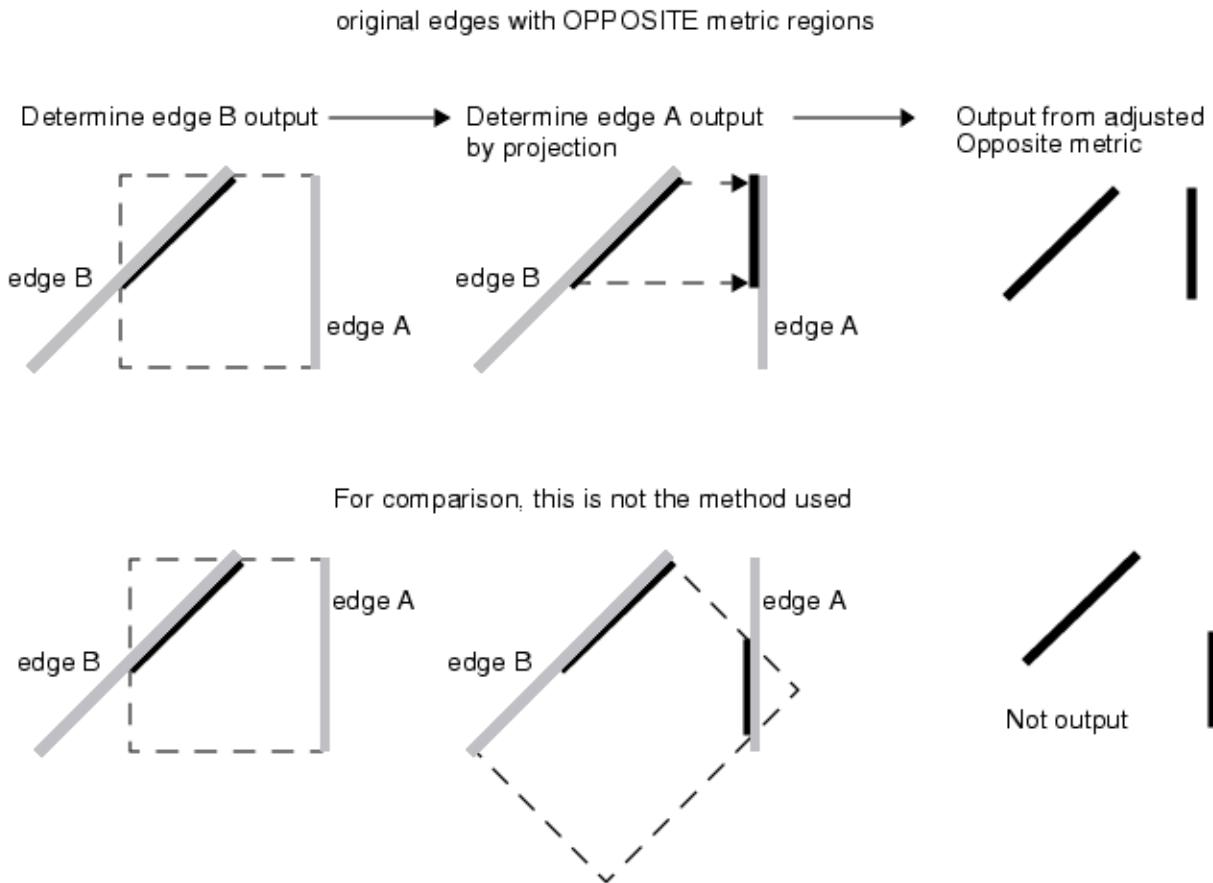
Edges that are not parallel to each other and not orthogonal to the database axes can result in non-commutative measurements, as shown in [Figure 6-7](#). There is no output from the Opposite metric in this case, or in any other case where such measurements occur.

Figure 6-7. Non-Commutative Measurement



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 by the measurement region from edge A. This produces output from B. Output from A is the projection of output from B onto edge A as shown in [Figure 6-8](#).

Figure 6-8. Output Adjustments for the Opposite Metric



Although it would be intuitive to think the output in this case would be as shown in the second set of figures, the first set of figures shows the actual output.

Advanced Metrics for Specialized Applications

Certain specialized metrics are used mostly for OPC applications and mask data preparation. They are not generally used for typical DRC checks, but can occasionally be useful in such applications.

The first set are the Opposite Symmetric metrics, which are discussed in detail under “[Opposite Symmetric Metrics](#).”

Opposite Symmetric — Applies special processing in addition to the Opposite metric for edges that are not intersecting, not perpendicular, and are not parallel.

Opposite FSymmetric — This metric is similar to Opposite Symmetric; however, it has an additional feature of filling in disjoint edges that are output from Opposite Symmetric.

Opposite Extended Symmetric *value* — This metric is similar to Opposite Symmetric, but uses the Opposite Extended measurement region instead.

Opposite Extended FSymmetric *value* — This metric is similar to Opposite FSymmetric, but uses the Opposite Extended measurement region instead.

The next set of metrics is the [Unidirectional Metrics](#).

Opposite1 — This metric is similar to Opposite Symmetric, but it performs the measurement from the first input layer to the second layer, not in the other direction. It is used when there are two input layers.

Opposite2 — This metric is similar to Opposite Symmetric, but it performs the measurement from the second input layer to the first layer, not in the other direction. It is used when there are two input layers.

Opposite Extended1 *value* — This metric is similar to Opposite Extended Symmetric, but it performs the measurement from the first input layer to the second layer, not in the other direction. It is used when there are two input layers.

Opposite Extended2 *value* — This metric is similar to Opposite Extended Symmetric, but it performs the measurement from the second input layer to the first layer, not in the other direction. It is used when there are two input layers.

The final specialized metric is the [Square Orthogonal 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. This

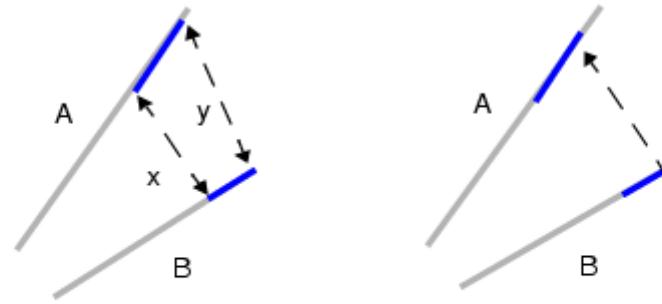
metric is useful for mask layer preparation and simulates mask misalignment in the x- and y-directions.

Opposite Symmetric Metrics

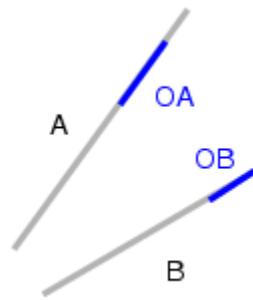
The set of Opposite Symmetric metrics use the Opposite metric for measurement, along with post-processing of the output to achieve better symmetry for non-parallel edges.

Opposite Symmetric is defined by the following steps:

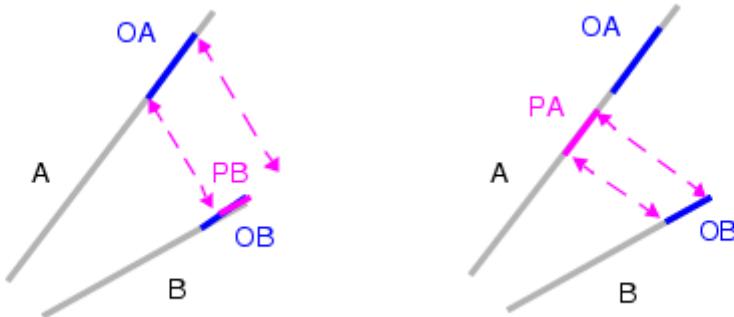
1. Given edges A and B, apply the Opposite metric if A and B are parallel, perpendicular, or intersecting. Output any results from such edges as usual.
2. Otherwise, measure A and B by using the Opposite metric; however, do not perform the special treatment (see “[Special Considerations for the Opposite Metric](#)”) for exactly one non-orthogonal edge, as is done with the Opposite metric. Do not discard the output if the measurement was non-commutative (see [Figure 6-7](#)), 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 (see [Trivial 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. Either OA or OB may be non-existent, depending on the orientation of the edges and the size of the measurement regions.

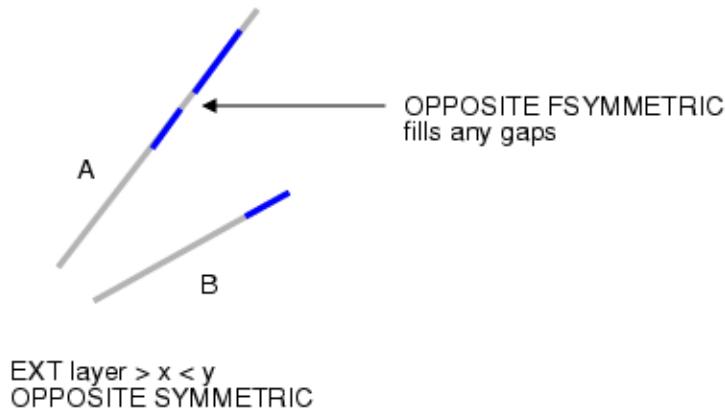


5. Project edge OA, if it exists, onto B, which forms a segment PB. Project edge OB, if it exists, onto A, which forms a segment PA. Discard either PA or PB if they are a result of round-off error.



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 is added to OA + PA, resulting in a single output edge from A. Likewise for B, OB, and PB. See the comment in the next figure.



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

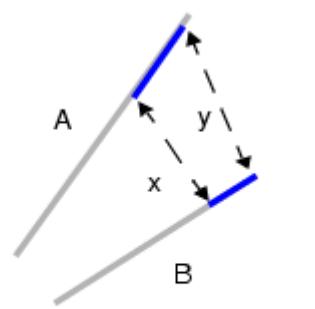
The Opposite Extended FSymmetric metric replaces Opposite measurement with Opposite Extended value measurement in Steps 1, 2, and 3 of the algorithm and performs the Step 6 modification in the same way as Opposite FSymmetric.

Unidirectional Metrics

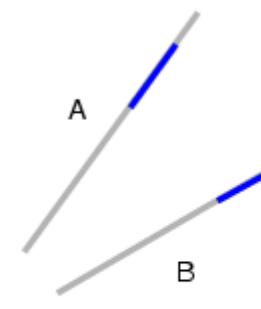
The set of unidirectional metrics use the Opposite Symmetric measurement but measure in one direction only, and then projecting any positive output in the other direction. These metrics assume two input layers.

The OPPOSITE1 metric is defined by the following steps. (If the metric is OPPOSITE2, this operation is reversed, and only coincident edges from the first input layer are retained. Similarly for the succeeding steps, the measurement is taken from the second input layer to the first.)

1. Given two edges, A from layer1 and B from layer2, 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 (see “[Special Considerations for the Opposite Metric](#)”) for exactly one non-orthogonal edge, as is done with the OPPOSITE metric. Do not discard the output if the measurement was non-commutative (see “[Non-Commutative Measurement](#)”), as is done with the OPPOSITE metric. This step can result in zero, one, or two outputs from each edge.

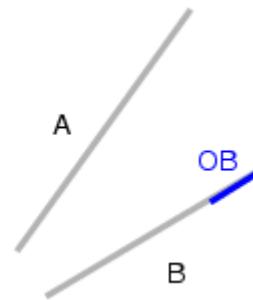


EXT layer1 layer2 > x < y

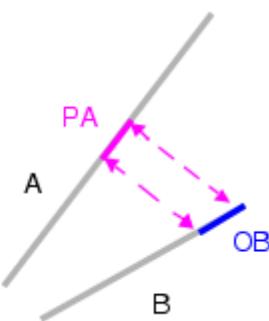


EXT layer1 layer2 > x < y
OPPOSITE

3. Discard all trivial output edges (see “Trivial Edges”). Because of properties within the Opposite metric, there is a one-to-one correspondence between output edges originating from each input edge (even for interval constraints).
4. Discard any output from Step 2 that is coincident with edge A (that is, retain only output coincident with edges from the second input layer).
5. Quit with no output if, after discarding trivial edges, there is no output on B. Otherwise, name the resulting output edge OB.



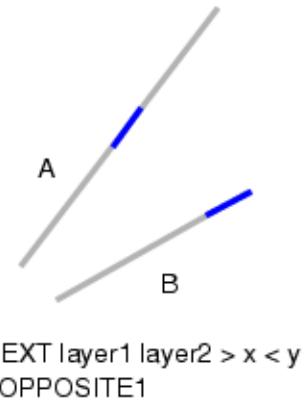
6. Project edge OB onto A, which forms a segment PA. Discard PA if it is trivial.



7. Produce the output of edge A which is PA. Produce the output from edge B, which is OB.

See [Figure 6-9](#) for the final output.

Figure 6-9. OPPOSITE1



The OPPOSITE EXTENDED SYMMETRIC metric replaces the OPPOSITE measurement with OPPOSITE EXTENDED *value* measurement in Steps 1, 2, and 3 of the algorithm.

Related Topics

[Opposite Symmetric Metrics](#)

Square Orthogonal Metric

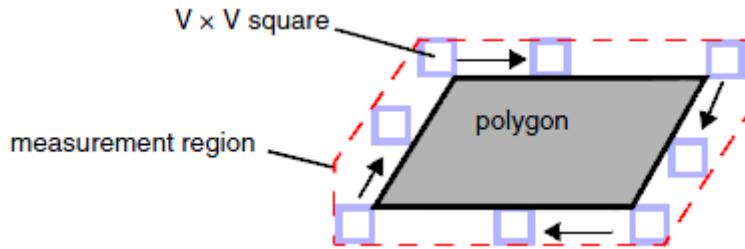
The Square Orthogonal metric for dimensional check operations EXternal, INTernal, and ENClosure simulates simultaneous mask misalignment in the x- and y-directions.

To illustrate, consider the EXternal operation with constraint value V:

EXT layer < V ...

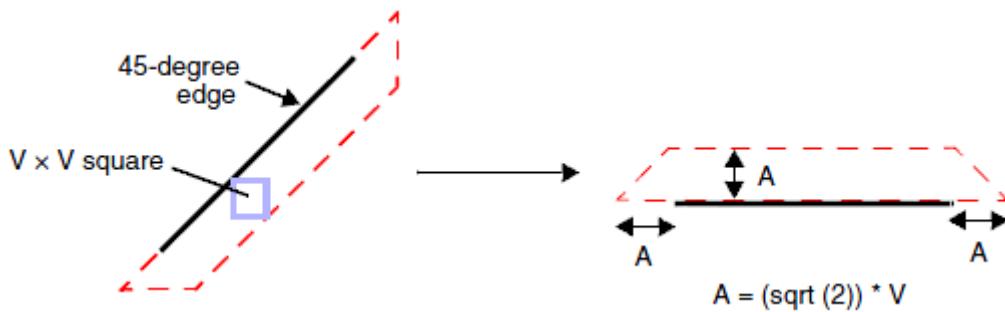
The Square Orthogonal metric simulates the effect of using a square of dimensions $V \times V$ to trace around the perimeter of each polygon on the input layer. At least one corner of the square is always in contact with the polygon as the square traces around the polygon. The square remains orthogonal 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 6-10](#).

Figure 6-10. Square Orthogonal Measurement Region



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 $A = (\sqrt{2}) * V$, 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 6-11.

Figure 6-11. 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 6-12](#)) having endpoints (x_1, y_1) and (x_2, y_2) , define:

$$DX = |x_2 - x_1|$$

$$DY = |y_2 - y_1|$$

$$R = \sqrt{DX * DX + DY * DY}$$

$$A = V * R / \max(DX, DY)$$

$$B = V * |DX - DY| / R$$

Construct an isosceles trapezoid having the following dimensions:

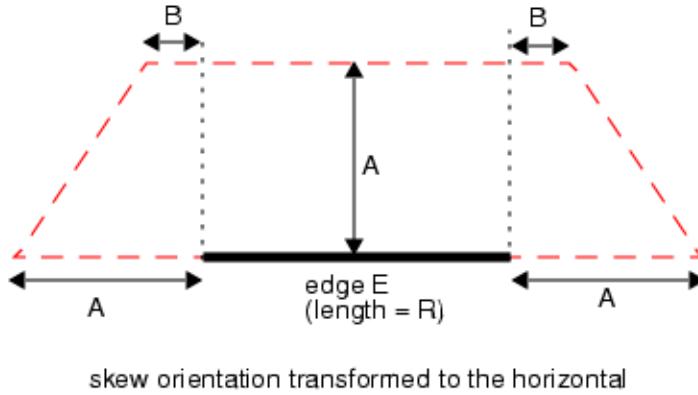
$$\text{base 1} = R + 2A$$

$$\text{base 2} = R + 2B$$

$$\text{altitude} = A$$

The skew edge E is coincident with base 1.

Figure 6-12. 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:

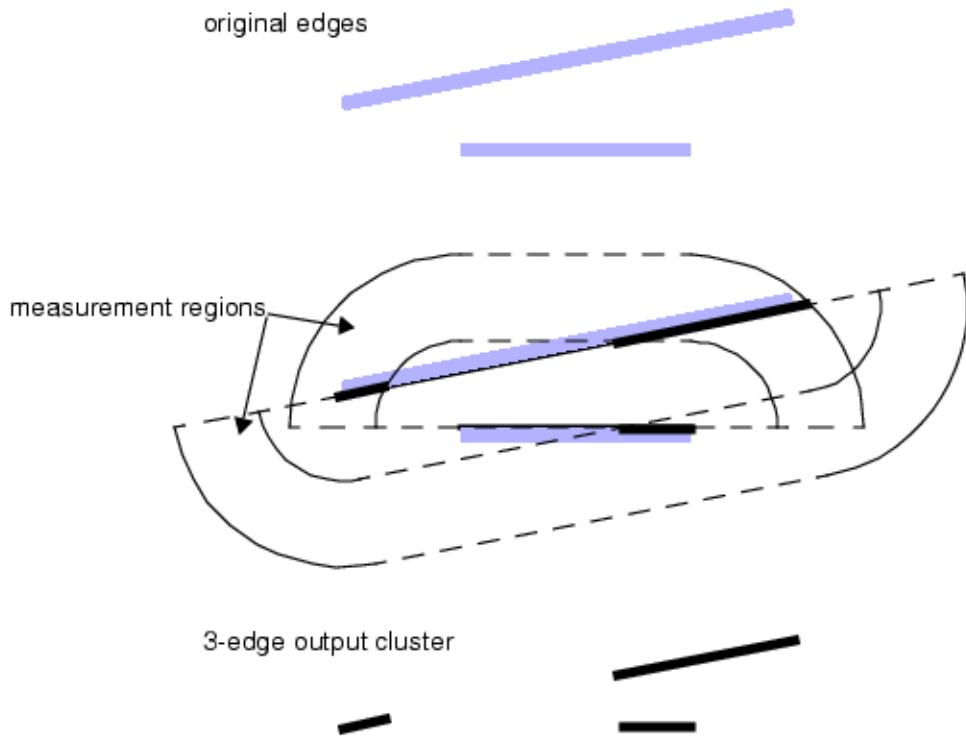
- Edge A intersects the measurement region around edge B, but not vice-versa, or
- Edge B intersects the measurement region around edge A, but not vice-versa.

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 6-13 shows the generation of a three-edge output cluster.

Figure 6-13. 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 in that context.

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

Trivial Edges	172
Four-Edge Output Cluster	173

Point-to-Point Measurement Output	174
---	-----

Trivial Edges

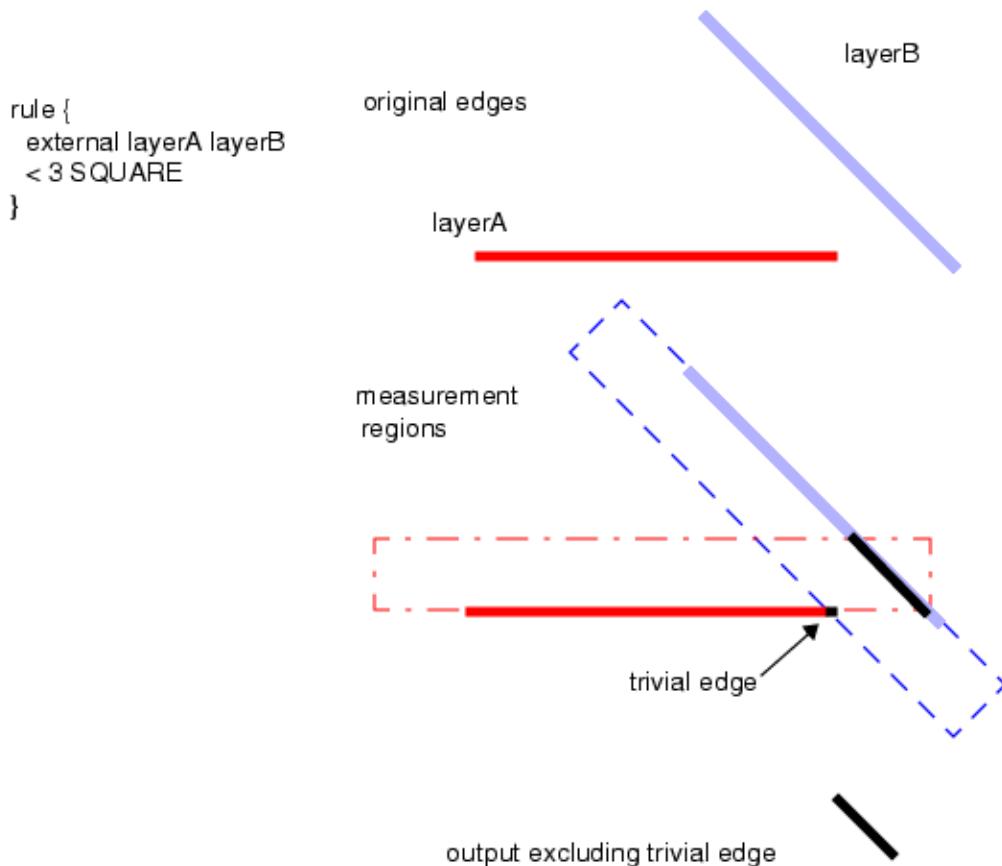
The edge measurement regions can create a trivial edge, which is an edge consisting of only one point. This is the result of measuring two edges when the constraint specifies that the boundary is to be included in the region (for example, $\leq a$), and one of the edges intersects the region around the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

[Figure 6-14](#) 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.

Calibre nmDRC 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, Calibre nmDRC 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.

Trivial edges do not 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 is simply discarded. However, trivial edges can appear as part of error-directed results, and play a role in polygon-directed results, as discussed under “[Point-to-Point Measurement Output](#)” on page 174.

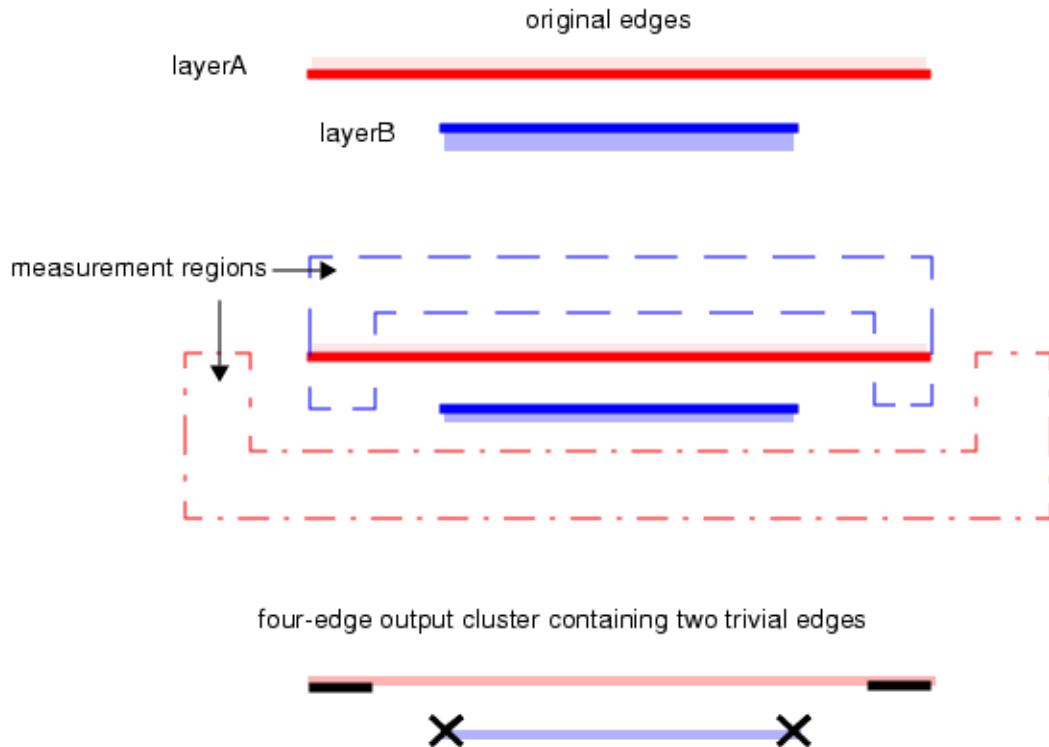
Figure 6-14. Trivial Edge Generation

Four-Edge Output Cluster

Trivial edge generation can lead to a four-edge output cluster, which primarily occurs when there are two output segments from edge A but none from edge B, or vice versa. In this case, Calibre nmDRC 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 6-15](#) 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.

Figure 6-15. Four-Edge Output Cluster

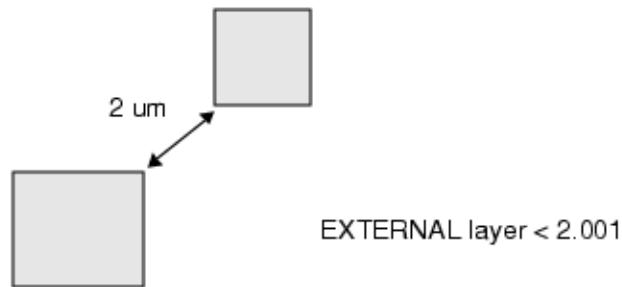


Point-to-Point Measurement Output

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

For example, if your user units are microns and your [Precision](#) is 1000, the situation in [Figure 6-16](#) would result in two output clusters consisting of trivial edges.

Figure 6-16. Point-to-Point Trivial Edge Generation



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

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

The [DRC Extend Trivial Edge YES](#) statement primarily supports error-directed input into multi-patterning operations. This statement directs that point-to-point trivial edge output from the DRC measurement process will be extended as described above and, if not already extended by the above algorithm, in the following scenario:

1. The dimensional check operation is not “output-only”; that is, it does not only feed the DRC results database or a Copy operation which itself only feeds the DRC results database.
2. The dimensional check operation is error-directed.
3. The trivial edge is part of a two-edge cluster.
4. It is a true trivial edge, that is generated as part of the actual measurement process and not created as the result of a non-commutative measurement. (See “[Trivial Edges](#)” on page 172 and “[Special Considerations for the Opposite Metric](#)” on page 160 for related information.)

Interval Constraints for Output Suppression

In conjunctive design rule checking (rule checks where more than one error output operation appears), you may want to suppress redundant errors with interval constraints containing two numerics in the final dimensional check operation.

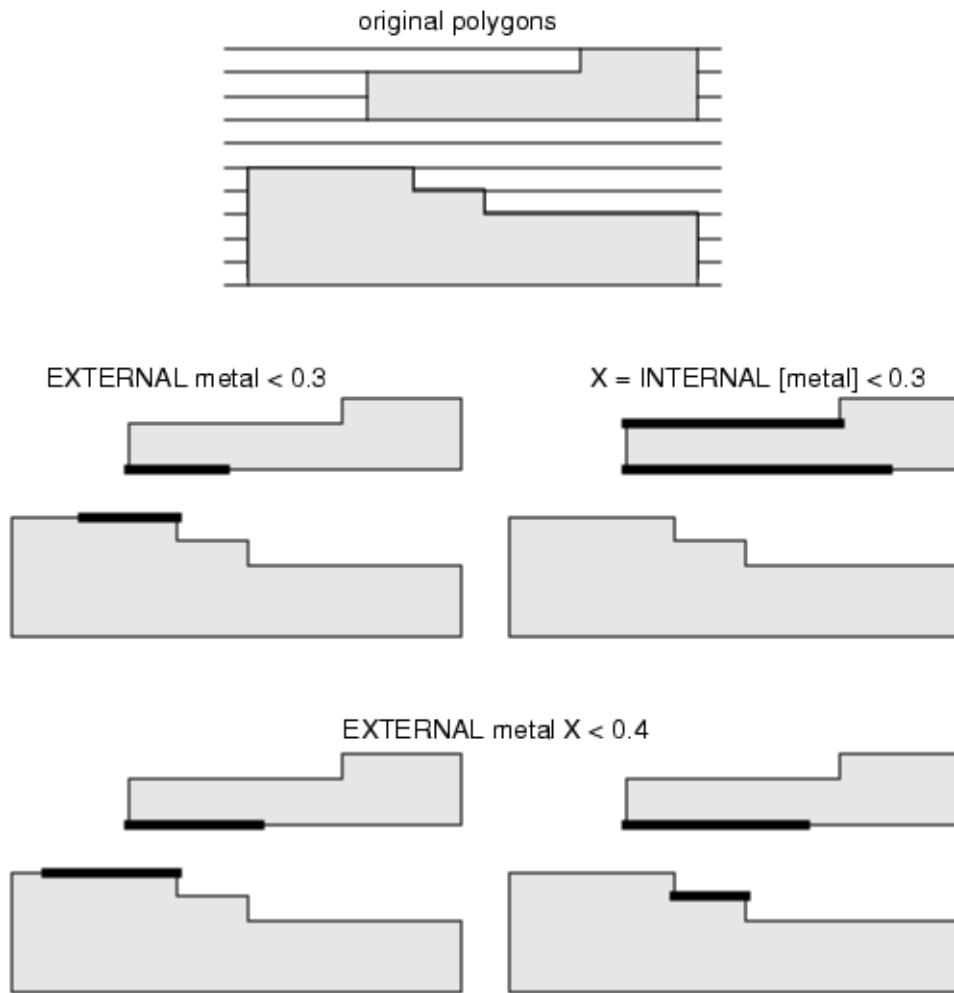
For example, consider this rule check statement (see [Figure 6-17](#)):

```
// Metal spacing must be 0.3 microns except where metal width is
// less than 0.3 microns; in this case, metal spacing must be 0.4 microns.

metal_spacing {
    EXTERNAL metal < 0.3
    x = INTERNAL [metal] < 0.3
    EXTERNAL metal x < 0.4
}
```

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

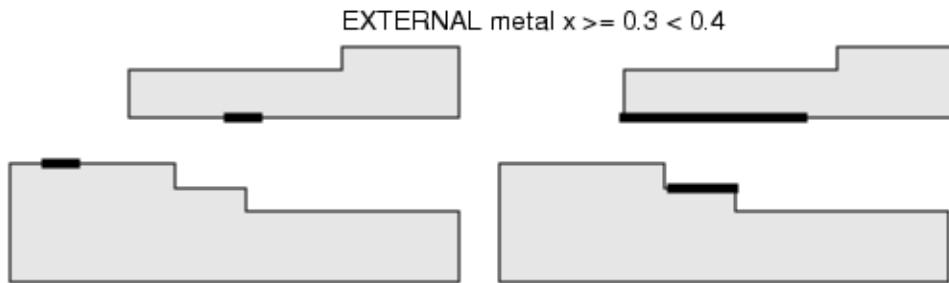
Figure 6-17. Suppressing Redundant Errors (part 1)



One way to suppress this duplication is to use an interval constraint in the second EXTernal operation. This prevents generating two errors for the spacing violations of 0.3 microns. For example, consider this rule check statement (refer to [Figure 6-17](#) and [Figure 6-18](#)):

```
// Metal spacing must be 0.3 microns except where metal width is
// less than 0.3 microns; in this case, the metal spacing must be
// 0.4 microns.
metal_spacing {
    EXTERNAL metal < 0.3
    x = INTERNAL [metal] < 0.3
    EXTERNAL metal x >= 0.3 < 0.4 // uses an interval constraint
}
```

Figure 6-18. 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 6-18](#), 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 left one in [Figure 6-18](#), 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 Calibre nmDRC applications. However, when you specify interval constraints in hierarchical applications without the OPPOSITE keyword, Calibre nmDRC-H may use a large amount of CPU overhead to process skew edges.

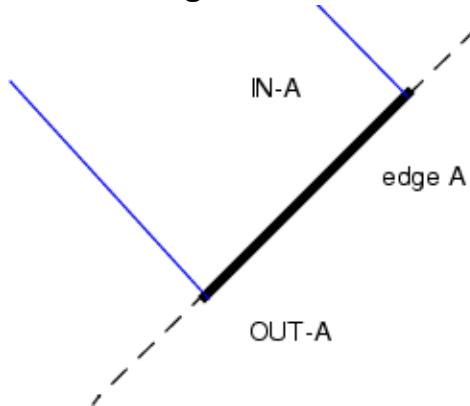
Appropriateness Criteria

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 6-19 illustrates how to make the notion of appropriateness for measurement more precise. Given edge A, region IN-A is the half-plane consisting of all points on the same side of the line as the inside of A. Region OUT-A is the half-plane consisting of all points on the same side of the line as the outside of A. Neither IN-A nor OUT-A contains the line determined by edge A.

Figure 6-19. Edge Inside and Outside Planes



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

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

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.

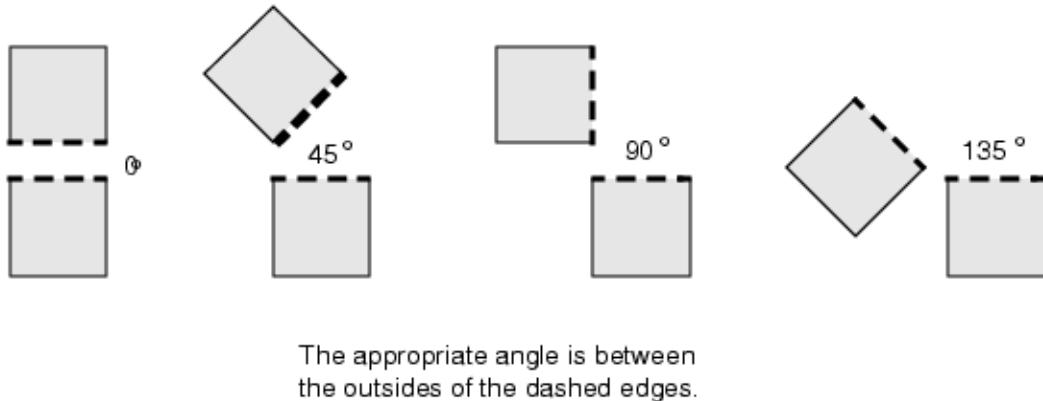
From the definition of appropriateness, the dimensional check operations consider two edges to face each other only if the angle between the corresponding sides of the edges is less than 180 degrees.

The notion of facing edges is maintained for derived edge layers. Edges have inside-facing sides and outside-facing sides (referenced back to their original polygons). These sides of edges are maintained by the Calibre database, and these sides determine appropriate measurements for dimensional check operations, as well as other operations for which the concepts of inside and outside are important.

[Figure 6-20](#) 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, which include the ACUTE, PARALLEL, PERPENDICULAR, and OBTUSE families of secondary keywords are discussed in the *SVRF Manual*.

Figure 6-20. Appropriate Angles Between the Outsides of Edges



Edge Shielding

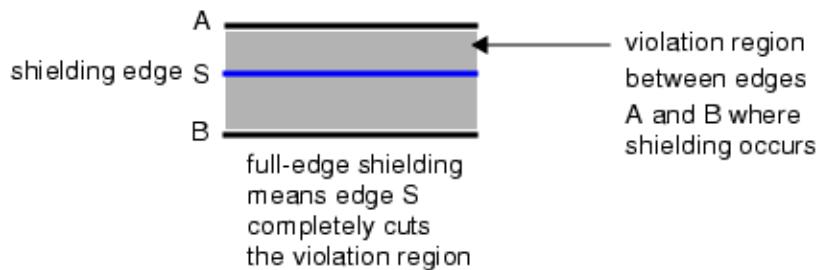
When two edges A and B are measured by dimensional check operations, output from the measurement process can be suppressed to various extents due to the presence of shielding edges. A *shielding edge* is an edge S that completely or partially blocks the line-of-sight between edges A and B.

Edge shielding is controlled in a SPACE rule by using this keyword set:

EXCLUDE SHIELDED [0 | 1 | 2 | 3 | 4]

[Figure 6-21](#) shows how shielding occurs.

Figure 6-21. Full-edge Shielding



The integers 0, 1, 2, 3, or 4 may be specified as the *level*, depending upon the level of shielding required. Integers greater than 4 are considered equivalent to 4, which is also the default if EXCLUDE SHIELDED is specified without an integer.

Level 0 means that no shielding is performed, which is less than what the tool does by default. Level 0 is used if MEASURE ALL is specified alone. If EXCLUDE SHIELDED is also specified, then the EXCLUDE SHIELDED setting is used.

Levels 1, 2, and 3 all perform full-edge shielding. *Full-edge shielding* suppresses output from the measurement of edges A and B by the presence of a shielding edge S, which completely cuts the region defining the violation (in the sense of that produced by the REGION keyword of the EXTERNAL operation) into two distinct areas. The only distinction between levels 1, 2, and 3 is the amount of processing time used to locate the shielding edges.

Level 1 requires a shielding edge S to intersect an endpoint of edge A or B. Refer to the section “[False Measurement Reduction](#)” for a discussion of this type of shielding. Level 1 is the default (EXCLUDE SHIELDED and MEASURE ALL are unspecified).

Level 2 is primarily for Siemens EDA research and development use and is not discussed further.

Note

 Level 2 is not recommended for general use.

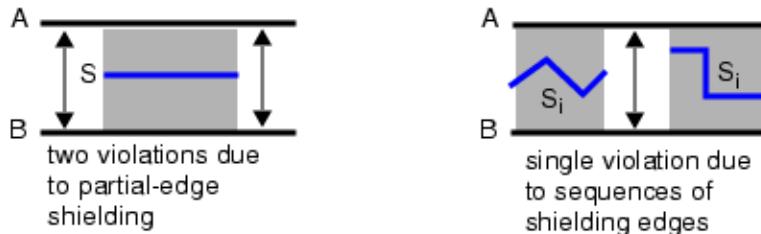
Level 3 guarantees that any shielding edge S is located, but this comes at the cost of substantially increased runtime.

Level 4, like level 3, guarantees the location of any shielding edge S, and it performs *partial-edge shielding* (sometimes referred to as *shadowing*). Partial-edge shielding does not require S to cut the violation region between edges A and B completely (but if it does, then full-edge shielding applies). Rather, only the actual portions of the violations between A having their line-of-sight blocked by S are suppressed.

Partial-edge shielding can produce multiple violations between edges A and B where no shielding would produce one violation. Partial-edge shielding also completely suppresses the

violation between edge A and B if a sequence of shielding edges S_i collectively cut the violation region. Partial-edge shielding is illustrated in [Figure 6-22](#).

Figure 6-22. Partial-Edge Shielding



EXCLUDE SHIELDED is ignored in any of the following cases:

- A and B intersect.
- The operation is a two-layer operation with a closed interval measurement constraint, and the default (Euclidean) or SQUARE metric is specified.
- The operation is a two-layer operation with NOT PROJECTING specified.

For performance reasons, the OPPOSITE metric is highly recommended when using levels 3 and 4.

When **EXCLUDE SHIELDED** is specified with a value greater than 2, processing is applied similar to the **EXCLUDE FALSE** keyword for the dimensional check operations. This comes with a performance cost.

See also the [Enclosure](#), [External](#), [Internal](#), and [TDDRC](#) specification statements in the *SVRF Manual*.

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.

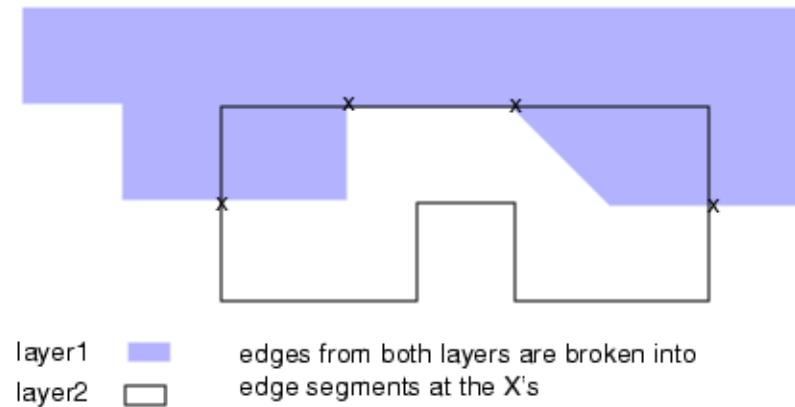
Tip  You should use ABUT < 90 SINGULAR in your dimensional rule checks unless you have good reasons not to.

Edge Breaking

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

Figure 6-23 shows an example of edge breaking.

Figure 6-23. Edge Breaking in a Two-Layer Dimensional Check Operation



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

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

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

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

- Any layer2 edge that intersects a layer1 edge at a single point (excluding the endpoint of the layer2 edge) breaks into two edge segments at the point where the layer2 edge intersects the layer1 edge.

- Any layer2 edge that is coincident with a layer1 edge breaks into two or more edge segments at the point where the layer1 and layer2 edges are no longer coincident.

After edge breaking, Calibre 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.

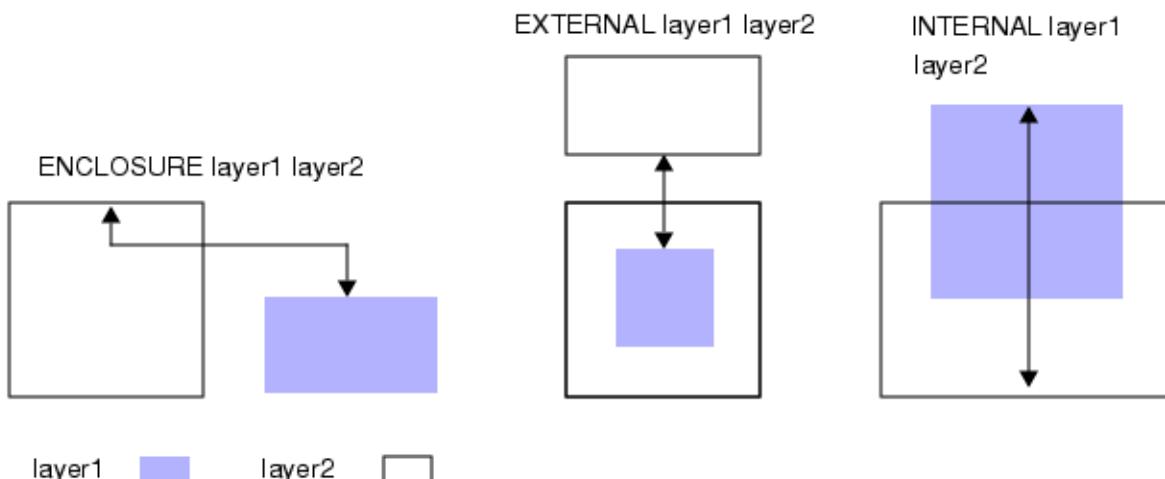
Polygon Containment Criteria

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 6-24](#). The edges shown are not measured by default.

Figure 6-24. Measurements Prevented by Polygon Containment



The polygon containment criteria for the two-layer EXternal check operation allow measured edges to be coincident inside with edges from the other input layer (but not coincident outside). In [Figure 6-24](#), 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 allows 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 Calibre nmDRC measures edge A only if it is not coincident with any edge from layer2. If layer1 is a derived edge layer, then Calibre nmDRC 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.

If MEASURE COINCIDENT is specified in the ENCclosure check, then edge A may additionally be outside-coincident. Coincident edges are measured unless layer2 is a derived polygon layer and A is outside-coincident with an edge from layer2 from a different polygon as edge B.

If MEASURE COINCIDENT is specified in the INTernal check, then edge A or B may additionally be inside-coincident. Coincident edges are measured unless layer2 is a derived polygon layer and edge A is inside-coincident with an edge from layer2 from a different polygon as edge B, or unless layer1 is a derived polygon layer and edge B is inside-coincident with an edge from layer 1 from a different polygon as edge A.

Output Modes

The dimensional check operations function in three primary output modes: error-directed, edge-directed, and polygon-directed.

- **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 < 0.1 ABUT < 90 SINGULAR
```

Output from this type of operation to a derived layer (that is, in a layer definition) cannot be passed to other layer operations with the following exceptions: [DFM Analyze](#), [DFM Property](#), and [DFM RDB](#).

- **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 = ENC [cont] metall < 0.1
```

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

```
x = ENC (cont) metall < 0.1
```

sends edges from layer cont that 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 or REGION EXTENTS keywords. For example, this operation:

```
x = ENC cont metall < 0.1 OPPOSITE REGION
```

creates polygonal regions from the edge segments that satisfy the constraint and sends the regions to layer x. It is usually best to use the OPPOSITE keyword with REGION when deriving layers to be presented to other operations. Otherwise, skew edges can be produced, which have additional processing overhead and can lead to grid snapping issues.

Edge-Directed Output	186
Polygon-Directed Output	186
Skew Edge and Sliver Polygon Handling	189

Edge-Directed Output

There may be times when you need to have error-directed measurement operations result in edge-directed output. The section provides examples of how to create derived edge layers from error-directed measurement operations.

For example the rule:

```
rule {EXT poly diff < 0.09}
```

results in a derived-error layer containing edge pairs from poly and oxide that are closer than 0.09 user units. If you want the output to be a derived edge layer containing only the edge segments pertaining to poly, you enclose the layer name in square brackets ([]). Enclosing the layer name in [] is called positive edge-directed output. For example, this operation:

```
x = EXT [poly] diff < 0.09
```

creates a derived layer of poly edges that satisfy the constraint.

Enclosing the layer name in parentheses is called negative edge-directed output. Negative edge-directed output returns the edge segments that are not normally returned. For example, this operation:

```
x = EXT (poly) diff < 0.09
```

creates a derived layer of poly edges that do not satisfy the constraint.

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, DFM Measure, and DFM Space operations.

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

```
rule { ENC contact [metal1] < 0.005 }
```

has positive edge-directed results from metal1 sent to the Calibre nmDRC results database (as opposed to creating a derived edge layer). The “[Rule File Examples](#)” chapter of the *SVRF Manual* has numerous examples to study.

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, this rule:

```
rule {EXT poly diff < 0.09 ABUT < 90 SINGULAR}
```

would result in a derived error layer containing edge clusters from poly and diff that are closer than 0.09 user units. If you want the output to be a derived polygon layer containing a region between the poly-to-diffusion violations, you include the OPPOSITE REGION keyword set. For example, this operation:

```
x = EXT poly diff < 0.09 ABUT < 90 OPPOSITE REGION
```

creates a derived polygon layer of regions between poly and diff violations.

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 output from the measurement process (see “[Point-to-Point Measurement Output](#)” on page 174). Since the polygonal projection would be a zero-area polygon which would be merged away, Calibre nmDRC 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:
 - Two *true* trivial edges are the result of the measurement process.
 - The OPPOSITE or OPPOSITE SYMMETRIC metric was not specified.
 - The operation’s constraint is of the form “< *a*”.

Although primarily intended to construct intermediate layers in conjunctive rule checks, you may want to use the polygon-directed form of a dimensional check operation for DRC results database instantiation. 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.

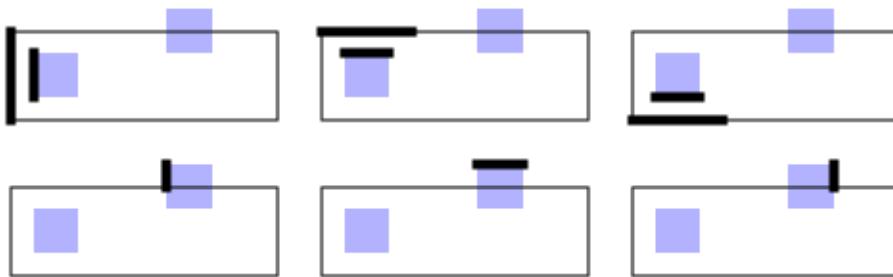
To illustrate the first two points, consider the following check, shown in both error-directed form and polygon-directed form:

```
Rule5.3 { ENCLOSURE cont met < 0.002 ABUT == 0 SINGULAR OUTSIDE ALSO }
Rule5.3.1 { ENCLOSURE cont met < 0.002 ABUT == 0 SINGULAR OUTSIDE ALSO
            REGION }
```

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

Figure 6-25. 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*.

Skew Edge and Sliver Polygon Handling

When deriving layers using the REGION keyword and the Euclidean (default) measurement metric, large numbers of skew edges can be generated. Although Calibre has internal optimizations to handle such layers, it is better to avoid producing such layers due to the performance impact of managing them. This is most commonly done with the OPPOSITE REGION keyword set, which causes the output polygons to have edges that are orthogonal to the database axes.

Skew output edges resulting from the REGION keyword require special handling. The final placement of such edges is the result of merging and numeric rounding. This can result in the placement of skew edge vertices that differ up to 1 dbu from the layout geometry.

In some cases, REGION output can generate narrow polygons (often called slivers) with edges that are extremely close together. These sliver edges could be so close together that they would ordinarily disappear because of grid snapping. If the sliver edges are Manhattan edges (both vertical or both horizontal) and non-projecting, the sliver polygons are oversized by a small amount to ensure they are preserved for output after grid snapping. Skew edge output is not oversized. The result is that there can be some differences in result counts between REGION and edge pairs. Using a higher precision can typically resolve the issue to show the region results.

False Measurement Reduction

There are cases where output from DRC measurements is undesirable, or false, due to the lack of endpoint blocking edges at the appropriate level of the hierarchy between two measured edges.

In Calibre nmDRC-H, it is possible (in rare cases) that the blocking edges are not available at the correct level of hierarchy when edges are measured. This can allow the false measurements to be made. This phenomenon is most often observed with notch measurements using the single-layer EXternal operation.

The following figures show examples of false notch measurement in the context of EXternal:

Figure 6-26. False Notch Measurement (single polygon)

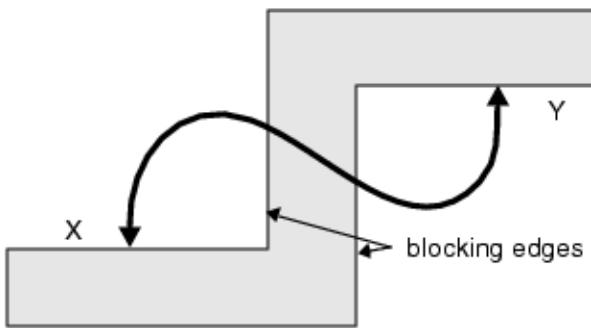
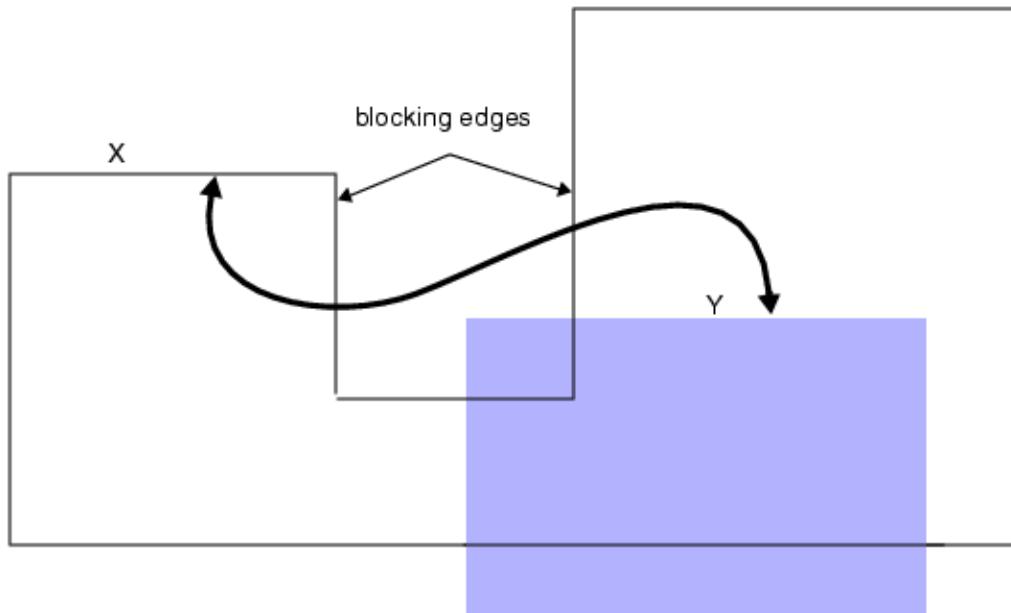


Figure 6-27. False Notch Measurement (two polygons)



In both cases, the measurement is considered false because of another edge completely blocking the line of sight between the two edges being measured. This can occur in certain polygons that traverse design hierarchy.

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

- You do not specify the OPPOSITE, OPPOSITE1, OPPOSITE2, OPPOSITE SYMMETRIC, or OPPOSITE FSYMMETRIC metrics.
- The edges do not intersect.
- The edges do not project onto each other while both are either horizontal or vertical.
- The MEASURE ALL keyword is not specified.
- The region defining a possible violation (in the sense of that produced by the REGION keyword) is cut completely into two pieces by a blocking edge intersecting an endpoint of X or Y.

Any of the blocking edges 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. This situation is rare, and even less common in ENClosure and INTernal operations.

Because eliminating false measurement in an edge-based system is time-consuming, Calibre nmDRC-H attempts to remove only the worst occurrences from consideration by default.

You can instruct Calibre nmDRC-H to expend maximum effort to avoid false notch measurements by using the EXCLUDE FALSE secondary keyword of the dimensional check operations. There is a global statement called DRC Exclude False. Using the YES keyword for that statement enables global false measurement reduction but is strongly discouraged in a production rule file.

Caution

 False notch detection has an adverse effect upon performance. Test it thoroughly before implementing in a production rule file.

Measurement Tolerances

Tolerances for dimensional check operations are controlled globally using the DRC Tolerance Factor specification statement. Tolerances can also be built into measurement constraints of individual layer operations.

Setting of measurement tolerances is desirable when dealing with angled or skew edge data. Vertex coordinate points are subject to grid snapping with such data. Grid snapping behavior can vary due to differences in software compilation, operating systems, and computational hardware. The floating-point representations of numbers, such as irrational numbers, can also differ across platforms due to such factors. Setting of measurement tolerances can be helpful in mitigating issues related to grid snapping and floating-point representations of numbers.

Related Topics

[DRC Tolerance Factor \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Chapter 7

Specialized Calibre nmDRC Applications

Calibre nmDRC has a number of features that are tailored to specialized requirements.

Mask DRC Results	194
Original Shape Flagging	199
Original Geometry Snapping	200
Soft Connection Checks	201
Disk-Based Layers	203

Mask DRC Results

The DRC Check Map specification statement provides a way to output layers to mask data DRC results databases. Among its many configuration options, it allows an $m \rightarrow n$ mapping between DRC rule check outputs 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 $m \rightarrow n$ mapping is of primary interest. For any given Calibre nmDRC run, a set of unique DRC rule checks $\{R_1, \dots, R_n\}$ ($n > 0$) 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 $\{D_1, \dots, D_m\}$ ($m > 0$) are generated from the Calibre nmDRC run such that the following hold:

- Any individual DRC rule check R_j may have output directed to any number of different DRC results databases in the set $\{D_1, \dots, D_m\}$.
- Any individual DRC results database D_j may contain output from any number of different DRC rule checks in the set $\{R_1, \dots, R_n\}$.
- The set $\{D_1, \dots, D_m\}$ does not have to have uniform type, that is, ASCII, GDSII, or OASIS.

The structure of the mask results database is subject to the set of rule checks that are actually used for the run. See “[Rule Check Selection and Mask Results Databases](#)” on page 116 for details.

Attribute Specification	194
Mapping Algorithm for Output	195
AREF Output	198
AUTOREF Output	199

Attribute Specification

The DRC Check Map specification statement allows per-rule-check specification of a DRC results database and its associated type (ASCII, GDSII, or OASIS), as well as the hierarchy form, and any cell prefixes or suffixes. 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.
- The prefix string for cell names in GDSII or OASIS DRC results databases, as specified by the PREFIX *string* 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 APPEND *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.
- Whether OASIS output is strict-mode and whether it includes CBLOCK records. These are controlled by the STRICT and CBLOCK keywords of the DRC Results Database statement.

Details of DRC results database structures are discussed under “[GDSII DRC Results Database Format](#)” on page 242 and “[OASIS DRC Results Database Format](#)” on page 243. Management of results counts is discussed in the section “[Result Count Limits](#)” on page 245.

Related Topics

[DRC Results Database \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Mapping Algorithm for Output

Given the set of unique DRC rule checks $\{R_1, \dots, R_n\}$ to be executed in a DRC run, a set of unique DRC result databases $\{D_1, \dots, D_m\}$ is determined which encompasses the cumulative

DRC output from the application. Each results database D_j has an associated (and unique) type (ASCII, GDSII, or OASIS). The mapping is n->m. Any given DRC rule check R_i may map to multiple members of $\{D_1, \dots, D_m\}$ and any given DRC results database D_j may be mapped into by multiple members of $\{R_1, \dots, R_n\}$.

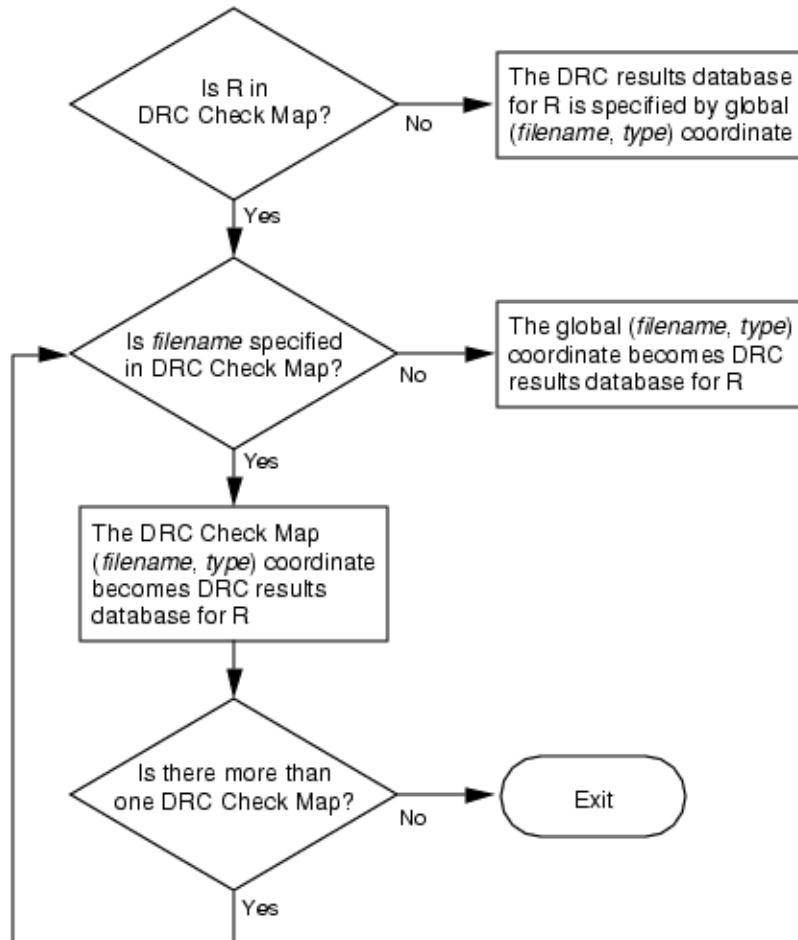
In order to allow completely unambiguous and non-conflicting construction of the previous mapping, the compiler enforces strict rules on the DRC Check Map specification statement:

- 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, 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, 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 filename.
- 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).
- For any two DRC Check Map statements writing to the same database, the PREFIX or APPEND specifications must be consistent when used. Similarly, a DRC Check Map statement that writes to the same database as the DRC Results Database statement must have consistent PREFIX or APPEND specifications when used.

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 $R_i \rightarrow D_j$ map. A conceptual pseudocode of this is shown in

[Figure 7-1](#).

Figure 7-1. Mapping Algorithm for Multiplexing of DRC Rule Check Output



AREF Output

In hierarchical Calibre applications, AREF structures are specified from arrayed rectangles in a GDSII-type DRC results database. AREF output is specified by the optional AREF parameters in the DRC Check Map specification statement and is discussed in detail in the *SVRF Manual*. AREF structures form orthogonal arrays. By default, 16 objects are placed into an AREF structure because this offers a good balance between compacting the database and performance.

The DRC Check Map SUBSTITUTE keyword allows an arbitrary polygon to be substituted for the specified rectangle in the output GDSII structure. The polygon must have at least two coordinates, and be simple and orientable. A two-point polygon is interpreted as an orthogonal rectangle. The coordinates are in user units.

OASIS databases have their own method of array compaction that happens automatically. In general, on OASIS database is the most compact form you can choose.

AREF Structures and Fill Cells 198

AREF Structures and Fill Cells

Fill cells for AREF structures can be implemented in one of two ways.

Assume you have the following in your rule file:

```
LAYOUT PATH main_design.gds fill_cell.gds
LAYOUT SYSTEM GDSII
LAYOUT PRIMARY TOP
DRC RESULTS DATABASE output.gds GDSII
fill { COPY fill }
DRC CHECK MAP ... AREF cell_name
```

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* design has one cell in its database that is not placed in the main design:

```
cell_name
```

After you run Calibre nmDRC-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
fill { COPY fill }
DRC CHECK MAP fill ... AREF cell_name
```

The file `new_output.gds` contains the AREF structure in a cell of the name `cell_name`.

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

Related Topics

[DRC Check Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

AUTOREF Output

Another option for outputting more compact GDSII mask results databases is to use the AUTOREF keyword for DRC Check Map. This option is highly efficient in compacting results that have large arrays of rectangles, such as from fill operations. This option is easier to use than the AREF option, but offers less customizable control.

Original Shape Flagging

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).

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.

The [Offgrid](#) layer operation offers many advanced features for pitch checking. It also allows you to specify the type of results output: error directed (the default), edge-directed, or polygon-directed.

Original Geometry Snapping

Original geometry snapping aligns vertices of original shapes on specified grids. 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.

The Layer Resolution statements (which establish the user grid) only apply if the layers they reference are actually read in during the DRC run. Calibre only reads in the layers from the layout database that are actually required by the run.

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 Calibre nmDRC-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.

For information about how floating-point arithmetic can affect grid snapping and how to mitigate related measurement issues, see “[Constraints Checking for Equality](#)” in the *SVRF Manual*.

Related Topics

- [DRC Map Text \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
- [Layer Resolution \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
- [Resolution \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
- [Snap Offgrid \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Soft Connection Checks

Calibre DRC applications can check for soft connections with the External operation. Calibre nmLVS checks for soft connections with the LVS Softchk specification statement, which you cannot use with DRC applications. The INSIDE ALSO and NOT CONNECTED secondary keywords of the External operation are useful in soft connection DRC checks.

The following examples show a pair of statements LVS applications use to check for soft connections followed by the associated DRC rule.

Example 1

```
SCONNECT upper_layer lower_layer
LVS SOFTCHK lower_layer LOWER

"SOFTCHK LOWER NO CONTACT" {
    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 UPPER

"SOFTCHK UPPER NO CONTACT" {
    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 UPPER ALL

"SOFTCHK UPPER ALL" {
    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

"SOFTCHK LOWER WITH ABUT" {
    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 BY contact_layer
LVS SOFTCHK lower_layer LOWER

"SOFTCHK LOWER" {
    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 6

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer CONTACT

"SOFTCHK CONTACT" {
    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 7

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer CONTACT ALL

"SOFTCHK CONTACT ALL" {
    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 8

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer UPPER

"SOFTCHK UPPER" {
    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 9

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer UPPER ALL

"SOFTCHK UPPER ALL" {
    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
    conf_contact = used_contact NOT OUTSIDE conf_lower
    upper_layer NOT OUTSIDE conf_contact
}
```

Related Topics

[External \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[LVS Softchk \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Disk-Based Layers

By default, geometric data on layers (original and derived) created during a DRC run is memory-based. 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).

Note

 Disk-based layers should only be used if absolutely required. Accessing data from disk carries a considerable performance penalty.

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. When execution is complete, all directories that were created are removed.

The amount of data 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.

For MTflex runs, the Layer Directory statement in the rule file applies only to the primary machine. Layer directories may also be specified for the remote machines by using the REMOTE HOST ... LAYERDIR option in a Calibre MTflex configuration file. Specifying

layer directories on remote machines for a Hyperscaling run may result in severe performance penalties.

Related Topics

[Multithreaded Modes of Operation](#)

[Layer Directory \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[REMOTE HOST \[Calibre Administrator's Guide\]](#)

Chapter 8

Calibre nmDRC Results

DRC applications output a transcript, a results database, and a summary report (if specified). The run transcript has features that are common to all batch Calibre tools that read geometric databases.

Session Transcripts	206
Header Information.....	206
Rule File Compilation.....	206
Layout Data Input.....	208
Results Database Initialization Section.....	211
Executive Process	213
Transcript Features Based Upon Run Mode.....	221
DRC Results Database.....	234
ASCII DRC Results Database Format	236
GDSII DRC Results Database Format	242
OASIS DRC Results Database Format.....	243
Result Count Limits	245
Hierarchical Calibre nmDRC Results Database.....	246
DRC Summary Report	247

Session Transcripts

Calibre applications produce a transcript showing statistics for the different runtime modules.

For details on the information found in the Post-Tapeout transcript, refer to “[Transcripts for Post-Tapeout Operations](#)” in the *Calibre Post-Tapeout Flow User’s Manual*.

Header Information	206
Rule File Compilation	206
Layout Data Input	208
Results Database Initialization Section	211
Executive Process	213
Transcript Features Based Upon Run Mode	221

Header Information

The transcript header provides information about the Calibre version, the host, Calibre executable, and the job.

- The first line shows the Calibre version used for the run:

```
// Calibre <version> <time stamp>
```

- These lines show the operating system, hostname, OS version, and build:

```
// Mentor Graphics software executing under <architecture>
// Running on <OS> <hostname> <version> <build>.
```

- The path of executable, the command line options you use, and the PID are shown on these lines:

```
// Running <version>/pkgs/icv/pvt/calibre -drc -hier rules
// Process ID: 29097
```

- The starting time of the run and the number of CPUs used for making runtime calculations (those for which licenses are required) are shown on these lines:

```
// Starting time: <time stamp>
// Running on 1 CPU (pending licensing)
```

Elapsed times are measured from the starting time.

Rule File Compilation

The “STANDARD VERIFICATION RULE FILE COMPILATION MODULE” section of the transcript includes information about the rule file and the status of the Calibre environment.

- The pathname of the rule file is given by this line:

```
--- RULE FILE = home/rule_files/rules
```

- The rule file is echoed to the transcript. If the rule file is in compile-time Tcl Verification Format (TVF), the TVF code is not expanded to SVRF by default.

Any compilation errors encountered are then shown. Refer to the chapter “Error Messages” in the *Standard Verification Rule Format (SVRF) Manual* for a description of compilation error messages.

- The amount of CPU and real time required for compilation are reported. If compilation is successful, you will see a line like this:

```
--- STANDARD VERIFICATION RULE FILE COMPILATION MODULE COMPLETED.  
CPU TIME = 0  REAL TIME = 0 LVHEAP = 1/3/4
```

The time and LVHEAP statistics are defined under “[Layout Data Input](#).”

- Any CALIBRE_ environment variables are then listed, along with license acquisition information and the tool that is running.

```
--- CALIBRE_* ENVIRONMENT VARIABLES:
```

```
<variable list>
```

```
-----  
-----  
-----          CALIBRE:::DRC-H - LICENSING MODULE  
-----  
-----  
-----  
// Applying licensing policy...  
// <feature> license acquired.  
// <feature> license acquired.  
...  
// Licensed Products  
// -----  
// Base products running on <n> cores:  
//   - DRC (Hierarchical)
```

Layout Data Input

The “CALIBRE LAYOUT DATA INPUT MODULE” section of the transcript includes information about the layout system, magnification, and layout statistics. The discussion in this section assumes hierarchical run data.

The layout data information is reported in the following subsections.

- Layout System File Summary Information

Example:

```
-----  
-----          OASIS FILE SUMMARY INFORMATION      -----  
-----  
OASIS FILENAME:      test.oas  
OASIS VERSION:       1.0  
DATABASE PRECISION: 1000  
MAGNIFICATION:      1
```

- Layout Input Data for Individual Cells

This section shows these statistics for all cells in the database: cell name, number of placements, number of array records, number of polygon records, number of path records, and number of text records. The statistics for the read-in of the layout appear in a table with a header similar to this:

CELL NAME	PLACEMENTS	ARRAYS	POLYGONS	PATHS	TEXTS
-----------	------------	--------	----------	-------	-------

Other information such as the layers from which data not used by the run originates, or empty layer information, is shown when applicable.

Following the preceding information you will see a line like this:

```
--- LAYOUT DATABASE CONSTRUCTOR COMPLETED.   CPU TIME = 3  
      REAL TIME = 4   LVHEAP = 3/4/4
```

CPU TIME is the total time in seconds the CPUs took to complete the module. For reporting purposes, times are rounded to the nearest integer.

REAL TIME is the clock time in seconds that elapsed to complete the module. For reporting purposes, times are rounded to the nearest integer.

LVHEAP is memory data in megabytes. This is described under “[LVHEAP Statistics](#).”

Next, the various algorithms used for constructing the Calibre internal database are shown, along with cell names and other data that the algorithm affects.

- Text Objects for Connectivity Extraction

This section lists text objects that are used for connectivity purposes. These objects appear on layers specified in [Layout Text](#) and [Text](#) statements. The [Text Print Maximum](#) statement can be used to control how many objects are reported here.

- Text Objects for With Text Operations

This section lists text objects that appear on layers specified in [Not] [With Text](#) operations.

- Text Objects for Expand Text Operations

This section lists text objects that participate in [Expand Text](#) operations.

- Text Objects for CAPI Operations

- Ports (Calibre nmLVS /Calibre nmLVS-H only)

This section lists the ports for the listed tools.

- Layer Read Summary (Simple Layer Geometries)

This section shows the simple layer numbers and the number of geometries read in on these layers. For example:

SIMPLE LAYER	GEOMETRIES
4	34669
5	65853
6	13525
35	2393
36	3732

Note

 Calibre tools read in *only the layers that are required for the run*, which may not be all of the layers in the layout database. The layers Calibre requires for a run are those needed to produce output from rule checks or for circuit extraction.

- Layer Read Summary (Original Layer Geometries)

This section shows statistics for the Calibre layers that are declared in Layer statements in the rule file and that are required for the run. The initial geometries are the counts of geometries before Calibre processes them. The hierarchical count is given first, followed by the flat count in parentheses. The final geometries are the counts of geometries after Calibre has processed the input data and constructed its own database representation of that data. For example:

ORIGINAL LAYER	INITIAL GEOMETRIES	FINAL GEOMETRIES
M1METAL	17257 (27524)	18493 (27524)
CONTACT	68246 (132621)	69434 (132621)
POLY_LAYOUT	34669 (66488)	43862 (66488)

- Layer Read Summary (Text for Connectivity Extraction)

This section lists the simple layers and text objects needed for connectivity extraction, such as are specified using [Text](#) and [Text Layer](#) statements.

- Layer Read Summary (Text for With Text Operations)

This section lists the simple layers and text objects needed for [[Not](#)] [With Text](#) layer operations.

- Layer Read Summary (Text for Expand Text Operations)

This section lists the simple layers and text objects needed for [Expand Text](#) operations.

- Layer Read Summary (Text for CAPI Operations)

- Cell and Placement Summary (only in hierarchical applications)

This section summarizes various types of cells according to how Calibre views them. User cells are cells in the original design. Pseudocells are cells that Calibre creates for hierarchy management. The numbers of hierarchical and flat placements are shown.

- Layout Data Input Module Summary

This section reports a summary of the module. Items of interest that may not be self-explanatory are these:

Simple layers — Drawn layers comprised of a single layer number.

Original layers — Drawn layers comprised of simple layers or sets of simple layers.

Database Extent — The x and y coordinates of the lower-left and upper-right corners of the layers actually needed for the Calibre run. This is not necessarily the extent of the original layout database because Calibre reads in only the layers actually required for the run.

Text Depth for Connectivity Extraction — The setting of the [Text Depth](#) statement in the rule file, or its default. This is the hierarchical depth from which connectivity extraction text objects are read.

Geometry Flagging — The Flag family of SVRF statements enabled for the run.

Excluded Cells — Cells that have been excluded by [Exclude Cell](#) statements in the rule file.

Layout Base Layer — The device-level layers defined by the [Layout Base Layer](#) statement in the rule file. A value of “NOT SPECIFIED” indicates this statement is not defined in the rule file.

Layout Top Layer — The top-level layers defined by the [Layout Top Layer](#) statement in the rule file. A value of “NOT SPECIFIED” indicates this statement is not defined in the rule file.

Limiting Text Object Output	211
Hierarchical and Flat Counts	211

Limiting Text Object 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

Both the transcript and the summary report file (if specified) provide many statistics independent of the derived layer statistics. In hierarchical applications, these statistics appear as a pair of numbers, with the second number 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, which are subject to various transformations when placed in the design. This can differ from the actual flat count of objects.

Results Database Initialization Section

The Results Database Initialization section of the transcript shows the rule file settings for results database settings for Calibre nmDRC applications.

The settings and their explanations are as follows:

GLOBAL DRC RESULTS DATABASE FILE	Shows the pathname and type of the DRC Results Database statement.
GLOBAL MAXIMUM RESULTS PER RULECHECK	Shows the setting of the DRC Maximum Results statement.
GLOBAL MAXIMUM VERTICES PER RESULT POLYGON	Shows the setting of the DRC Maximum Vertex statement.
CHECK TEXT MAPPING	Shows the setting of the DRC Check Text statement.
KEEP EMPTY RULE CHECKS	Shows the setting of the DRC Keep Empty statement.
DRC RESULTS MAGNIFICATION	Shows the setting of the DRC Magnify Results statement.

DRC RESULTS DATABASE PRECISION

Shows the setting of the [DRC Results Database Precision](#) statement.

DRC RULECHECK -> RESULTS DATABASE MAPPING

Shows the settings of [DRC Results Database](#) and [DRC Check Map](#) statements in the rule file.

Executive Process

The section entitled “CALIBRE::<tool> EXECUTIVE MODULE” reports layer derivations, rule check execution, connectivity extraction, tool-specific processes, operating parameters, various event logs, warning messages, and summary information. The <tool> varies depending on command line options used for the run.

Time Statistics	213
Layer Statistics in Flat Processing	213
Layer Statistics in Hierarchical Processing.....	215
LVHEAP Statistics	217
Run Summary	217

Time Statistics

Time stamps appear throughout the transcript. Some of the time stamps are for individual modules, some are for the generation of layers, and some are for other purposes. Time of generation for layers may differ across various executive modules.

The definitions of the time statistics are the same everywhere in the transcript:

- **CPU TIME** — The amount of time in seconds the CPUs took to complete the module. For reporting purposes, times are rounded to the nearest integer.
- **REAL TIME** — The clock time in seconds that elapsed to complete the module. For reporting purposes, times are rounded to the nearest integer. CPU time can be greater than REAL TIME if there is more than one CPU involved in an operation.
- **ELAPSED TIME** — The clock time in seconds that elapsed since the Starting time in the header of the transcript. ELAPSED TIME takes into account license acquisition and the reading and writing of data across a network.

To print the current timestamp for each generated layer, set the CALIBRE_PRINT_TIME_STAMPS environment variable to any value (including nil). The current timestamp is output in parentheses following the ELAPSED TIME as shown in the following example:

```
CPU TIME = 2  REAL TIME = 0  LVHEAP = 22/69/69  OPS COMPLETE = 191 OF 193
ELAPSED TIME = 52 (2013/05/01 12:32:23)
```

Layer Statistics in Flat Processing

Certain transcript statistics are relevant to flat processing of layers.

Example layer statistics:

```
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
ELAPSED TIME = 987
```

The operation has generated layer nplus. It required 140 CPU seconds and 142 real time seconds. It was the 34th layer operation completed out of a total flow requiring 589 layer operations. The elapsed time since the beginning of the run is 987 seconds. 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** — These 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
ELAPSED TIME = 146
```

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 expanding 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.

Note that multiple operations may appear in a single block. This indicates concurrency of layer operations. 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
ELAPSED TIME = 787
```

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

Layer Statistics in Hierarchical Processing

Certain statistics in the run transcript are relevant to hierarchical processing of layers.

This is an example output:

```
nplus = sdm NOT ppm
-----
nplus (HIER TYP=1 CFG=1 HGC=4578 FGC=316325 HEC=137654 FEC=37475542 VHC=F
VPC=F)
CPU TIME = 4 REAL TIME = 4 LVHEAP = 27/31/31 OPS COMPLETE = 34 OF 589
ELAPSED TIME = 982
```

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 flat DRC. 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 (TYP 1), this is the total number of edges on the polygons counted by HGC. For derived edge layers (TYP 2), HEC is identical to HGC. For derived error layers (TYP 3), 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 is based 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.

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 and results layer statistics in the transcript, report approximate current memory usage for Calibre applications. These applications run completely in memory when layers are memory-based.

The report of memory usage consists of three numbers, represented in megabytes (2^{20} bytes). For example:

LVHEAP = 28/47/49

The first number is the amount of memory being used at the time of the report. The second number is the total amount of memory allocated by the application. The third number is the maximum amount of memory that has been allocated during the run up to that point of the report.

The third LVHEAP number is typically the largest. The second and third numbers are sometimes equal.

Run Summary

The end of a DRC transcript shows data based upon layer operation categories. This shows performance information that can be useful in rule file optimization.

Depending upon which layer operations and connectivity operations are used during your run, you will see the following type of report:

```
Cumulative ONE-LAYER BOOLEAN Time: CPU = 0 REAL = 0
Cumulative TWO-LAYER BOOLEAN Time: CPU = 1 REAL = 3
Cumulative POLYGON TOPOLOGICAL Time: CPU = 2 REAL = 4
Cumulative POLYGON MEASUREMENT Time: CPU = 0 REAL = 0
Cumulative SIZE Time: CPU = 0 REAL = 0
Cumulative EDGE TOPOLOGICAL Time: CPU = 1 REAL = 2
Cumulative EDGE MEASUREMENT Time: CPU = 0 REAL = 0
Cumulative STAMP Time: CPU = 4 REAL = 6
Cumulative ONE-LAYER DRC Time: CPU = 10 REAL = 11
Cumulative TWO-LAYER DRC Time: CPU = 7 REAL = 8
Cumulative NET AREA (RATIO) Time: CPU = 6 REAL = 8
Cumulative DENSITY Time: CPU = 0 REAL = 0
Cumulative MISCELLANEOUS Time: CPU = 34 REAL = 36
Cumulative CONNECT Time: CPU = 1506 REAL = 1620
Cumulative RDB Time: CPU = 7 REAL = 12
```

Here are the descriptions:

Table 8-1. Cumulative Runtime Statistics

Category	Operations Reported
ONE-LAYER BOOLEAN	AND, OR, XOR
TWO-LAYER BOOLEAN	AND, NOT, OR, XOR
POLYGON TOPOLOGICAL	Cut, Enclose, Inside, Interact, Outside, Touch, including NOT counterparts.
POLYGON MEASUREMENT	Area, Donut, Enclose Rectangle, Perimeter, Holes, Rectangle, Vertex, and any NOT counterparts, where applicable.
SIZE	Size
EDGE TOPOLOGICAL	Operations from the Coincident Edge, Inside Edge, Outside Edge, and Touch Edge families, including NOT counterparts; OR Edge
EDGE MEASUREMENT	Angle, Convex Edge, Expand Edge, Length, Path Length, and any NOT counterparts.
STAMP	Stamp
ONE-LAYER DRC	External, Internal
ONE-LAYER DRC(A)	Enclosure, External, Internal two-layer operations where the input layers are actually aliases for the same layer.
TWO-LAYER DRC	Enclosure, External, Internal
NET AREA (RATIO)	Net Area, Net Area Ratio, Net Area Ratio Accumulate, Net Area Ratio Print
DENSITY	Density

Table 8-1. Cumulative Runtime Statistics (cont.)

Category	Operations Reported
TDDRC	TDDRC
DFM COPY	DFM Copy
MISCELLANEOUS DFM	DFM Text, DFM Stamp
DFM PROPERTY	DFM Property, DFM Property Merge
DFM RDB	DFM RDB
DFM SPACE	DFM Space
MISCELLANEOUS	Copy, Deangle, Drawn family of operations, Expand Text, Extent, Extent Cell, Extent Drawn, Extents, Flatten, Grow, Inside Cell, Magnify, Merge, Net, Net Interact, Offgrid, Ornet, Push, Rectangle Enclosure, Rectangles, Rotate, Shift, Shrink, Snap, With Edge, With Neighbor, With Text, With Width, and any NOT counterparts, where applicable.
CONNECT	Connect, Sconnect
RDB	Results database generation, including those databases created with the RDB keyword.

This line reports the CPU time and real time in seconds for the entire executive module:

```
--- CALIBRE::DRC-H EXECUTIVE MODULE COMPLETED.  CPU TIME = 47
      REAL TIME = 49
```

This line reports the number of rule checks executed during the run. The DRC [Un]Select Check statements affect this number:

```
--- TOTAL RULECHECKS EXECUTED = 7
```

This line reports the total number of results generated from the executed rule checks:

```
--- TOTAL RESULTS GENERATED = 7000 (9006)
```

The first number is the total number of hierarchical results that appear in the results database. This number is controlled by the **DRC** or **ERC** Maximum Results statement. The number in parentheses is the number of flat results. The number in parentheses is greater than or equal to the first number.

This line reports the results database filename and type specified in the DRC Results Database statement:

```
--- DRC RESULTS DATABASE FILE = drc_results (ASCII)
```

These lines report the CPU time and real (clock) time in seconds for the entire run, along with the timestamp of completion for the run:

```
--- CALIBRE::DRC-H COMPLETED - <time stamp>
--- TOTAL CPU TIME = 93    REAL TIME = 110
```

If the run was interrupted by a process outside of Calibre's control, such as by a job queue manager, then this message appears:

```
--- PROCESS WAS STOPPED/CONTINUED VIA JOB CONTROL, COUNT = <n>
```

The <n> is the number of times the calibre process was stopped and re-started.

This line reports the number of CPUs used for performing calculations (that is, processors that count toward license usage) during the run:

```
--- PROCESSOR COUNT = 1
```

This line reports the location of the **DRC** or **ERC** Summary Report file, if specified:

```
--- SUMMARY REPORT FILE = drc_summary
```

Transcript Features Based Upon Run Mode

Calibre multithreaded run modes have specific transcript features. Some of these features are common among the various modes, while others differ.

MT and MTflex Modes With Hyperscaling	221
Unique Transcript Features of MTflex Mode With Hyperscaling	226
MT Mode Without Hyperscaling	231
MTflex Mode Without Hyperscaling.....	232

MT and MTflex Modes With Hyperscaling

Most of the features in this section are present in MT and MTflex transcripts with hyperscaling enabled. When this is not the case, MT mode is specifically used.

Specific features of the Calibre MTflex transcript are described in “[Unique Transcript Features of MTflex Mode With Hyperscaling](#).” If you run in MT or Calibre MTflex mode, you should use hyperscaling (-hyper option) unless you have good reasons not to (see “[Hyperscaling](#)” on page 146).

Header Information

Here is an example transcript of the header information generated from a hyperscaling MT run.

```
// Running home/pkgs/icv/pvt/calibre64 -drc -hier -turbo -hyper rules
//
// Starting time: <time stamp>
//
// HYPERSCALING ENABLED with 4 pseudo HDBs.

// Running on 4 CPUs (pending licensing)
//
// LITHO operations available for use on 4 CPUs (pending licensing)
//
// Initializing MT on HDB 0: 0
// Creating log for pseudo HDB 1 on host, logfile = /path/
CalibreHDBLog.host.cpuid
// Creating log for pseudo HDB 2 on host, logfile = /path/
CalibreHDBLog.host.cpuid
// Creating log for pseudo HDB 3 on host, logfile = /path/
CalibreHDBLog.host.cpuid.
// Creating log for pseudo HDB 4 on host, logfile = /path/
CalibreHDBLog.host.cpuid.
// Initializing MT on pseudo HDB 1
// Initializing MT on pseudo HDB 2
// Initializing MT on pseudo HDB 3
// Initializing MT on pseudo HDB 4
```

This example shows the -hyper command line option has been used and that four pseudo databases are initialized, numbered HDB 1 through 4. HDB 0 is the primary database

maintained by the master processor. These are in-memory databases used to collect data generated by the CPUs used during the run. There are four CPUs used for this particular run.

Memory Allocation

When you run a Calibre job in hyperscaling mode, the transcript includes LVHEAP and SHARED statistics.

For example:

LVHEAP = 4/6/6 SHARED = 1/1

The LVHEAP numbers differ from other run modes due to the requirements of multithreading with hyperscaling. See [LVHEAP Statistics](#) for a description of the LVHEAP numbers.

When Calibre runs an operation, it needs memory for computation and memory to hold the data. When hyperscaling is used, sometimes an operation on one HDB uses the same memory as an operation on a separate HDB. To reduce the overall memory footprint in such cases, data is transferred from the first HDB to the second HDB. When this occurs, the amount of data transferred is what is written into the SHARED component. The SHARED statistic is the amount of memory in megabytes that has been transferred between HDBs for a particular operation. The first number is the shared memory for the current step, the second is the maximum shared memory. This is a subset of the LVHEAP statistics.

In MT mode, the SHARED statistic is not an indicator of any sort of inefficiency.

In MTflex mode, if RDS is not enabled and you are seeing large SHARED numbers, this means you can see improvement by enabling RDS. See “[Remote Data Server](#)” in the *Calibre Administrator’s Guide*.

If a remote data server (RDS) is enabled, -hyper remote is not used, and SHARED numbers other than 0/0 appear, it could mean there is not enough remote memory available. This is not necessarily a bad thing, but it indicates improvement is possible with the addition of remote memory.

If -hyper remote is used with RDS, it is important that the SHARED numbers be 0/0, as memory sharing in this configuration degrades performance.

Constructing Hierarchical Database

The constructing hierarchical database section lists two algorithms that are used only with hyperscaling:

FLATTENING SELECTED VERY SMALL CELL PLACEMENTS
FLATTENING SELECTED TOP LAYER CELL PLACEMENTS

These are internal optimizations needed for a hyperscaling run. These algorithms affect object counts that appear in the run transcripts. Object counts can differ between a hyperscaling run and a non-hyperscaling run. This is an expected behavior.

Injecting Hierarchy

The order of the cell names may differ between a hyperscaling and a non-hyperscaling run. This is due to how the data is partitioned for a hyperscaling run.

Pseudo Hierarchical Database Construction

This portion of the transcript contains information on the construction of pseudo hierarchical databases. For example:

```
PSEUDO HDB 1 CONSTRUCTED.  CPU TIME = 0  REAL TIME = 0  LVHEAP = 1/1/1
PSEUDO HDB 2 CONSTRUCTED.  CPU TIME = 0  REAL TIME = 0  LVHEAP = 1/1/1
PSEUDO HDB 3 CONSTRUCTED.  CPU TIME = 0  REAL TIME = 0  LVHEAP = 1/1/1
PSEUDO HDB 4 CONSTRUCTED.  CPU TIME = 0  REAL TIME = 0  LVHEAP = 1/1/1
```

This information means the pseudo hierarchical databases used by the CPUs are ready to store data.

Layer Read Summary (Original Layer Geometries)

This portion of the transcript may also differ from a non-hyperscaling run. The FINAL GEOMETRIES count could be different because of how the layout database has been partitioned for hyperscaling. The FINAL GEOMETRIES count will likely be higher for a hyperscaling run than for a run without hyperscaling.

For example, this line of the transcript typically shows higher numbers than for a non-hyperscaling run:

LAYERNAME	68246 (132621)	132593 (132621)
-----------	----------------	-----------------

Cell and Placement Summary

This portion of the transcript will likely show some smaller numbers than the corresponding section of a non-hyperscaling run. This is because of the two flattening algorithms mentioned previously in this section.

Layout Data Input Module Summary

This portion of the transcript reflects the Layer Read Summary section data and differs from a non-hyperscaling run.

Calibre Executive Module

The Executive Module shows differences from a non-hyperscaling run in how data is partitioned to the pseudo hierarchical databases and how that data gets reported in the transcript. The examples in this section are from a DRC-H run.

In a hyperscaling MT transcript, you often see reports such as this for layer derivations:

```
CONTACT = OR CONTACT
-----
Operation EXECUTING on HDB 1 (T2 S1 E4:4 H0:0 A15:16 L14:14 C1)
```

This means the operation has been assigned to one of the CPUs and the data is being stored in HDB 1. The information in the parentheses is for use within Siemens EDA research and development. When the tool is finished with this layer, the following is reported in the transcript:

```
Original Layer CONTACT DELETED -- LVHEAP = 1/5/5 SHARED = 0/1
```

Elsewhere in the MT transcript, the statistics of this operation are reported as follows:

```
CONTACT = OR CONTACT
-----
CONTACT (HIER TYP=1 CFG=1 HGC=132964 FGC=132992 HEC=532450 FEC=532562
VHC=F VPC=F)
CPU TIME = 0 REAL TIME = 0 LVHEAP = 2/5/5 SHARED = 0/1 OPS COMPLETE =
1 OF 2 ELAPSED TIME = 2
Operation COMPLETED on HDB 1 LVHEAP = 1/5/5
```

This report shows the operation completed on HDB 1, and the LVHEAP statistics are reported for HDB 1.

For a set of concurrent rule checks, you would see reports like this:

```

min_m1_space::<1> = EXT M1METAL < 0.18 ABUT < 90 SINGULAR
min_m1_width::<1> = INT M1METAL < 0.18 ABUT < 90 SINGULAR
-----
Operation EXECUTING on HDB 1 (T3 S1 E0 H0 A0 C3)
...
...
min_m1_space::<1> = EXT M1METAL < 0.18 ABUT < 90 SINGULAR
min_m1_width::<1> = INT M1METAL < 0.18 ABUT < 90 SINGULAR
-----
min_m1_space::<1> (HIER TYP=3 HGC=2356 FGC=2356 HEC=4712 FEC=4712)
min_m1_width::<1> (HIER TYP=3 HGC=3511 FGC=3511 HEC=7022 FEC=7022)
CPU TIME = 0 REAL TIME = 0 LVHEAP = 14/16/16 SHARED = 1/1 OPS COMPLETE
= 7 OF 13 ELAPSED TIME = 3
Operation COMPLETED on HDB 1 LVHEAP = 1/5/5
...
DRC RuleCheck min_m1_space COMPLETED. Number of Results = 10
Layer min_m1_space::<1> DELETED -- LVHEAP = 6/12/18 SHARED = 0/1
...
DRC RuleCheck min_m1_width COMPLETED. Number of Results = 10
Layer min_m1_width::<1> DELETED -- LVHEAP = 6/12/18 SHARED = 0/1

```

The statistics such as HGC, FGC, HEC, FEC, and so forth (see “[Layer Statistics in Hierarchical Processing](#)” for descriptions) may be higher in a hyperscaling run than in a non-hyperscaling run. This is because of how the geometry is partitioned for the run.

Summary

The summary section at the end of the transcript is similar to transcripts for non-hyperscaling runs.

Near the end of an MT run transcript, you will see a section like this:

```

Cumulative ONE-LAYER BOOLEAN Time: CPU = 0/2 REAL = 0/2
Cumulative TWO-LAYER BOOLEAN Time: CPU = 0/0 REAL = 0/0
Cumulative POLYGON TOPOLOGICAL Time: CPU = 0/2 REAL = 0/6
Cumulative POLYGON MEASUREMENT Time: CPU = 0/0 REAL = 0/0
Cumulative SIZE Time: CPU = 0/4 REAL = 0/4
Cumulative EDGE TOPOLOGICAL Time: CPU = 0/14 REAL = 0/15
Cumulative EDGE MEASUREMENT Time: CPU = 0/0 REAL = 0/0
Cumulative ONE-LAYER DRC Time: CPU = 0/2 REAL = 0/2
Cumulative TWO-LAYER DRC Time: CPU = 0/1 REAL = 0/1
Cumulative MISCELLANEOUS Time: CPU = 0/1 REAL = 0/1
Cumulative CONNECT Time: CPU = 5/0 REAL = 5/0
Cumulative RDB Time: CPU = 0/0 REAL = 0/0

```

This shows the CPU and REAL times in seconds for HDB 0 followed by the aggregate time for HDBs 1-4 in this form: “HDB 0/HDB 1-4”.

Although this excerpt shows the results of an MT run with hyperscaling, this formatting of HDB times using the / character applies in a MTflex run also for both local and remote hosts. See “[Unique Transcript Features of MTflex Mode With Hyperscaling](#)” for details.

The times are rounded to the nearest integer. However, the times are stored internally as floating-point numbers, which affects the cumulative times reported in this section from an MT transcript:

```
HDB 0: CPU TIME = 6 MAX LVHEAP = 28
HDB 1: CPU TIME = 8 MAX LVHEAP = 13
HDB 2: CPU TIME = 4 MAX LVHEAP = 9
HDB 3: CPU TIME = 7 MAX LVHEAP = 17
HDB 4: CPU TIME = 8 MAX LVHEAP = 15
HDB 0-4 TOTAL LVHEAP = 82
```

These lines show the cumulative CPU times in seconds for each of the HDBs. The maximum memory used for each of the HDBs and the total of all of these maxima are reported.

The Total Results Generated in a hyperscaling run may differ from a non-hyperscaling run. This is an expected behavior (see “[Hyperscaling](#)” on page 146) and is due to the selective flattening of certain data that is performed in a hyperscaling run.

The number of processors used (that is, they contribute to license usage) is reported on a line like this:

```
--- PROCESSOR COUNT = 4
```

Unique Transcript Features of MTflex Mode With Hyperscaling

The Calibre MTflex transcript with hyperscaling enabled is similar to the transcript produced in MT mode with hyperscaling, but the MTflex transcript has some unique features.

See the “[MT and MTflex Modes With Hyperscaling](#)” section for a complete discussion of elements common to both multithreaded modes. If you run in Calibre MTflex mode, you should use hyperscaling (-hyper option) unless you have good reasons not to (see “[Hyperscaling](#)” on page 146).

Header Information

Here is an example transcript of the header information generated from a Calibre job run in MTflex mode with hyperscaling. The differences in the output generated from a Calibre MT run with hyperscaling are highlighted in the following transcript:

```
// Running home/pkgs/icv/pvt/calibre64 -drc -hier -turbo -remotefile
mtflex.config -hyper rules
//
// Starting time: <time stamp>
//
// Reading remote host configuration file: mtflex.config
...
//
// HYPERSCALING ENABLED with 4 pseudo HDBs.
// Assigning RCSs to Pseudo HDBs, RCS allocated per PHDB = 12

// Initializing MTflex on pseudo HDB 1
// Calibre Remote Connection node#:12345
// Launching Calibre Remote server on: node
// Waiting for launch of remote hosts
// Launching Calibre Remote server on: node
// Connected to CPU on remote host node, logfile =
//           /tmp/CalibreRemoteLog.node.<n>.<time>.node.<id>
// Connected to CPU on remote host node, logfile =
//           /tmp/CalibreRemoteLog.node.<n>.<time>.node.<id>
// Connected to CPU on remote host node, logfile =
//           /tmp/CalibreRemoteLog.node.<n>.<time>.node.<id>
// Connected to CPU on remote host node, logfile =
//           /tmp/CalibreRemoteLog.node.<n>.<time>.node.<id>
// Connected to CPU on remote host node, logfile =
//           /tmp/CalibreRemoteLog.node.<n>.<time>.node.<id>
// Connected to CPU on remote host node, logfile =
//           /tmp/CalibreRemoteLog.node.<n>.<time>.node.<id>
// Connected to CPU on remote host node, logfile =
//           /tmp/CalibreRemoteLog.node.<n>.<time>.node.<id>
// Connected to CPU on remote host node, logfile =
//           /tmp/CalibreRemoteLog.node.<n>.<time>.node.<id>
// List of connected remote cpus:
// REMOTE CPU node: pid = 18372
// REMOTE CPU node: pid = 18373
// REMOTE CPU node: pid = 18374
// REMOTE CPU node: pid = 18375
// REMOTE CPU node: pid = 15225
// REMOTE CPU node: pid = 15226
// REMOTE CPU node: pid = 15227
// REMOTE CPU node: pid = 15228
// REMOTE CPU node: pid = 15229
// REMOTE CPU node: pid = 15230
// REMOTE CPU node: pid = 15231
```

```
//      REMOTE CPU node: pid = 15232
...

```

From this example, the -turbo, -remotefile, and -hyper command line options are shown. In a typical run, four pseudo HDB are initialized, in addition to HDB 0.

The contents of the MTflex configuration file are read (see “[Creating a Configuration File](#)” in the *Calibre Administrator’s Guide* for information about configuration files). The name of the remote host and the process ID are reported. This is repeated for each HDB.

A summary of the run configuration is provided:

```
//  MTflex initialization for pseudo HDBs completed.
//  Running on 12 CPUs (pending licensing)
//  List of connected remote hosts:
//    REMOTE HOST node: np = 8, nc = 2, ns = 4
//    REMOTE HOST node: np = 8, nc = 2, ns = 8
//  MTflex CPU resources: 8 Local, 12/16 Remote
//
//  LITHO operations available for use on 12 CPUs (pending licensing)
//
//  Initializing MT on HDB 0: 0
//  Creating log for pseudo HDB 1 on host, logfile =
//    /path/CalibreHDBLog.host.cpuid
//  Creating log for pseudo HDB 2 on host, logfile =
//    /path/CalibreHDBLog.host.cpuid
//  Creating log for pseudo HDB 3 on host, logfile =
//    /path/CalibreHDBLog.host.cpuid
//  Creating log for pseudo HDB 4 on host, logfile =
//    /path/CalibreHDBLog.host.cpuid
//  Initializing MT on pseudo HDB 1
//  Initializing MT on pseudo HDB 2
//  Initializing MT on pseudo HDB 3
//  Initializing MT on pseudo HDB 4

```

In this section, the total number of local and remote CPUs used for the run is reported. The names of the remote hosts and the processor numbers used on the remote hosts are reported.

For MTflex applications, the CPUs on the primary (master) machine often do not factor into the count in the “Running on N CPUs” line because they are used for data management purposes only, not layer operations. The CPUs on the primary typically process only the data generated by the remote CPUs, which always are considered “running” CPUs.

If the primary machine is *overdriven*, then some or all of its CPUs are used for runtime data generation in addition to the data management services they perform. Such CPUs are counted toward the total processor count for the run. The total number of CPUs being used for the run that count toward license usage is found on the line:

```
//  Running on ## CPUs
```

For example, suppose you have a local machine with four CPUs and your remotes have 16 CPUs. If you specify -turbo 18 (or -turbo_litho 18) on the command line, then the primary

machine is overdriven by two CPUs. These two CPUs then count toward your license usage (CPUs that are not overdriven do not count toward license usage), and are counted in the total processor count for the run. *It is not recommended that you overdrive the primary host.*

Injecting Hierarchy

The order of the cell names will differ between a MTflex run with hyperscaling and other modes of operation. This is due to how the data is partitioned for the different modes.

MTflex Small Cells

During a MTflex run, an algorithm for identifying small cells is used. This is an internal optimization step. You will see lines like this in the transcript indicating this step has been performed:

```
IDENTIFYING MTFLEX SMALL CELLS
MTFLEX SMALL CELLS COUNT: 39
```

CPU Time and Real Time

During a MTflex run, the transcript contains information on CPU time and real time.

For example:

```
CPU TIME = 1 + 0  REAL TIME = 1
```

For CPU TIME, the format is this: HDB 0 + HDBs 1-n (aggregate). The REAL TIME is the elapsed clock time. The numbers are stored internally as floating-point numbers. These numbers are rounded to the nearest integer for reporting purposes.

The cumulative time format is discussed in the Summary section that follows.

Calibre Executive Module

The Calibre Executive Module section of the report shows the differences from a non-MTflex run. The order in which various operations are performed, the completion of those operations, and the deletion of layers from memory is different from a non-MTflex run. This is due to how data is partitioned and operations scheduled in MTflex.

Reported times and LVHEAP numbers will also differ from other types of runs. This is due to network performance constraints and heuristics within MTflex.

Summary

For Calibre jobs run in MTflex mode using hyperscaling, the summary section at the end of the transcript contains statistics for both the local and remote hosts. The local host CPU numbers are reported before the + sign, and the remote host numbers are reported after it as follows: “local HDB 0/HDB 1-n + remote HDB 0/HDB 1-n”.

For example:

```
Cumulative ONE-LAYER BOOLEAN Time: CPU = 0/0 + 0/1  REAL = 0/2
Cumulative TWO-LAYER BOOLEAN Time: CPU = 0/0 + 0/0  REAL = 0/1
Cumulative POLYGON TOPOLOGICAL Time: CPU = 0/0 + 0/2  REAL = 0/2
Cumulative POLYGON MEASUREMENT Time: CPU = 0/0 + 0/0  REAL = 0/0
Cumulative SIZE Time: CPU = 0/1 + 0/2  REAL = 0/2
Cumulative EDGE TOPOLOGICAL Time: CPU = 0/0 + 0/11  REAL = 0/4
Cumulative EDGE MEASUREMENT Time: CPU = 0/0 + 0/0  REAL = 0/0
Cumulative ONE-LAYER DRC Time: CPU = 0/0 + 0/1  REAL = 0/0
Cumulative TWO-LAYER DRC Time: CPU = 0/0 + 0/1  REAL = 0/0
Cumulative MISCELLANEOUS Time: CPU = 0/0 + 0/0  REAL = 0/1
Cumulative CONNECT Time: CPU = 0/0 + 3/0  REAL = 1/0
Cumulative RDB Time: CPU = 0/0 + 0/0  REAL = 0/0
```

The semantics of this section are similar to the statistics (see “[Summary](#)” on page 229) in an MT hyperscaling run.

The REAL times are reported as follows: local host/remote hosts (aggregate).

This section contains additional statistics for each HDB, where the format of the CPU TIME is local host + remote hosts (aggregate):

```
HDB 0: CPU TIME = 2 + 4  MAX LVHEAP = 15
HDB 1: CPU TIME = 0 + 7  MAX LVHEAP = 9
HDB 2: CPU TIME = 1 + 5  MAX LVHEAP = 9
HDB 3: CPU TIME = 0 + 5  MAX LVHEAP = 9
HDB 4: CPU TIME = 0 + 5  MAX LVHEAP = 7
HDB 1-4 TOTAL LVHEAP = 34
```

At the end of the transcript, statistics are reported for the remote hosts by HDB and per CPU used for data processing, similar to this:

```
MTflex HDB 0
    REMOTE CPU node: CPU TIME = 0, STATUS = OK(0), LVHEAP = 1/1/2, RX = 0,
        TX = 0, RSS = 4222
    REMOTE CPU node: CPU TIME = 0, STATUS = OK(0), LVHEAP = 1/12/12, RX =
0,
        TX = 0, RSS = 4331
...
--- PROCESSOR COUNT = 12
```

The CPU TIME and LVHEAP statistics have the same meanings as elsewhere in the report. The only difference being that they pertain to the remote host, not the local host, and they pertain to the given HDB. The STATUS = OK(0) indicates communication with the remote CPU is normal. The RX and TX statistics are the numbers of megabytes received and transmitted, respectively, by the remote host for the given HDB.

For Litho operations, the summary may include an RSS statistic. Both RSS (Resident Set Size) and LVHEAP statistics report the amount of memory a process uses (not including swap). However, the RSS statistic is generated by the operating system memory management module, while the LVHEAP statistic is generated by the Calibre memory management module. For

Calibre runs, you should pay attention to the LVHEAP statistic (and not RSS) since Calibre uses LVHEAP to allocate memory for a job.

The Processor Count is the total number of local and remote processors used in the run. If only remote processors use licenses (the primary is not overdriven), then only remote CPUs are reported in this count. If the primary is overdriven, then the CPUs on the primary that use licenses are also reported in this count.

MT Mode Without Hyperscaling

If you use MT mode without hyperscaling, then the features of the transcript have some minor differences from a single-processor run, as well as from a hyperscaling run.

You should use hyperscaling (-hyper option) in MT mode unless you have good reasons not to. The “[MT and MTflex Modes With Hyperscaling](#)” section shows the transcript features of this mode.

Header Information

Here is an example of the header information generated from a Calibre job run in MT mode without hyperscaling. The differences in the output generated from a non-MT or MT hyperscaling run are highlighted:

```
// Running home/pkgs/icv/pvt/calibre64 -drc -hier -turbo rules
//
// Starting time: <time stamp>
//
// Running on 4 CPUs (pending licensing)
```

The -turbo option (or -turbo_litho in some applications) is used in an MT run. The number of CPUs is shown.

Runtime Statistics

The section describes the runtime statistics that are frequently different for a non-MT run.

- LVHEAP (these numbers are often larger for an MT run)
- REAL TIME (these numbers are often smaller for an MT run)
- ELAPSED TIME (these numbers are often smaller for an MT run)

Injecting Hierarchy

The order of the cell names will differ between an MT run and a non-MT run. This is due to how the data is partitioned for an MT run.

MTflex Mode Without Hyperscaling

If you use MTflex mode without hyperscaling, then the features of the transcript differ somewhat from a hyperscaling transcript.

You should use hyperscaling (-hyper option) in MTflex mode unless you have good reasons not to. The “[Unique Transcript Features of MTflex Mode With Hyperscaling](#)” section shows the unique transcript features of this mode.

Header Information

Here is an example of the header information generated from a job run in MTflex mode without hyperscaling. The differences in the output generated from a non-MTflex or MTflex hyperscaling run are highlighted in the following transcript:

```
// Running home/pkgs/icv/pvt/calibre64 -drc -hier -turbo -remotefile
mtflex.config rules
//
```

This is similar to the hyperscaling run transcript header (see “[Header Information](#)” on page 232) but lacks the -hyper command line argument, and the subsequent hyperscaling statistics.

Runtime Statistics

The section describes the runtime statistics that are frequently different for a non-hyperscaling run.

- LVHEAP (these numbers are often larger for a hyperscaling run)
- REAL TIME (these numbers are often smaller for a hyperscaling run)
- ELAPSED TIME (these numbers are often smaller for a hyperscaling run)

Hierarchical Database Construction

This portion of the transcript lists internal algorithms used during construction of the hierarchical database in memory. You will see differences in these algorithms versus a MTflex run with hyperscaling. This is because hyperscaling selectively flattens portions of the layout database.

Layer Read Summary (Original Layer Geometries)

This portion of the transcript may also differ from a hyperscaling run. The FINAL GEOMETRIES count could be different because of how the layout database has been partitioned for hyperscaling. The FINAL GEOMETRIES count will likely be lower for a non-hyperscaling run than for a hyperscaling run.

For example, this line of the transcript typically shows lower numbers than for a hyperscaling run:

LAYERNAME	68246 (132621)	84031 (132621)
-----------	----------------	----------------

Cell and Placement Summary

This portion of the transcript will likely show some larger numbers than the corresponding section of a hyperscaling run. This is because hyperscaling flattens certain parts of the database.

Layout Data Input Module Summary

This portion of the transcript will reflect the aggregate differences from a non-hyperscaling run described in the Layer Read Summary section.

Calibre Executive Module

This section of the report will show differences from a hyperscaling run. The order in which various operations are performed, the completion of those operations, and the deletion of layers from memory will be different from a hyperscaling run. Objects counts and memory usage can also differ from a hyperscaling run. This is due to how data is partitioned and operations scheduled in hyperscaling. The statistical summary at the end of the executive module will also differ from a hyperscaling run because there are no pseudo HDBs in a non-hyperscaling run.

DRC Results Database

A DRC results database is created for each DRC run. This file can be an ASCII, GDSII, or OASIS results database. ASCII is the most common DRC results database format.

Note

 ASCII databases can be read by Calibre RVE and viewed interactively with supported layout editors. See “[Using Calibre RVE for DRC](#)” in the *Calibre RVE User’s Manual* for more information on Calibre RVE for DRC. (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 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 78 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. Derived edge layer data becomes edge clusters of 1 edge each.

In addition to DRC results, each rule check in the DRC results database may also contain text that has been mapped from the rule file during DRC execution. This is called check text. Check text may consist of the rule check comments, 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.

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. 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, the ordering of rule check statements in the DRC results database corresponds to the order in the rule file.

Calibre nmDRC 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 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.

Calibre nmDRC/Calibre nmDRC-H allows you to output as many DRC results databases as needed during one Calibre nmDRC/Calibre nmDRC-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.

An ASCII DRC results database produced by Calibre nmDRC can be loaded by ICrules into Pyxis Layout. As such, a Calibre nmDRC results database can be scanned by the ICrules DRC results presentation commands. An ICrules DRC results database can also be written out as an ASCII DRC results database.

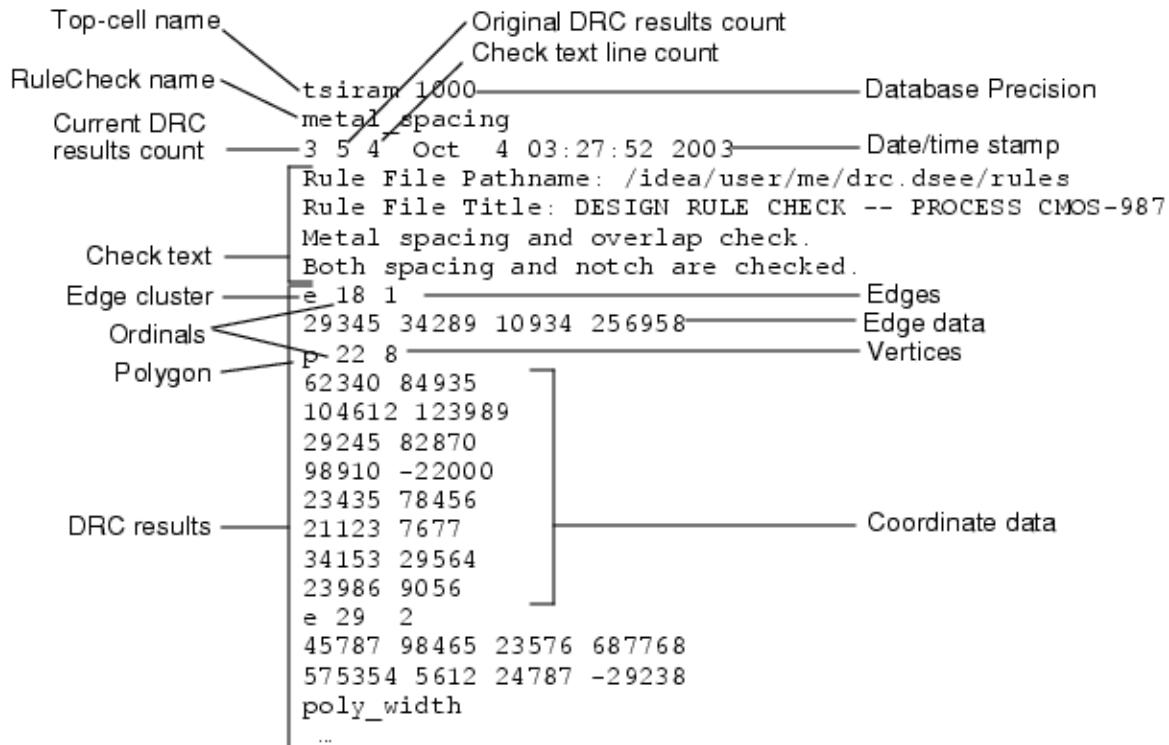
ASCII DRC Results Database Format	236
GDSII DRC Results Database Format	242
OASIS DRC Results Database Format	243
Result Count Limits	245
Hierarchical Calibre nmDRC Results Database	246

ASCII DRC Results Database Format

Calibre nmDRC generates an ASCII DRC results database by default. 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 8-1](#) shows an example ASCII DRC results database. Depending upon the operations that write to the database, properties and other elements can also appear.

Figure 8-1. ASCII DRC Results Database Example



RDBs	237
Cell Name and Database Precision	237
Rule Check Name, Result Count, and Execution Time	237
Check Text Report	237
Check Text for RDBs	238
Check Text for PRINT Files	239
Calibre nmDRC Result Listing	239
DRC Cell Name Results	240
Properties in ASCII DRC Results Databases	241

RDBs

Some layer operations can produce an ASCII results database that is ancillary to the global DRC results database generated by the DRC Results Database statement. These ancillary results databases are called RDBs (results databases), which is taken from the RDB keyword option. These RDBs are generally the same format as the global DRC results database, but they have calculated properties that do not appear in the global DRC results database.

Cell Name and Database Precision

The first line in the results database shows the top-cell name. The top-level cell name is given unless the Layout System is ASCII, in which case the string drc is given.

A number specifying the database precision (ratio of database units to user units) 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.

```
TOPCELL 1000
rule.extent
```

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 and time stamp, separated by one or more spaces.

```
rule.extent
1 1 2 May 23 12:24:43 2018
```

- The first number is the current count of DRC results for the rule check.
- The second number is the original count of DRC results. In results files produced by Calibre, this number is the same as the first number. In results files produced by ICVerify, this number can be greater than the first number.
- The third number is the number of [DRC Check Text](#) generated lines.
- The date/time stamp shows when the rule check was executed. The date/time format is as follows (blanks are significant):

mmm dd hh:mm:ss yyyy

Check Text Report

After the rule check name, result counts, and date and time stamp, the default check text shown includes the pathname of the rule file, title of the rule file (if any), and any rule check

comments. You can remove this information or you can add more information with the DRC Check Text specification statement.

```
Rule File Pathname: rules
Rule File Title: test rules
check text comment
```

Related Topics

[DRC Check Text \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Title \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Check Text for RDBs

The filenames for all RDBs created by certain operations like Density, Net Area Ratio, and DRC Check Map are listed (as separate DRC rule checks) at the end of the global DRC results database if it is ASCII type. The intent is for these filenames to be shown as an “error” in Calibre RVE and similar applications so that you do not inadvertently fail to notice their existence after an DRC run.

The DRC rule checks are named DENSITY_RDBS, NET_AREA_RATIO_RDBS, and DRC_RDBS, and each is only created if there are any corresponding RDB filenames to list. The RDB filenames are output as check text. For example:

```
...
13089600 132400
13089600 272400
12949600 272400
DENSITY_RDBS
0 0 3 Aug 1 12:55:23 2011
/net/illamp/density_rdbs/M1_density.rdb 1356
/net/illamp/density_rdbs/M2_density.rdb 0
/net/illamp/density_rdbs/M3_density.rdb 44
NET_AREA_RATIO_RDBS
0 0 2 Aug 1 12:55:23 2011
./nar_rdbs/M1_antenna.rdb 0
./nar_rdbs/M2_antenna.rdb 560
DRC_RDBS
0 0 1 Aug 1 12:55:23 2011
./drc_rdbs/drc_results.asc 1
```

The result count is shown after the filename.

If the DRC Check Map results database coincides with that in the DRC Results Database statement then it is not listed.

Related Topics

[DRC Results Database \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Density \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Net Area Ratio \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Check Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Check Text for PRINT Files

The filenames for all PRINT files created by Density PRINT and Net Area Ratio Print are listed (as separate DRC rule checks) at the end of the global DRC results database if it is ASCII type. The intent is for these filenames to be shown as an “error” in Calibre RVE and similar applications so that you do not inadvertently fail to notice their existence after a DRC run.

The two DRC rule checks are named DENSITY_PRINT_FILES and NET_AREA_RATIO_PRINT_FILES, and each is only created if there are any corresponding filenames to list. The filenames are output as check text. For example:

```
...
DENSITY_PRINT_FILES
0 0 2 Oct 22 08:58:52 2010
den.print.1
den.print.2
NET_AREA_RATIO_PRINT_FILES
0 0 1 Oct 22 08:58:52 2010
nar.print
```

Related Topics

[Density \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Net Area Ratio Print \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Calibre nmDRC 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.

```
p 1 4
-20600 0
45000 0
45000 85000
-20600 85000
...
e 1 2
500 12000 500 28250
3500 12000 3500 28250
```

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.

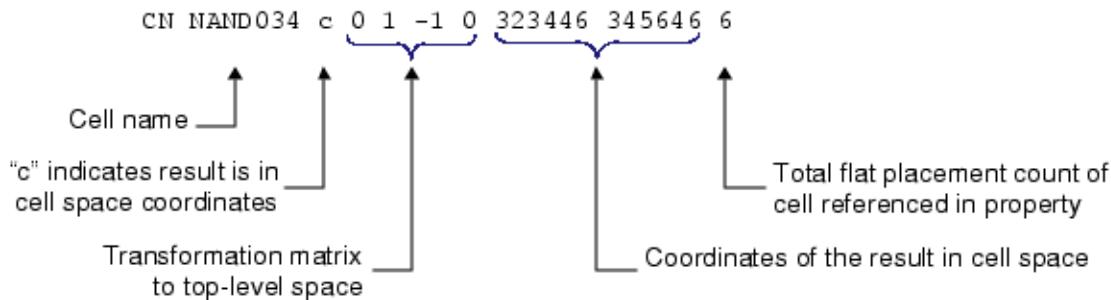
- For polygons, the coordinate data include a list of coordinates; each coordinate occupies one line showing the x-coordinate then the y-coordinate, separated by one or more spaces. The coordinates are listed in counterclockwise order; the number of coordinates corresponds to the vertex count on the 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 YES option in DRC Cell Name or ERC Cell Name specification statements, the results database contains additional cell name information. The beginning CN characters stand for “Cell Name.” The CELL SPACE and XFORM keywords cause additional information to be printed in the results database. This property is described in detail in the *SVRF Manual* under each specification statement’s section.

```
e 1 2
CN nand034 c 0 1 -1 0 323446 345646 6
500 12000 500 28250
3500 12000 3500 28250
```

Figure 8-2. Description of Cell Name CELL SPACE XFORM Property



Related Topics

[DRC Cell Name \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Using Calibre RVE for DRC \[Calibre RVE User's Manual\]](#)

[ERC Cell Name \[Standard Verification Rule Format \(SVRF\) 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).

Properties appear 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 Calibre RVE.

Related Topics

[Density \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Net Area Ratio \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Cell Name \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Using Calibre RVE for DRC \[Calibre RVE User's Manual\]](#)

GDSII DRC Results Database Format

Calibre nmDRC generates a GDSII DRC results database if the GDSII keyword is specified in the DRC Results Database or DRC Check Map specification statement.

Note

 This information applies only to flat Calibre applications. Refer to section “[Hierarchical Calibre nmDRC Results Database](#)” on page 246 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 these are true:

- 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 1 of the common era 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.

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 or OASIS), 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.

For Calibre nmDRC-H, the database hierarchy is preserved in a GDSII DRC results database and there is no suppression or top-level transformation of DRC results (as with ASCII type DRC result databases). Cell names are output with an optional string appended, prefixed, or both, as specified by the respective APPEND or PREFIX options in the DRC Results Database statement.

Calibre nmDRC/Calibre nmDRC-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.

Note

 The limit for coordinate space is 32-bit for a GDSII DRC results database. When writing a GDSII-type DRC results database, Calibre aborts if a coordinate exceeds the 32-bit space capacity of GDSII.

Related Topics

[DRC Check Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Precision \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Results Database \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Unit Length \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Layout Primary \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

OASIS DRC Results Database Format

Calibre nmDRC generates an OASIS DRC results database if the OASIS keyword is specified in the required DRC Results Database or DRC Check Map statement. 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):

```
<magic-bytes> START <name>* PROPERTY* <cell>* END
<name> -> CELLNAME | TEXTSTRING | LAYERNAME | PROPNAME | PROPSTRING
<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, XNAME, XYRELATIVE, XYABSOLUTE, XELEMENT, TRAPEZOID, CIRCLE, or XGEOMETRY records present. There are no forward references to <name> records.

For Calibre nmDRC, 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 or OASIS) then the cell name is drc.

For Calibre nmDRC-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 type DRC result databases). Cell names are output with an optional string appended, prefixed, or both, as specified by the respective APPEND or PREFIX options in the DRC Results Database 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 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. Specifying mask results output with a layer or datatype number greater than 65535 is disallowed.

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 nmDRC 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 or 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 employed.
- OASIS LAYERNAME records are output to an OASIS DRC Results Database for all DRC rule checks which map to that database. The record consists of the name of the

DRC rule check and the layer/datatype specified in the appropriate DRC Check Map statement (or the default 0/0).

- The PSEUDO and USER options for the DRC Results Database and DRC Check Map statements exhibit much greater scalability in MT and MTflex runs than does USER MERGED for OASIS results.

Related Topics

[DRC Check Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Results Database \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Result Count Limits

Calibre nmDRC allows you to specify an upper bound on the number of DRC results per rule check written to the 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.

You may want to limit the number of results in the DRC results database. This is useful when debugging the rules (on a large database) or during initial checks of new databases.

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 denotes 18446744073709551614 for 64-bit machines. This value defaults to 1000 if the statement is omitted.

The DRC results database initialization module for Calibre nmDRC applications prints a warning if any DRC rule check outputs to a GDSII- or OASIS-type DRC results database with a maximum result count less than ALL. This setting is available from the DRC Maximum Results and the [DRC Check Map](#) specification statements. The warning is printed for each results database so generated. The intent is to warn of inadvertent use of the default value 1000 when writing mask data versus DRC errors. Setting the environment variable CALIBRE_EXIT_MAXIMUM_RESULTS to a non-null value causes Calibre nmDRC applications to immediately exit after any such warnings are printed.

The DRC results database initialization module for Calibre nmDRC applications also prints a warning for each DRC rule check that outputs to a GDSII- or OASIS-type DRC results database with a default layer number of 0. This may occur because a layer number is not specified in the DRC Check Map specification for the rule check, or because no DRC Check Map is specified for the rule check.

Hierarchical Calibre nmDRC Results Database

A hierarchical DRC results database is produced by Calibre nmDRC-H. For GDS and OASIS output, the Calibre representation of the database hierarchy is preserved, and there is no suppression or top-level transformation of DRC results (as is found with ASCII type DRC results databases). Cell names are output with an optional string appended; this optional string is determined by the DRC Results Database specification statement.

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 and ensures 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 that contains the result data prior to being instantiated in the DRC results database. Note that pseudocells of this type cannot be used as hcells in LVS.

Hierarchical Mask Results

For a mask data Calibre nmDRC-H results database, output geometry appears in cells of the original design hierarchy and is not merged (overrides of this default are discussed later in this section). Any geometry in a Calibre database pseudocell (ICV_n cell) is promoted to the next highest user cell. This behavior is controlled through the PSEUDO, USER (general default), and USER MERGED keywords of the [DRC Results Database](#) and [DRC Check Map](#) statements.

USER is similar to USER MERGED except that USER does not merge the transformed polygons. Hence, the USER option provides performance close to that of the PSEUDO option, while allowing you to suppress output of pseudo-hierarchy. The drawback of the USER option versus USER MERGED is that output polygons on the same layer may abut or overlap without merging.

For GDSII- and OASIS-type DRC results databases that are used for mask-preparation, as opposed to DRC checking, use of the PSEUDO keyword can dramatically reduce output database size and write time, and is recommended.

The default behavior is chosen as follows:

- PSEUDO is the default if the results database type is GDSII or OASIS, and any Litho, Fracture, or [DFM Fill](#) operation is required in the flow.
- PSEUDO is the default if the results database type is GDSII or OASIS and [LAYOUT TURBO FLEX](#), [LAYOUT ULTRA FLEX](#), or [LAYOUT MONDOFLEX](#) is enabled.
- USER is the default for configurations other than the preceding two.

The [DRC Keep Empty](#) specification statement, which regulates retention of empty rule checks in ASCII type DRC results databases generated by Calibre nmDRC applications, also affects geometric DRC results databases written by Calibre nmDRC-H. When DRC Keep Empty NO is specified in the rule file, then cell and placement records are not written to geometric results databases if the cell or the placement’s cell, respectively, contain no results.

Related Topics

[Calibre nmDRC-H Results Data Storage](#)

DRC Summary Report

The DRC Summary Report specification statement in the rule file writes a summary of a DRC run. This report consists of heading information, runtime warnings, original layer statistics, rule check results statistics, and other information that encapsulates the run. The HIER option is recommended for hierarchical runs.

Heading information — The first part of the [DRC Summary Report](#) lists general information about the run. For example:

```
=====
== CALIBRE:::DRC-F SUMMARY REPORT
==
Execution Date/Time:      Wed Apr 28 15:04:33 2003
Calibre Version:          <version>
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  ANGLED = NO  OFFGRID = NO
                           NONSIMPLE POLYGON = NO  NONSIMPLE PATH = 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. By default, the DRC summary report also lists the rule checks that were not performed. You can use the EXECUTED option to suppress output of rule checks that were not performed.

In DRC-H, rule check results are shown both hierarchically and flat. Here is an example:

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

The first result count is the number of results counted once per cell that are sent to the global DRC Results Database (results sent to DRC Check Map or DFM RDB databases are not included in this count). 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.

Chapter 9

Connectivity Extraction

Connectivity extraction recognizes electrically-connected regions in the layout called nets. Nets are recognized from layout shapes through analysis of the relations between the shapes and other objects on various layers such as text objects.

Each electrical net is given a unique node number for identification during connectivity extraction. Geometry that has the same node number is part of the same net. In addition to the node number, the net may also be named based upon the presence of layout text objects or text statements in the rule file.

When connectivity extraction is performed on a layer, all geometry on the layer receives a node number. Geometry that becomes part of the same net is defined through various rule file constructs, chiefly the Connect and Sconnect operations. Numerous layer operations require connectivity to be established in order for the operations to be valid.

When connectivity extraction is required, the verification system performs this automatically according to the rule file configuration. Connectivity extraction is used by Calibre DESIGNrev, Calibre nmDRC, Calibre nmLVS, Calibre PERC, Calibre YieldEnhancer, Calibre YieldAnalyzer, Calibre xRC, and Calibre xL applications.

Mask Connectivity Extraction	249
Connectivity and Rule File Compilation	250
Electrical Net Recognition.....	252
Use of Text in Calibre Applications	258
Incremental Connectivity and Antenna Checks	266
Connectivity Warnings in Calibre nmDRC	274
LVS Circuit Extraction	275

Mask Connectivity Extraction

Mask connectivity extraction analyzes 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, such as DRC and LVS.

Mask connectivity extraction internally merges overlapping shapes and paths on any single database layer. References to the original database objects are not maintained. Hierarchical Calibre tools analyze the layout hierarchy; connectivity is then extracted from completely flat shapes. For efficiency, Calibre tools only extract connectivity for layers that are actually required to produce outputs for the run.

Calibre nmDRC only uses connectivity information internally for connectivity-dependent rule checks. No persistent database containing connectivity is produced. The layer operations that require this are discussed under “[Operations Requiring Layer Connectivity](#).”

ERC, which is performed as a part of LVS connectivity extraction, behaves similarly to DRC connectivity extraction. (However, if the [Mask SVDB Directory](#) statement is present during the ERC run, layout connectivity information is stored in that database.)

DFM applications are similar to DRC in many respects; however, layer connectivity is stored in the DFM results database, something that is not done for DRC.

In Calibre nmLVS-H or Calibre PERC, circuit extraction can be initiated by the `-spice` command line argument. The hierarchical SPICE netlist of the layout is written to the pathname that you specify. Circuit extraction is also triggered when your Layout System is geometric. In LVS, the extracted layout netlist is written to the Mask SVDB Directory (if specified) as *layout_primary.sp* or to the current working directory as *lvs_report_name.sp*. In Calibre PERC, the `-spice` command line option or the Mask SVDB Directory statement must be used when connectivity extraction is required.

If the `-flatten` option is used, then the layout is flattened and a flat layout SPICE netlist is produced. LVS comparison is netlist-to-netlist when `-lvs` is also specified. Other runtime behaviors are similar to what occurs in Calibre nmLVS-H.

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. Direct connectivity extraction is discussed in the *ICverify Manual*.

Connectivity and Rule File Compilation

Rule file compilation verifies that any operation requiring connectivity has the requisite nodal information on the appropriate input layer(s).

In order to have connectivity, a layer must satisfy one of these conditions:

1. Appear directly in a [Connect](#) or [Sconnect](#) operation.
2. Be derived by a sequence of node-preserving operations from a layer appearing directly in a Connect or Sconnect operation.
3. Be derived by a sequence of node-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. If case 1 does not apply to the second input layer, then the second input layer may not appear in the layer derivation chain of a layer that appears as an argument to a Connect or Sconnect operation.

Note

 **DRC Incremental Connect** YES relaxes some of the connectivity derivation criteria. See the *SVRF Manual* for details.

Operations Requiring Layer Connectivity

Certain layer operations require connectivity on the layers they process. Connectivity is only extracted for layers that actually require it.

Layers require connectivity if they meet any of the following criteria:

- The layer appears in an operation that requires connectivity to be established. The following layer operations require connectivity to be established on their input layers.

Table 9-1. Connectivity-Dependent Layer Operations

[Not] Net	Net Area	Net Area Ratio
Net Interact	Ornet	Stamp
ERC Pathchk	Pathchk	DFM Property Net
DFM Property NODAL	DFM Stamp	DFM DV

The following operations with the CONNECTED or NOT CONNECTED keywords:

Enclosure	External	Internal
[Not] With Neighbor	DFM Property	

The following operations with the BY NET keyword:

[Not] Cut	[Not] Enclose	[Not] Interact
[Not] Touch		

These operations trigger the assignment of nodal information to electrical nets on layers that appear in these operations. If none of these operations appears in an DRC run, then connectivity is not extracted for that run.

- The layer appears as a pin layer in a **DEvice** statement.
- The layer appears in a DFM operation containing the NODAL keyword. See the *Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual* for more information about DFM operations.
- The layer appears as the first input layer to a node-preserving operation, and the output layer from that operation requires connectivity. This is a recursive criterion. See “**Node-Preserving Layer Operations**” on page 90 for additional information about node-preserving operations.

Connectivity is extracted in a single step before any layer operations are performed that either require connectivity on input layers or pass connectivity to derived layers. Derived layers that appear in Connect or Sconnect operations are generated before connectivity is established. For this reason, if a layer is derived from a node-preserving operation and the layer appears in a Connect or Sconnect operation, then that node-preserving operation does not pass node IDs.

There are two exceptions to the preceding paragraph. First, DRC Incremental Connect YES causes connectivity to be derived in stages in the rule file. This is discussed under “[Incremental Connectivity and Antenna Checks](#)” on page 266. Second, if a node-preserving operation appears in the local scope of a rule check, and that operation derives a layer using the same name as one in a Connect or Sconnect statement, then the locally derived connectivity is used. For example:

```
CONNECT a b
CONNECT c

b = a AND d // global scope
...
rule {
  b = c AND d // local scope; b gets c's node ID in the local scope
}
}
```

Electrical Net Recognition

A number of rule file statements cause connectivity to be established on layers. The shapes that have the same connectivity are recognized as being on the same electrical net.

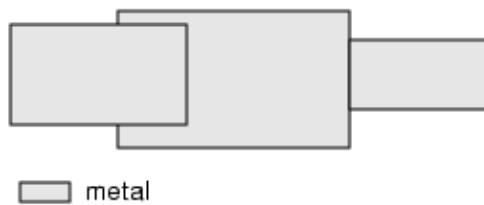
Connect

Interconnect layers are layers that appear in Connect or Connect BY operations in the rule file.

Abutting or overlapping polygons on a *single* interconnect layer are always considered to be part of a single net, such as those in the following figure. Polygons that touch only at corners are not considered to be part of the same net.

Figure 9-1. Connected Shapes on a Single Layer

CONNECT metal

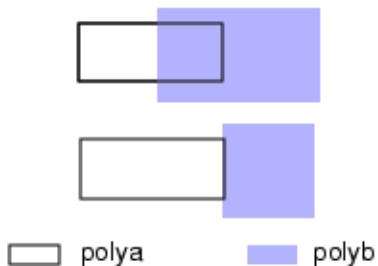


Polygons on different layers can be connected directly by overlap or abutment, as specified in **Connect** operations in the rule file. Again, polygons touching at corners do not form valid connections.

[Figure 9-2](#) shows an example of a valid connections for two input layers.

Figure 9-2. Connected Shapes from Different Layers

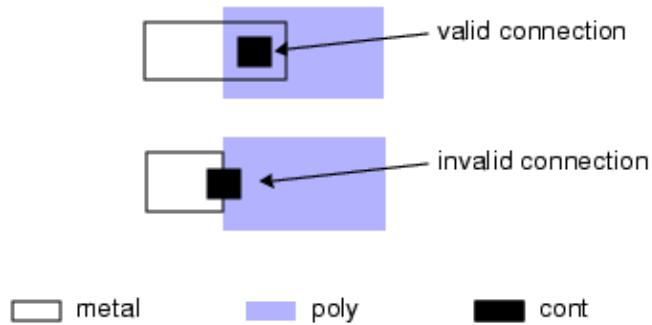
CONNECT polya polyb



Polygons on two interconnect layers can be connected to each other by mutual intersection with a third polygon on a contact layer. This is specified in a Connect BY operation in the rule file. [Figure 9-3](#) shows an example. If all three polygons (one from each layer) do not have a common intersection, then there is no connection.

Figure 9-3. Polygons Connected By Contact

CONNECT metal poly BY cont



Shielding

A Connect BY operation causes *shielding* if it has this form:

CONNECT layer1 layer2 layer3 ... layerN BY layerC

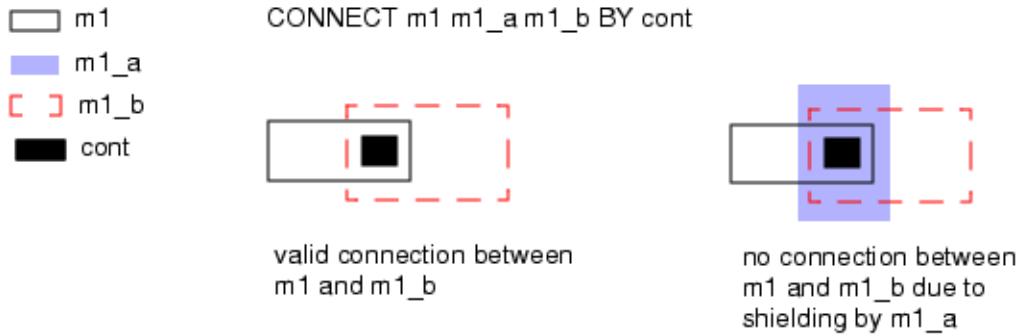
The distinguishing feature is there are at least three *layerN* parameters.

A similar behavior occurs for the Sconnect BY operation with more than two layers before the BY keyword. The remainder of this discussion is in the context of Connect, but it also applies to Sconnect.

Connections are always formed between *layer1* and *layerC* when they mutually overlap a third layer from *layer2* through *layerN*. The first available object from *layer2* through *layerN*, in that order, that intersects *layer1* and *layerC* at a given location participates in the connection; no other layers from *layer2* through *layerN* participate.

For example, in the preceding Connect BY statement, connections between *layer1* and *layer3* are only made if there are no *layer2* polygons present at the point of connection. This can cause unexpected results in some cases. See [Figure 9-4](#).

Figure 9-4. Shielding



In Calibre nmLVS-H, the Connect BY statement in [Figure 9-4](#) could break the connectivity of hcells. In the example in the right-hand figure, the connectivity extractor has to search the entire hierarchy to determine if any *m1_a* objects are present. If so, it could be necessary to promote *m1_b* objects whose connectivity cannot be determined out of a cell and up to a level of hierarchy where connectivity can be determined. This can cause undesired side-effects like the promotion of pins, which in turn leads to incorrect connectivity in hcells. Additionally, there is a performance penalty for Connect BY statements with shielding.

Given a shielded Connect statement:

```
CONNECT a b c BY d
```

The simplest way to remove the shielding is to do this:

```
CONNECT a b BY d
CONNECT a c BY d
```

which is the recommended method in most cases.

In some cases, downstream parasitic extraction tools may require that the contact/via polygons be separated for each transition. If that is needed, you can do something like this:

```
d1 = d INSIDE (a AND b)
d2 = (d NOT INSIDE (a AND b)) INSIDE (b AND c)
// you get "d NOT INSIDE (a AND b)" for no extra overhead due to
// concurrency with the d1 derivation
CONNECT a b BY d1
CONNECT a c BY d2
```

There could be increased memory and runtime costs for the contact/via layer derivations, however.

In cases such as the one in [Figure 9-4](#) where layers m1_a and m1_b are metal routing layers derived from m1, it is wasteful to include m1_a and m1_b in Connect BY statements. It would be better to establish connectivity on m1 using a common (and necessary) statement such as this:

```
CONNECT m1 poly BY cont
```

With connectivity established on m1 by such a statement, you are free to derive any layers from m1 that you need using [Node-Preserving Layer Operations](#) such as AND and NOT. For example:

```
// these layers inherit nodal information from m1, which appears in a
// Connect BY operation.
m1_a = m1 AND a_layer
m1_b = m1 NOT b_layer
```

This way, all m1 routing layers have their connectivity established without the problems of shielding or wasteful Connect BY operations.

In some cases, node-preserving operations may not be applicable, or a layer may be derived from an operation that does not preserve nets. In such cases, the [Stamp](#) operation can be useful. For example:

```
CONNECT m1
m1_a = SIZE m1 BY -1
```

Layer m1_a does not have connectivity because Size is not a node-preserving operation. You could then write this:

```
stamp_m1_a = STAMP m1_a BY m1
```

There is another beneficial method to use in situations like this one. In the following case, layer m1_a is guaranteed to be contained inside m1. You can take advantage of the containment and restore the connectivity like this:

```
copy_m1_a = m1 AND m1_a
```

The key feature of this method is the derived layer (in this case, copy_m1_a) must be a subset of the originating layer (m1 here).

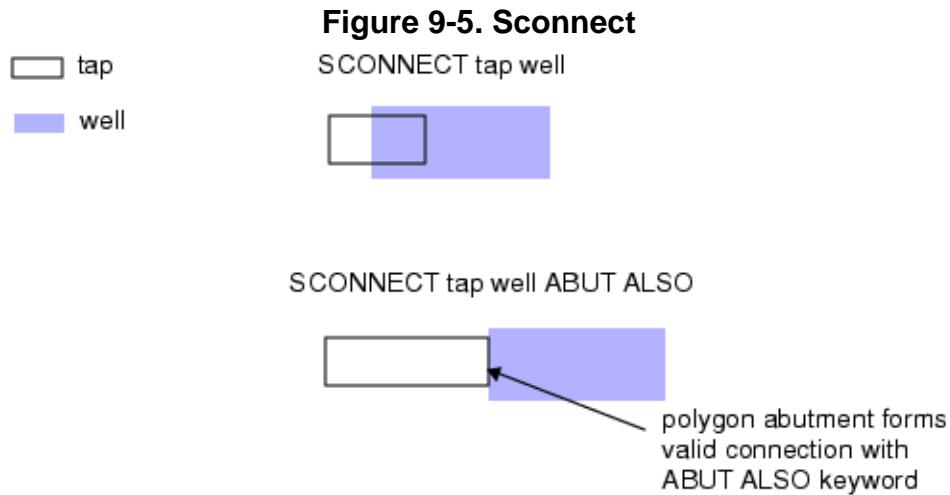
In general, Connect BY operations with shielding should be avoided. However, in situations involving multiple substrate layers, use of shielding may be necessary. In such cases, the Connect BY operations usually list the layers from top to bottom from the perspective of the cross-section of the chip's layer profile.

Sconnect

Sconnect establishes one-way connections from an upper layer to a lower layer.

Sconnect is often used to detect soft connections (high-resistance connections) through well regions. 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.

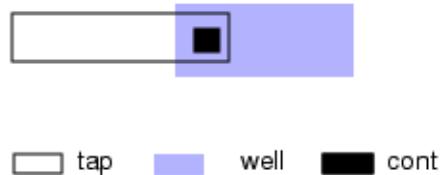
The Sconnect operation without a contact layer forms unidirectional connections from the upper layer to the lower layer when the layers overlap. Polygons that abut can form connections if you specify the ABUT ALSO keyword. Polygons touching only at corners do not form valid connections.



You can establish connectivity by specifying a contact layer and multiple lower layers using the Sconnect BY keyword. [Figure 9-6](#) shows an example. As with the Connect BY operation, all three polygons from the three layers must intersect a common point for the connection to be valid.

Figure 9-6. Sconnect Operation

SCONNECT tap well BY cont



Sconnect BY is subject to [Shielding](#) in a similar way to Connect BY. In situations where connections are established by polygon promotion due to shielding, Sconnect attempts to push the connections down to the appropriate hierarchical level; however, this cannot be guaranteed. In certain situations, the promotion of polygons can cause connectivity problems in hcells. For this reason, it is good practice to have only three layers in any Sconnect BY operation: two interconnect layers and the contact layer.

Stamp

Stamp is a layer operation used 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 metal1 polygons, that can receive valid connectivity information from metal1 polygons. The connectivity information is assigned to the derived layer named x.

```
x = STAMP polyc BY metal1
// x is a derived layer constructed from polyc
// with metal1 nodal 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 an 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 not allowed to connect a contact layer directly 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.

Use of Text in Calibre Applications

Text objects that are present in the layout database or that are created in the rule file are processed for a number of purposes.

- Net naming in connectivity extraction.
- Port naming in LVS, Calibre PERC, and PEX applications.
- Using TEXT MODEL LAYER and TEXT PROPERTY LAYER in **DEvice** statements.
- Using [Not] **With Text** and **Expand Text** operations in DRC.
- Mapping text objects to mask Calibre nmDRC-H output using **DRC Map Text YES**.

The following statements control text object processing in Calibre.

Table 9-2. Text Processing Statements

Attach	Label Order	Text
DRC Cell Text	Layout Ignore Text	Text Depth
DRC Map Text ¹	Layout Preserve Case	Text Layer
DRC Map Text Depth ¹	Layout Property Text ¹	Port Depth
Expand Cell Text	Layout Rename Text ¹	Port Layer Text
Expand Text ¹	Layout Text ¹	[Not] With Text ¹
	Layout Text File ¹	

1. Statement may or may not affect connectivity extraction text objects.

Net Name Specification

Net names can be specified in the rule file, be written in the layout database as text objects, or both.

The typical way to assign net names is through layout database text objects (geometric layout such as GDSII or OASIS). 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's basepoint. The label layer is the layer of that object, and the label name is the value of that object. The database text object must intersect some location on the net in order for the label to be assigned to the net. Layout text objects are assigned to layers using one of three label attachment methods. These are discussed in the section "**Net Label Attachment**."

The **Text Layer** specification statement tells Calibre the layers from which layout text objects are read for connectivity extraction. If you do not use a Text Layer statement, then no layout database text objects are read for connectivity extraction.

The [Text Depth](#) specification governs the hierarchical depth from which to read layout text objects as connectivity extraction text. The default is top level. Text objects (from recognized text layers) 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 Calibre nmDRC-H application, only those layout database text objects selected by a Text Depth statement are used in the connectivity extractor.

In Calibre nmLVS-H, 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.

These are the ways to specify net names in the rule file:

- **Text specification statements** — The [Text](#) specification statement allows you to specify free-standing text directly in the rule file without the need for text objects in the layout. It also allows you to edit text objects read from the layout database.

The label has a specified *name*, *x y location*, and *layer*. The label applies to the top level of the hierarchy only. This statement is used only in connectivity extraction. The [Text Layer](#) and [Text Depth](#) statements do not apply to text objects entered with [Text](#) specification statements.

- **Layout Text specification statements** — The [Layout Text](#) specification statement is applicable only when the Layout System is geometric. The Layout Text statement allows you to specify free-standing cell-based text directly in the rule file. The text object behaves exactly as if it were in the layout database.

The label has a specified name, x y location, layer, and cell. The coordinates are cell coordinates, with top-level being the default. This statement is used in connectivity extraction as well as for other purposes. Like any database text, Layout Text objects can be used both in hierarchical and flat applications.

The [Layout Text File](#) statement specifies a file that uses similar syntax to Layout Text. The text objects specified in a layout text file act like Layout Text objects. The Layout Text File statement should be used instead of Layout Text when you have many rule file texts to specify.

The [Text Layer](#) and [Text Depth](#) statements apply to Layout Text and Layout Text File statements.

The following are differences between text objects in Text and Layout Text specification statements:

- Text statement objects are always in global coordinates and have no cell association. Layout Text objects use cell coordinates.
- Text statement objects can override existing database text, whereas 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 statement objects do not.
- Text statement 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, whereas Text statement objects do not.
- Only simple layer numbers can be associated with Layout Text objects. Text statement objects can have original layer names.

Connectivity extraction is case-insensitive by default. For example, net names “abc” and “ABC” are not netlisted separately (unless requested with the [Layout Preserve Case YES](#) statement). In the default mode, the connectivity extractor selects one of these labels and discards the other. The net for which the label is discarded receives a numeric ID.

Labels have leading and trailing whitespace (including newlines) and non-printable characters removed from names. If a label name consists entirely of whitespace or non-printable characters then the label is discarded.

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.

See also the [Layout Property Text](#), [Layout Rename Text](#), and [Layout Ignore Text](#) specification statements in the *SVRF Manual*.

Logical Information on Merged Layers

Layout verification applications that use connectivity extraction merge original database layers internally before using them for any other operations.

In LVS, polygons on those merged layers inherit port and pin information from shapes and paths of the corresponding original layers, namely:

- A polygon on an internally-merged layer that originated from a shape or path of a port is considered to belong to the port for the purpose of connectivity extraction.
- A polygon on an internally merged layer that originated from a shape or path of an instance pin is considered to belong to the instance pin for the purpose of connectivity extraction.

After merging has occurred, labels are attached through various rule file statements.

Related Topics

[Net Label Attachment](#)

[Net Name Specification](#)

Net Label Attachment

The connectivity extractor generates internal label objects to represent text data from various sources: rule file Layout Text, Layout Text File, and 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.

There are three primary means for assigning label names to geometric objects:

- [Attach](#) specification statement.
- Text labels on a given [Connect](#) layer that intersect polygons on that same layer.
- [Label Order](#) specification statement.

These methods do not apply to the [Text](#) statement.

The [Attach](#) operation transfers connectivity information from a specified original database source layer (a text layer) to a specified original or derived target layer that appears in a [Connect](#) or [Sconnect](#) operation. Most often, [Attach](#) is used when there is a single text layer (or a few text layers) that is used for label assignment throughout the design.

For LVS purposes, port locations are transferred from port objects on the first input layer (layer1) of the [Attach](#) operation to overlapping polygons on the second input layer (layer2). Port names are transferred along with port locations. In these cases, layer1 polygons 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. 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. See “[Netlisting of Ports and Pins](#)” on page 282 for additional information.

You can specify a [Label Order](#) specification statement that determines the order in which connectivity extraction looks for shapes that intersect a label location. Label Order only applies to text layers that do not appear in Connect or Sconnect statements, and that do not appear in Attach statements.

Connectivity extraction attaches labels to nets in the following order of priority:

1. **Explicitly** — If the rule file contains the Attach operation:

```
ATTACH A B
```

where A is a [Text Layer](#) argument (or A is a layer set that contains the label layer), then the connectivity extractor looks for a polygon on layer 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 these:

```
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, then the label is ignored and a warning is issued.

Rule file excerpt:

```
TEXT LAYER poly.txt
CONNECT metal poly BY contact
ATTACH poly.txt poly

TEXT LAYER "text"
CONNECT m1 po BY co
ATTACH text m1
ATTACH text po
```

2. **Implicitly** — If the rule file contains a Connect operation:

```
CONNECT ... A ...
```

where A is a [Text Layer](#) argument (or A is a layer set that contains the text label layer), then the connectivity extractor looks for a polygon on A that intersects a text label location. If found, the label name is assigned to the net that contains that polygon.

Example:

```
TEXT LAYER metal // text objects exist on this layer
CONNECT metal poly BY cont
```

Here, any text labels on the metal layer are implicitly attached to the metal layer where the text objects intersect metal polygons.

3. **Freely** — If the rule file contains the [Label Order](#) operation:

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

then the layer order specified in the operation is used for label attachment priority. 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 any layer B_n appears in a Connect, Sconnect, or Attach statement, then that layer does not participate in any free attachment from Label Order.

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:

```
TEXT LAYER txt
CONNECT metal poly BY cont
LABEL ORDER metal poly
```

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

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:

Example 9-1. Explicit Text Label Attachment

```
src_drn = diff NOT poly
CONNECT src_drn metal BY contact
TEXT LAYER diff
```

Diff and poly are original database layers. If you include this statement in the rules:

```
ATTACH diff src_drn
```

then src_drn receives the nodal information from diff. To assign a net name directly to a source or drain region, place a text object on the diff layer over this region.

If you had the following statement in the rule file:

```
CONNECT diff
```

then layout text objects from the diff layer *would not be* implicitly attached to diff polygons so long as the Attach statement is present. This is because explicit attachment overrides implicit attachment of text labels.

Similarly, if you had this statement in the rule file:

```
LABEL ORDER diff
```

layout text objects from the diff layer would not be freely attached to diff polygons because of the Attach statement. Explicit attachment overrides free attachment of text labels.

- **Implicit attachment** — Suppose the original database layers poly and metal appear in your database. Further suppose you have the following rule file statements:

Example 9-2. Implicit Text Label Attachment

```
CONNECT a
CONNECT b
TEXT LAYER b
```

If you have text objects on layer b, these text objects are implicitly attached to polygons from layer b when the text objects' base points intersect such polygons.

If you have the following statement in your rule file:

```
LABEL ORDER b a
```

layout text objects from layer b *would not be* freely attached to layer a. This is because of the CONNECT b statement. This statement causes implicit attachment of layer b text objects to layer b polygons, and this overrides whatever free attachment may occur in the Label Order statement.

- **Free attachment** — Suppose you use the database layers metal1, metal2, and poly for interconnect and you want to use database layer "txt" to specify net names using any of the three interconnect layers. To do this, include the Label Order statement in the rules:

Example 9-3. Free Label Attachment

```
CONNECT poly metal1 BY contact
CONNECT metal1 metal2 by via

TEXT LAYER txt
LABEL ORDER metal1 metal2 poly
```

To assign a name to a net, place a text object from the txt 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 to receive the label name.

Open and Short Circuits

In Calibre applications, open and short circuits are reported for text labels that cause such discrepancies.

In DRC applications, open and short circuits are reported in the run transcript and the [DRC Summary Report](#), if specified. Discrepancies are reported like this:

```
Open circuit. Label A2 used at location (-25,30) and ignored at location (5,30).  
Short circuit. Label A2 at location (-25,30) used. Label B at location (-20,30) ignored.
```

In LVS circuit extraction, open and short circuits are reported in the run transcript and circuit extraction report like this:

```
WARNING: Open circuit - Same name on different nets:  
Name: "A2"  
(1) at location (-25,30) on layer 10 on net id 1  
(2) at location (5,30) on layer 10 on net id 4  
WARNING: Short circuit - Different names on one net:  
Net Id: 1  
(1) name "A2" at location (-25,30) on layer 10  
(2) name "B" at location (-20,30) on layer 10  
The name "A2" was assigned to the net.
```

Additionally, these discrepancies are annotated as SPICE comments in an extracted netlist within the subcircuits that the discrepancies occur.

Incremental Connectivity and Antenna Checks

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 chip yield.

This section discusses some details involving incremental connectivity in the context of antenna checks performed using Calibre nmDRC. You can employ these methods in any problem involving incremental connectivity, however. Note that incremental connectivity applies only to DRC.

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  
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  
cm12 = COPY cm11           // at each stage are truly different since  
                          // the rule file compiler combines identical  
cc2 = COPY cc1              // operations.  
cm22 = COPY met2  
cv12 = COPY vial
```

```

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 to a diffusion diode.
rule1 { NET AREA RATIO m1_check cg1 > 300 }

// Second level antenna check:
cdiode2 = cm12 AND diode // Diffusion diodes.
m2_check = NET AREA RATIO cm22 cdiode2 == 0
// Check only m2 not connected to a diffusion diode.
rule2 { NET AREA RATIO m2_check cg2 > 300 }

// Third level antenna check:
cdiode3 = cm13 AND diode // Diffusion diodes.
m3_check = NET AREA RATIO cm33 cdiode3 == 0
// Check only m3 not connected to a diffusion diode.
rule3 { NET AREA RATIO m3_check cg3 > 300 }

```

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 behavior that all Connect operations are executed together as one unit at the beginning of the run. This methodology 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 an inadequate solution because it requires numerous layer copies and connect operations—this consumes much memory space. 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 check each antenna level in 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. (Connectivity on layers derived from node-preserving operations does not cross connectivity zones.)
...
 - n. Execute the remainder of the Connect operations.
- n+1. Execute the layer operations where connectivity requirements are derived from all of the Connect operations.

Hence, with incremental connectivity, the flow for the previous example could be as follows:

Example 9-4. Antenna Checks with Incremental Connectivity

```
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 enabled 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 $i > 0$, are not allowed if that connectivity can only be established by Connect statements after connectivity zone i.

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 must mitigate connectivity conflicts, if any, in rule files covering multiple Calibre 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, [Not] [With Neighbor](#) 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 when using incremental connectivity. All errors resulting from this schema are flagged at compilation time.
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 nmDRC 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 metall1
x = NET metall1 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; starting a new CONNECT zone
```

This removal of the previous two connectivity restrictions is only for Calibre nmDRC 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.

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:

1. Produce layer parameters for all new Connect statements which defined the zone.
2. Execute all new Connect operations which defined the zone, appending new connectivity to existing connectivity. Node annotate all Connect layers at, or prior to, the zone which require annotation. Perform net naming based upon existing connectivity if there are [Not] Net operations in the zone.

3. Produce data for all DRC rule check output operations in the zone.
4. Produce data for all referenced connectivity layer operations in the zone.

Notice the requirement that referenced connectivity layer operations in the connectivity zone must be executed (if they have not been already) while in the zone. This is to capture the connectivity of the zone and to correctly satisfy backward references to the zone later.

Virtual Connections in an Incremental Connect Flow.....	272
Incremental Connectivity and Runtime Efficiency	272
Suppression of Connectivity Extraction Warnings for Incremental Connect.....	273
Disconnect Operation	274

Virtual Connections in an Incremental Connect Flow

By default, virtual connection in Calibre nmDRC-H with DRC Incremental Connect YES is only performed within the last Connect block or within a Connect block where there is a [Not] Net operation between the current Connect block and the subsequent one.

The [Virtual Connect Incremental](#) YES statement causes virtual connections to be applied globally in all Connect zones. This statement does not apply to Calibre nmDRC (flat) since virtual connection is always applied within all Connect blocks in a flat incremental flow.

Incremental Connectivity and Runtime Efficiency

When using incremental connectivity, there are a number of rule file organization practices that offer better runtime performance.

1. Place the layer derivations used for connectivity near the beginning of the rule file. For example, place them immediately after the specification statements that begin with the words Layout and DRC, which are often placed at the head of the rule file.
2. Place the incremental connect sequence in its own section of the rule file immediately following the layer derivations from Step 1. Place *only* rule checks that require connectivity zones (like antenna checks) in this section.
3. Place rule checks that use the global connectivity model, but do not need incremental connect zones, after the incremental connect section of the rule file.
4. Place layer derivations and rule checks that are not needed for incremental connectivity after the rule checks in Step 3.

It is wasteful to establish global (or substantial) connectivity, break that connectivity (using Disconnect), and then establish incremental connectivity later in the rule file. A well-organized rule file can usually avoid Disconnect altogether.

Most modern designs necessitate using incremental connectivity in a multithreaded environment with hyperscaling (-turbo -hyper on the command line). Small design blocks may not require this.

Related Topics

[Incremental Connectivity and Antenna Checks](#)

Suppression of Connectivity Extraction Warnings for Incremental Connect

There are cases where it is desirable to suppress connectivity extraction warnings in incremental connectivity flows. An example could be an antenna check flow where any warnings could be considered false. Also, for certain reasons, warnings in intermediate connectivity zones due to [Not] Net operations may also be considered undesirable.

Calibre nmDRC provides connectivity extraction warning suppression capability through the [DRC Incremental Connect Warning](#) specification statement. This statement has two options, ENABLE and DISABLE, and may be specified multiple times in a rule file to control which connectivity zones report warnings during the run. For example:

```
DRC INCREMENTAL CONNECT YES
//First connectivity zone
// DRC INCREMENTAL CONNECT WARNING ENABLE is set by default.
// Connectivity warnings are output.

CONNECT ...
CONNECT ...
<layer operations>
...

DRC INCREMENTAL CONNECT WARNING DISABLE
// Disable connectivity warnings until enabled again.

CONNECT ...
CONNECT ...
<operations involving NET or NOT NET>
...

DRC INCREMENTAL CONNECT WARNING ENABLE
// Warnings are restored.

CONNECT ...
CONNECT ...
<operations>
```

Disconnect Operation

The Disconnect operation 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.

Judicious sequencing of Connect operations in an incremental connectivity flow, along with careful copying of Connect layers, make the [Disconnect](#) operation rarely useful (see “[Incremental Connectivity and Runtime Efficiency](#)”). It is, however, indispensable in certain advanced DRC checks where any existing connectivity must be completely discarded. This is not true of most antenna checks

Appropriate modifications of the compile-time connectivity verification algorithms for layers in an incremental connect setting are made in the presence of Disconnect operations. For example, connectivity of a layer cannot be verified across any Disconnect operation.

Connectivity Warnings in Calibre nmDRC

Warnings from connectivity extraction in Calibre nmDRC appear in the run transcript and in the DRC Summary Report, if specified.

See “[DRC Connectivity Extraction and Stamp Warnings](#)” in the *SVRF Manual*

LVS Circuit Extraction

Circuit extraction is part of the LVS flow. Circuit extraction occurs before LVS comparison and includes various modules such as connectivity extraction, short isolation, device recognition, netlist writing, ERC checking, and soft connection checking.

Circuit extraction can be done as a separate step or in the same run as LVS comparison. If performed separately, you use the -spice command line option. If you perform circuit extraction in the same run as LVS comparison, then there are two methods:

- Use the -lvs -spice options together with the -hier or -flatten options. The -spice option specifies the pathname of the extracted SPICE netlist.
- Use the -lvs option with the -hier or -flatten option. In this case, your [Layout System](#) must be geometric. The extracted SPICE netlist is written either to the [Mask SVDB Directory](#) as *layout_primary.sp* or to the current working directory as *lvs_report_name.sp*.

If -lvs is specified without the -hier or -flatten option, the SPICE parser creates temporary files in the 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.

Other tasks that are performed during circuit extraction include soft connection checking, [Device Recognition](#), [Electrical Rule Checks](#), [Detection of Soft Connections](#), and [Short Isolation](#).

The circuit extraction run transcript is similar to that produced in a DRC run for modules pertaining to rule file compilation, hierarchical database construction, and rule checking. See “[Session Transcripts](#)” on page 206 for details. The device recognition, ERC, and short isolation modules do not apply to DRC, so those sections of the transcript are unique to LVS circuit extraction. See “[Hierarchical Circuit Extraction Transcript](#)” on page 539 for details.

See “[LVS Best Practices](#)” in the *Calibre Solutions for Physical Verification* manual for rule file setup instructions.

Hyperscaling for LVS Connectivity Extraction	276
Hcells and Circuit Extraction	276
Hierarchy Modification Suppression in LVS Connectivity Extraction	277
Selection of Extracted Netlist Names	278
Extraction of Subcircuits.....	279
Hierarchical Treatment of Net Labels.....	280
Netlisting of Ports and Pins.....	282
Virtual Connect Statements	286

LVS Connectivity Extraction Messages.....	288
Detection of Soft Connections.....	289
Short Isolation	300

Hyperscaling for LVS Connectivity Extraction

Hyperscaling is enabled in Calibre nmLVS-H connectivity extraction through the -hyper command line option. The -hyper option can only be specified with the -turbo option. A minimum of four CPUs is recommended on the local host machine when running with -hyper.

```
calibre -spice layout.net -turbo -hyper rules
calibre -spice layout.net -turbo -remotefile config.txt -hyper rules
```

This would also apply during LVS connectivity extraction that occurs in a -lvs -hier run (but not -spice) where your [Layout System](#) is geometric.

Performance improvements in connectivity extraction by using -hyper are significant. Savings up to 25 percent in run time are possible over a non-hyperscaling run.

Only layer operations required for connectivity extraction participate in hyperscaling. These operations appear in the CONNECTIVITY EXTRACTION TOTAL LAYER PREPARATION TIME section of the transcript.

The Short Isolation, Device Layer Preparation, Device Extraction, and ERC modules, which are all a part of connectivity extraction, benefit from hyperscaling.

If LVS comparison is run together with connectivity extraction and hyperscaling is enabled with -hyper cmp, then the comparison module performs certain tasks like reading in the source netlist during the extraction module. If running in MTflex mode, then “-hyper cmp remote” is often desirable.

If -hyper pathchk is used, then a connectivity extraction run uses concurrency of [Pathchk](#) operations in HDBs. This option is useful when Pathchk operations consume much time. Using this option comes at the cost of additional memory usage.

Hcells and Circuit Extraction

Hierarchical circuit extraction (as with the -spice option) does not require an hcell list. Extraction is performed hierarchically, based on the original database hierarchy. Some cells may be expanded or flattened based upon internal heuristics that determine when it is more efficient to do so.

You may provide an hcell list for circuit extraction if you choose. See “[Hcells](#)” on page 474 for details about hcell lists. Note the -automatch option has no effect during circuit extraction.

Hcells specified in an Hcell statement, or using the -hcell or -genhcells command line options, are generally preserved and netlisted during Calibre nmLVS-H circuit extraction. Hcells with no devices are extracted as primitive subcircuits but are not called in the netlist. Note that hcell specification may slow down circuit extraction in cases where it would be beneficial to expand those cells (such as in dense overlap, for example). You can control the automatic expansion of high-cost hcells using the [LVS Auto Expand Hcells](#) statement in the rule file.

Using hcells for connectivity extraction has a performance cost because hcells circumvent certain layout management optimizations within the Calibre hierarchical database constructor. If you are using an hcell list for circuit extraction, then, by default, the run transcript reports high-cost hcells in a section like this:

```
HIGH-COST HCELLS
blk1 (EXPANDING DENSE OVERLAPS) : 73.8
route2 (EXPANDING DENSE OVERLAPS) : 70.8
blk2 (EXPANDING DENSE OVERLAPS) : 51.2
seg (EXPANDING DENSE OVERLAPS) : 25.7
```

The report shows the cell names and the cost of maintaining the cells as hcells. The cost is on a scale from 0 to 100, with 100 being the highest cost. The hierarchy optimization for which the cost is estimated is shown in parentheses (these module names are for internal use and not described). If there are cells with a high cost number, you may want to either expand them through LVS Auto Expand Hcells settings, or exclude them from your hcell list. [LVS Exclude Hcell](#) can be useful in the latter case.

Hierarchy Modification Suppression in LVS Connectivity Extraction

The Layout Preserve Cell List statement allows you to protect certain layout cells from most hierarchy modifications, including automatic expansion, during connectivity extraction. Such preserved cells are treated as if they were hcells during connectivity extraction but are not used as hcells during LVS comparison.

You use this functionality if you want to suppress hierarchy modification for specific cells during connectivity extraction, but not necessarily during circuit comparison. Here is an example:

```
LAYOUT PRESERVE CELL LIST plist // name the list of cells to be preserved
LAYOUT CELL LIST plist "nm$$*" "pm$$*" // the list is defined here
```

The [Layout Preserve Cell List](#) statement works in conjunction with the [Layout Cell List](#) statement. The cells specified in the Layout Cell List statement are exempt from hierarchy modification during circuit extraction.

This functionality has no interaction with hcell specifications, either through the [Hcell](#) specification statement or the -hcell command line option. However, specification of hcells

suppresses hierarchy modification in both circuit extraction and comparison phases. As with hcells, specifying preserved cells can have an adverse effect on performance.

Selection of Extracted Netlist Names

Netlists extracted from the layout have a SPICE-like format.

Instance, net, and subcircuit names appear in the extracted netlist using SPICE conventions. SPICE format introduces limitations for characters that may be used in such names. In general, you may not use the following in layout object names that are netlisted:

- white space (space, tab, newline, carriage return, and so forth)
- equals sign (=)
- comma (,)
- dollar sign (\$) at beginning of a name

Slash characters (/) in top-level port names cause warnings during circuit extraction. While the slash character is allowed by the connectivity extractor for net and port names, such names are not netlisted, but numbers are used instead.

Note

 Slash characters in names are problematic in SPICE netlists. See “[Characters Allowed](#)” on page 635 for a complete discussion of characters allowed in SPICE.

Extracted layout names using the Calibre nmLVS-H netlister (calibre -spice) appear as follows:

- Cell names are netlisted according to how they appear in the layout.
- Net names are netlisted according to how they appear in the layout. The [Layout Preserve Case](#) specification statement controls the text case of net names that appear in the netlist. See “[Hierarchical Treatment of Net Labels](#)” on page 280 for related information.
- Port names are netlisted according to how they appear in the layout.
- Built-in device names are netlisted in uppercase.
- User-defined device component names are netlisted as they appear in the [Device](#) statements. Such devices appear in primitive .SUBCKT definitions.
- Pin names for devices that result in primitive .SUBCKT definitions are netlisted as they appear in the Device statements.
- Model names are netlisted as they appear in the Device statements.
- Built-in device properties that are calculated internally appear in uppercase.

- Device properties that are calculated using a property computation program appear in lowercase. If Layout Preserve PROPERTY Case YES is specified, then user-defined properties appear as they are written in the rule file PROPERTY statements in the property computation programs.

The [Layout Case](#) statement affects the netlisting of duplicate user-defined devices during generation of primitive .SUBCKT definitions. When Layout Case YES is specified, then two Device statements having the same user-defined element name and the same number of pins, but differing in case, generate two distinct .SUBCKT definitions in the extracted netlist. The cases of the element names are preserved. When NO is specified, only one .SUBCKT definition is generated, and the case of the element name is chosen arbitrarily. The Layout Case statement has no other effect during connectivity extraction.

For information regarding case-sensitive LVS comparison of SPICE netlists, see “[Case-Sensitive Handling of Names](#)” on page 375.

Related Topics

[SPICE Format](#)

Extraction of Subcircuits

When the hierarchical circuit extractor is used, unexpanded layout cells are extracted as SPICE subcircuits. If an extracted subcircuit contains no devices anywhere in its hierarchy, it is not called in the extracted netlist.

Note

 SPICE format introduces limitations for characters that may be used in layout cell names. See “[Selection of Extracted Netlist Names](#)” on page 278 for a complete discussion of characters allowed in SPICE.

In certain cases, cells containing devices can be expanded by the hierarchical database constructor. This behavior is overridden by using either an hcell list, the [Hcell](#) statement, or the [Layout Preserve Cell List](#) statement. (The [LVS Box](#) statement also overrides cell expansion, but it has the additional effect of causing LVS comparison to stop at the input pins of a box cell.) Note that specifying hcells or preserving cells simply to maintain the original layout hierarchy can have negative effects on performance.

ICV Cell Creation

To obtain better performance, the hierarchical database constructor can introduce pseudohierarchy into the Calibre database. Pseudocells in this modified hierarchy are netlisted as subcircuits with the prefix ICV_. Pseudocells have been altered from the original layout during the run. If warnings are issued regarding these pseudocells, then the warnings match the subcircuit names used in the extracted SPICE netlist.

Some ICV_ cells are referred to as FAUX BIN in the run transcript. These pseudocells can be traced back to original layout cells. Any circuit extraction warnings for FAUX BIN cells are referenced back to the original cell names to facilitate debugging.

For LVS comparison discrepancies involving ICV_ cells, you can use Calibre RVE to trace their devices in layout and source.

Device and Cell Name Conflicts

If the rule file contains a Device definition for a non-standard SPICE device whose name conflicts with a design cell name, then the cell name is changed in the extracted subcircuit definition to a unique name starting with a sequence of underscore (_) characters. This warning is given:

WARNING: LAYOUT CELL <name> conflicts with a DEVICE with the same name and was renamed to __<name> in the layout netlist.

Hierarchical Treatment of Net Labels

If a net is named (texted) in the layout, then that name usually appears in the extracted 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 certain rules:

- 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 metacharacters and must not consist solely of digits.

The latter restriction prevents collisions with internal net numbers. Every layout net receives a internal net number; naming of nets using connectivity text (see [Text Layer](#)) is optional but desirable.

Net labels in hierarchical netlists receive the following treatment:

- 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.
- Labels from a cell are used locally in the cell to name nets in the cell. (You can have Calibre nmDRC mimic this behavior by specifying [DRC Cell Text YES](#).)
- 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 represent the connections properly 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).

Connectivity Text in ICV_* Cells

The hierarchical connectivity extractor preserves connectivity text in some “ICV_*” cells. ICV_* cells are referred to as pseudocells and are created by the tool to optimize the Calibre internal database hierarchy. ICV_* cells with preserved connectivity text are referred to as FAUX BIN cells in the run transcript. The text is preserved from such cells, and warnings related to the text are reported using the original cell names. The only user-visible consequences of this are as follows:

- The extracted SPICE netlist may contain net names in some cells whose names start with ICV. Note, however, that LVS comparison discards user-given names in such cells, so this has no effect on the results of LVS comparison.
- The PHDB contains the same net names and these names can be accessed using Calibre RVE or the Query Server.

Netlisting of Ports and Pins

A port is an entry point to a cell and forms the external interface of the cell. A port becomes an instance pin (or simply, a pin) when the cell is placed in the layout hierarchy and the port makes a connection outside the cell. In LVS and Calibre xRC applications, text and polygon objects on port layers can be read and netlisted as ports.

Each port object is represented in the connectivity extractor by an (x,y) location, a layer, and an optional port name. In LVS, you can specify ports for the top-level cell using [Port Layer Text](#), [Port Layer Polygon](#), or [Port Layer Merged Polygon](#) specification statements in the rule file.

These statements are not used to label pins of lower-level cells; instead, net IDs are used for this purpose.

Port objects appear in the hierarchical SPICE netlist as ports 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.

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 port text object, respectively. The process of assigning label names to port objects is based on the label location and layer. The allowed attachment methods for Port Layer Text objects are the same as for net names; that is explicit, implicit, and free attachment are all supported. See “[Net Label Attachment](#)” on page 261 for methods of attaching text labels.

The [Port Layer Polygon](#) and [Port Layer Merged Polygon](#) specification statements support shapes that identify port objects. The port layer is a specified layer, and the port location is the center of a port polygon's extent. If the center of the extent is not on the polygon itself, then a point on the polygon is chosen by the tool. Port polygon objects are attached to geometric objects in the layout. The assigned net names of such geometric objects become the names of the respective ports. If no user-given net name is assigned to the geometric object, then a

system-generated number is used. The methods for attaching port shapes to geometry in the layout are the same as for port text objects, as discussed previously.

Text objects and shapes defined in the rule file with the [Text](#) and [Polygon](#) specification statements cannot name ports. [Layout Polygon](#) shapes are treated as polygon ports, but they are not merged together or with any other layers.

The depth from which port objects are read is controlled by the [Port Depth](#) statement. The default is from the top level.

Given the preceding ideas, the following definitions of trivial objects are important to understand.

Trivial nets are not connected to any devices, so a trivial net could be removed in the flat view of the design, and it would not make a difference.

A *trivial pin* is attached to a trivial net. Trivial pins are not netlisted by default during circuit extraction. You can get all texted pins to be netlisted by using the [LVS Netlist All Texted Pins YES](#) statement in the rule file. (Such texted pins must be referenced by an X subcircuit call in the netlist in order to appear, however.)

A *trivial port* is a port object attached to a net that does not serve as a pin in its parent cell in any placement of that cell. A net serves as a pin of a cell if the net interacts with geometries outside the cell, or with top-level port objects, or if the cell is an LVS Box cell and it has a named port object attached to it. Otherwise, the net does not serve as a pin, and any port object on that net is trivial. Trivial ports are not netlisted. You can cause trivial layout port objects to be reported by specifying [LVS Report Trivial Ports YES](#).

For a discussion of trivial netlist objects in LVS comparison, see “[Trivial Ports, Pins, and Nets](#)” on page 438.

Hierarchical Processing of Port Text and Polygon Objects.....	283
Unattached Ports	285
Propagation of Power and Ground Names from Lower-Level Ports	285
Omission of Floating Pins	285

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 on the .SUBCKT line by the SPICE netlister and thus participate in LVS comparison. This behavior is consistent with flat LVS. Port objects at lower levels of hierarchy are not output by the SPICE netlister and do not specify cell pins. Net names are used for pin names at lower levels of the hierarchy.

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.

For the top-level cell, the hierarchical SPICE netlister outputs ports in the pin list of the top-level .SUBCKT statement. The .SUBCKT port names are the same as the respective net names within the subcircuit.

Ports contributed by Port Layer Text, Port Layer Polygon, and Port Layer Merged Polygon statements can be named or unnamed. The name chosen for netlisting is the port name or the original net name, whichever is present. 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.

If Port Layer Text is used, and if both the port and the associated net are texted, then the original net name prevails in the event of a conflict. 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.

The [LVS Netlist Connect Port Names](#) statement controls whether the circuit extractor (for example, calibre -spice) should connect top-level port names to nets in the top-level cell when the port names (from Port Layer Text) differ from any net name in the cell. If LVS Netlist Connect Port Names YES is specified, then port text names can override layout net names that would be used for port names by default.

If Port Layer Polygon or Port Layer Merged Polygon is used, and if the associated net is not texted, then a system-generated number is used for the port name. Otherwise, the associated net name becomes the name of the port.

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 nmDRC application.
- These statements are not used in 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 “[Unattached Ports](#).”

Unattached Ports

Unattached ports occur in the layout when the port layer does not appear in Connect, Attach, or Label Order statements; or there is no geometry that the port can be attached to at the port location.

If there are unattached ports in the layout, the hierarchical connectivity extractor indicates this in the run transcript and the circuit extraction report as follows:

WARNING: Unattached port pads; port ignored.

Such unattached ports are not processed by LVS comparison because a bare wire cannot be represented in a SPICE netlist. Unattached ports might not cause an INCORRECT result.

Propagation of Power and Ground Names from Lower-Level Ports

In Calibre nmLVS-H, you can propagate power and ground net names throughout the hierarchy from lower-level ports by specifying LVS Cell Supply YES.

[LVS Cell Supply](#) YES can be useful when top-level nets do not contain power and ground net names but lower-level cells do. When you specify this, port names on cells that match the [LVS Power Name](#) and [LVS Ground Name](#) settings propagate power and ground signals through the hierarchy into attached nets, so that the entire net, considered flat, becomes a power or ground supply.

Omission of Floating Pins

A *floating pin* is unconnected to any other instance pin or to a pin of a parent subcircuit. If a subcircuit in the extracted netlist has a floating pin in all instances of this subcircuit, that is, the pin is never connected to any other instance or to a pin of the parent subcircuit, this pin can be omitted from the netlist.

Whether or not a pin is omitted is affected by the rule file statements that use connectivity information, such as Mask SVDB Directory, ERC Pathchk, Pathchk, and connectivity-dependent layer operations like Net, Net Area Ratio, Pathchk, Stamp, and so forth.

Note that only the netlist is affected: the PHDB created by Calibre tools for use with Calibre RVE and Query Server always has all connectivity information for every geometry

written into the PHDB, including all pins needed to connect geometries into nets across cell boundaries.

Floating pins can be reported during circuit extraction by using the [LVS Report Trivial Ports](#) YES specification statement if they are identified as port objects.

Virtual Connect Statements

The *virtual connect* technique in layout verification uses 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 processing.

Virtual connectivity is triggered by the rule file Virtual Connect family of specification statements. You can instruct Calibre to report virtual connections using the [Virtual Connect Report](#) YES specification statement, which is a recommended practice.

Virtual connectivity is of primary interest in LVS applications. While it can be used in DRC applications, virtual connectivity is not applied by default when using [DRC Incremental Connect](#) YES in a hierarchical DRC run (but can be turned on with [Virtual Connect Incremental](#) YES). Flat DRC always applies virtual connectivity when virtual connections are defined.

Virtual connections are made based upon case-insensitive net name matching by default. Matching can be made case-sensitive through the [Layout Preserve Case](#) YES statement.

Virtual Connect Colon

[Virtual Connect Colon](#) is used to virtually connect nets that share a common prefix before a colon, like VDD:1, VDD:2, and so forth.

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 that originally contained a 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 as colons (default is YES).

You can enter labels with rule file Text statements or geometric database text as described in section [“Net Name Specification”](#) on page 258. A virtual connection between labels causes a virtual connection between the net segments to which those labels are assigned, regardless of whether 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 Virtual Connect Box Colon YES does not strip off colon suffixes from node names in .GLOBAL statements in SPICE netlists.

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 when YES is specified:

```
.GLOBAL VCC:P VSS:XYZ VDD:  
.GLOBAL VCC VSS VDD
```

If you specify NO, or the statement is not specified, 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

[Virtual Connect Name](#) virtually connects nets that share the same name. Each *name* argument is a 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 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.

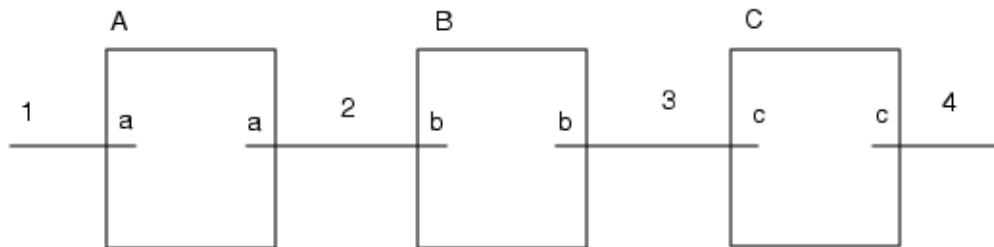
For LVS Box cells, the [Virtual Connect Box Name](#) specification statement works similarly to Virtual Connect Name.

Example 9-5. Virtual Connection of Named Nets for LVS Box Cells

1. VIRTUAL CONNECT BOX NAME “?”

This connects net segments with identical names in LVS Box cells. It also connects together respective pins. In [Figure 9-7](#), 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 9-7. Example of Virtual Connect Box Name



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. (It does not connect VCC to VSS.) Virtual name connection does not apply to the LVS Box BLACK keyword option.

LVS Connectivity Extraction Messages

Connectivity extraction messages are issued to the run transcript and the circuit extraction report.

The errors, warnings and notes reported by connectivity extraction are listed under “[Circuit Extraction Messages](#)” in the *SVRF Manual*. The netlist extraction module reports such messages in the session transcript and in the circuit extraction report file.

Detection of Soft Connections

Conflicts in Sconnect connectivity are reported in the circuit extraction transcript and the circuit extraction report. In addition, SPICE comments are written into the extracted SPICE netlist in subcircuits where Sconnect conflicts occur.

LVS Report Option S turns on detailed reporting of Sconnect conflicts in the session transcript and in the circuit extraction report. This option is ignored in DRC applications. Report option EC causes Sconnect conflicts to be reported as errors in LVS comparison.

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

In LVS circuit extraction, you can find polygons from **Sconnect** layers that are involved in conflicting uni-directional connections by using the **LVS Softchk** statement. Depending on the options you specify, you can output upper layer, lower layer, contact layer, or all layers involved in conflicting soft connections to a lower layer.

Sconnect stamping conflicts (soft connections) can also be found based upon the contents of a **Mask SVDB Directory** produced during a circuit extraction run. One way to do this is to use Calibre nmLVS Reconnaissance Softchk. This type of run is initiated by using the `calibre -recon -softchk` command line. You can also use the `qs::softchk` or `qs::softchks` commands, which can be called in the Tcl script specified by the `-softchk=tcl_script` option.

LVS Softchk selects the upper-layer net with the highest vertex count as the prevailing net in a stamping conflict with a lower-level polygon. Polygons on conflicting nets containing fewer vertices are reported as errors. See “[Report Soft Connections Using Contact Counts](#)” to use contact counts to select the prevailing net in a stamping conflict.

The output from LVS Softchk is an ASCII results database similar to the DRC or ERC results database. This is viewable in Calibre RVE. You can control the results count in the database with the **LVS Softchk Maximum Results** specification statement.

LVS Softchk statements are executed after ERC rule checks and **ERC Pathchk** statements 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; therefore, ERC checks are done before LVS Softchk.

LVS Softchk Debug Method	290
LVS Softchk Results Database File Format	293
Report Soft Connections Using Contact Counts	294
lvs::softchk	298

LVS Softchk Debug Method

When running LVS Softchk, sometimes the number of results can be overwhelming to debug. In order to fix soft connection problems efficiently, it is best to focus on fixing the central problem, and then divide the remaining problems up into more manageable pieces.

While the material in this section is written from the standpoint of LVS Softchk, it also applies to Calibre nmLVS Reconnaissance Softchk (calibre -recon -softchk), qs::softchk, and qs::softchks results, with appropriate accommodation of the syntax elements that map to each other in the various usages.

A primary problem that [LVS Softchk](#) detects is a connection of differing nets through a well polygon, although there are other applications. The ability to detect this is based upon the [Sconnect](#) statement, which has two forms:

```
SCONNECT upper_layer lower_layer
```

or

```
SCONNECT upper_layer lower_layer BY contact_layer
```

The lower_layer is often a well layer. The upper_layer is often a tap (diffusion) layer or a metal layer. The contact_layer is usually a contact layer, or a tap layer if metal has been used for an upper_layer.

From this point forward, assume you are debugging problems with a lower_layer called pwell and another layer called ptap. Bad pwell polygons are ones that contain shorts between two or more nets.

To find bad pwell polygons, this statement is used in the rule file:

```
LVS SOFTCHK pwell LOWER
```

The LOWER keyword is optional. If this keyword is not in the rule file, it is helpful to add it for clarity.

Bad pwells appear in the softchk results database in this form:

```
SOFTCHK pwell.
```

This check name flags all the bad pwell polygons. Once the bad pwell polygons are known, the immediate goal becomes resolving the bad connections to these polygons.

To find bad taps, this statement is used in the rule file:

```
LVS SOFTCHK pwell UPPER
```

If ptap polygons have been specified as an upper_layer in an Sconnect operation, then bad ptap polygons are found in the softchk results database in a check name of this form:

```
SOFTCHK ptap UPPER.
```

If ptap polygons are specified as a contact_layer, then you can specify this:

```
LVS SOFTCHK pwell CONTACT
```

and bad ptap polygons are found in a check name of this form:

```
SOFTCHK ptap CONTACT.
```

This also works for contact polygons that are not necessarily well taps.

If these types of check names do not appear in your softchk database, you should consider adding the appropriate LVS Softchk statement to your rule file to generate the corresponding results.

By highlighting the appropriate SOFTCHK results discussed in the preceding paragraphs, you can see both the bad pwell polygons and the ptap polygons that cause the well stamping problems. Note that only the ptap polygons that were *not chosen* to stamp the pwell polygons are highlighted. The ptap polygons that LVS chooses to stamp the pwells are not output by default. If you want to see UPPER or CONTACT polygons from all the nets involved in the stamping conflict, use the ALL option for LVS Softchk.

For each of the bad pwell polygons, look for the corresponding ptap polygon that is bad. Try to determine why the ptap polygon is not on the same node as the pwell it touches. Is there an open or short circuit somewhere? Is there a bad connection to a tap? Is there a text problem? In some cases, the problem will be obvious. In other cases, the issue may be more subtle.

To assist in debugging more complicated cases, you can use ERC rule checks and some layer derivations. You can add the following SVRF code to your rule file and run LVS circuit extraction again.

To derive all bad pwell polygons, you can use this definition:

```
bad_pwell = pwell INTERACT ptap != 1 BY NET
```

This finds all pwells that do not interact with ptap polygons that are on only one net. The bad_pwell layer contains both shorted and floating pwell polygons.

You can then derive all ptaps in bad pwells like this:

```
ptap_in_bad_pwell = ptap INTERACT bad_pwell
```

Assume that all pwells are supposed to be on the VSS node. Use these layer derivations:

```
bad_pwell_vss = bad_pwell NET "VSS" // bad pwell that is vss
// ptap that is vss interacting with bad vss pwell
bad_ptap_vss = ptap_in_bad_pwell NET "VSS"
```

These checks show bad ptaps and pwells:

```
bad_ptap_other {
@ ptap is not VSS
ptap_in_bad_pwell NOT bad_ptap_vss
}

bad_pwell_other {
@ pwell is not VSS
@ also show all taps in these wells
// either the pwell is stamped by some net other than vss,
// or it is floating
bad_pwell_other = bad_pwell NOT bad_pwell_vss
// show all taps in the bad well
ptap_in_bad_pwell INTERACT bad_pwell_other
COPY bad_pwell_other
}

bad_pwell_vss {
@ pwell is vss and shorted to some other net
@ also show non-vss taps in these wells
COPY bad_pwell_vss
(ptap_in_bad_pwell NOT bad_ptap_vss) INTERACT bad_pwell_vss

LVS EXECUTE ERC YES // default
LVS ABORT ON ERC ERROR YES // default
GROUP pwell_checks bad_p?
ERC SELECT CHECK ABORT LVS pwell_checks
ERC SUMMARY REPORT erc.report
ERC RESULTS DATABASE erc.db // results database for Calibre RVE
ERC CELL NAME YES CELL SPACE // present results in subcell context
whenever possible
```

These layer derivations and checks can be expanded upon if there is more than one allowed ground net (but only one per well).

Related Topics

[Soft Connection Checks](#)

LVS Softchk Results Database File Format

Output for: [LVS Softchk](#) specification statement, `qs::softchk`, and `qs::softchks`.

This file is output when soft connection checking is requested.

Format

The format is essentially the same as for a DRC or ERC results database. See “[ASCII DRC Results Database Format](#)” on page 236. The LVS Softchk filename is of the form *layout_primary.softchk*, where *layout_primary* is the name of the top-level cell. The Query Server `qs::softchk` and `qs::softchks` commands output a file called *softchk.rdb*.

A softchk results database has additional lines to cross-reference a listed polygon to its net and its [Sconnect](#) warning. Each cross-reference is comprised of two comment lines that show the selected net and the rejected net(s):

```
Net <net> is selected for stamping.  
Rejected nets: <net names>
```

This format is borrowed from the SOFTCHK warnings in the circuit extraction report. Multiple comment line pairs may appear for a soft connection check.

The following is contained in each result:

Property	Definition
DN <i>net_name</i>	The net name of a polygon from the designated layer (LOWER, UPPER, CONTACT).

Parameters

None.

Examples

This is an example of a soft connection result associated with two Sconnect warnings. This listing shows that polygon 14 comes from rejected net E, which appears in the first set of comment lines. Polygon 109 is on net VSS and appears in both sets of comment lines. By default, only rejected polygons are listed. Hence, polygon 109 unambiguously belongs to net VSS from the first set of comment lines.

```
top 1000
SOFTCHK psub CONTACT.
1 1 2 Nov 12 10:23:33 2020
Net 3 is selected for stamping.
Rejected nets: VSS D E 1
Net VSS is selected for stamping.
Rejected nets: AGND
p 1 4
DN 1
-9280 -8460
-8920 -8460
...
p 14 4
DN E
-9280 32245
-8920 32245
...
p 15 4
DN AGND
-9280 32905
-8920 32905
...
p 109 4
DN VSS
9475 31460
9835 31460
...
```

If there are multiple comment line pairs listed for a single check, it may not be possible to determine a unique warning for each polygon.

Report Soft Connections Using Contact Counts

Soft connections can be determined by contact count as opposed to vertex count.

As discussed under “[Detection of Soft Connections](#),” the **LVS Softchk** statement decides which polygons are error polygons based on the *vertex count* of the conflicting upper-layer nets.

The TVF compile-time **lvs::softchk** command determines which net gets reported in a stamping conflict by counting the *number of discrete contacts* shared with a lower-layer polygon. *Shared contacts* in this context means there is a mutual intersection among the upper, lower, and contact layers. The net with the greatest number of shared contacts is chosen as the intended stamping net. All contacts on other nets involved in the stamping conflict are considered error polygons.

The contact count method handles conflicts between two or more large nets more accurately, since vertex count of each net does not necessarily reflect the design intent. Counting the number of contacts shared with a lower-level polygon reveals which net is probably intended.

The CalibreDFM_SOFTCHK Tcl package contains lvs::softchk. Here is an example:

Example 9-6. Soft Connection Check Using Contact Counts

```
#!tvf
tvf::VERBATIM {
...
// any SVRF code
...
}

# contact-based soft-connect checks

package require CalibreDFM_SOFTCHK
lvs::softchk -upper_layer tpdiff -lower_layer psub -contact_layer pplug
lvs::softchk -upper_layer tndiff -lower_layer nxwell -contact_layer nplug
...
```

Using the lvs::softchk function requires that you specify an [ERC Results Database](#) statement; otherwise, the following runtime error occurs:

```
ERROR: Cannot determine the ERC RESULTS DATABASE from rule file
<filename>.
```

The ERC Results Database statement is needed because a different database from the LVS Softchk database is used for storing results. You do not need any [ERC Select Check](#) statements specified when lvs::softchk is used.

The [lvs::softchk](#) results output to the specified database is like typical ERC results and appears as follows:

```
LVS_SOFTCHK_1_SOFTCHK
1 1 2 Jul 28 23:48:15 2009
Rule File Pathname: rules
lvs::SOFTCHK tpdiff psub BY pplug
p 1 4
5570 2445
5930 2445
5930 2775
5570 2775
```

The results have the following check name and check text comment formats:

```
LVS_SOFTCHK_<N>_SOFTCHK
lvs::SOFTCHK <upper_layer> <lower_layer> BY <contact_layer>
```

The check text comment shows the names of the upper layer, lower layer, and contact layer. The result record contains only the contact polygons from nets involved in the stamping conflict, where the nets had a lesser number of contacts than the prevailing net.

An ASCII DFM RDB database is always produced by lvs::softchk and is called *softchk.rdb* by default. This database contains additional debugging information like net IDs of the contacts and the lower-layer polygons. Results appear as follows:

```
lvs::SOFTCHK tpdiff psub BY pplug (CONTACT)
1 1 2 Jul 29 00:14:06 2009
lvs::SOFTCHK tpdiff psub BY pplug (minority contacts on conflicting soft
connections)
IL: LVS_SOFTCHK__2__SOFTCHK
p 1 4
CN sconn_test 1 0 0 1 0 0 1
CONTACT_NETID 2 VSS
MAJORITY_NETID 1 AGND
5570 2445
5930 2445
5930 2775
5570 2775

lvs::SOFTCHK tpdiff psub BY pplug (FLAGGED)
1 1 2 Jul 29 00:14:07 2009
lvs::SOFTCHK tpdiff psub BY pplug (lower polygons not soft connected by
majority contact)
IL: LVS_SOFTCHK__2__FLAGGED
p 1 4
CN sconn_test 1 0 0 1 0 0 1
SCONNECT_NETID 2 VSS
MAJORITY_NETID 1 AGND
-10180 -9240
6915 -9240
6915 7430
-10180 7430
```

There are two categories of comments and rule names in the database:

```
lvs::SOFTCHK <upper_layer> <lower_layer> BY <contact_layer> (CONTACT)
IL: LVS_SOFTCHK__<N>__SOFTCHK
```

and

```
lvs::SOFTCHK <upper_layer> <lower_layer> BY <contact_layer> (FLAGGED)
IL: LVS_SOFTCHK__<N>__SOFTCHK__FLAGGED
```

The CONTACT record contains only the contact polygons from nets involved in the stamping conflict that were not on the prevailing net. Each contact polygon is annotated with a CONTACT_NETID property. The property value is the contact's net ID as assigned by the connectivity extractor. Each contact also receives a MAJORITY_NETID property. The MAJORITY_NETID is the net ID of the prevailing net (based on shared contact count) in the stamping conflict. This net ID is assigned to the lower-layer polygon.

The FLAGGED check may or may not be empty. This check contains lower-layer polygons only if Sconnect did not stamp the lower-level polygon by the net ID that matches the MAJORITY_NETID property value (recall that Sconnect chooses the stamping net based upon

vertex count, not contact count). If any FLAGGED checks are non-empty, you should not proceed with LVS comparison until the net stamping conflicts are fixed.

The output databases are both viewable in Calibre RVE as DRC/ERC databases.

lvs::softchk

CalibreDFM_SOFTCHK package Tcl command.

Used for soft-connection reporting based upon shared contact counts.

Usage

```
lvs::softchk -lower_layer layer_name -upper_layer layer_name
[-checkname "name"]
[-comment "string"]
[-contact_layer layer_name]
[-softchk_rdb filename]
[-debug]
[-help | ?]
```

Arguments

- **-lower_layer *layer_name***

Required parameter set that specifies the name of a lower layer in an [Sconnect](#) statement.

- **-upper_layer *layer_name***

Required parameter set that specifies the name of an upper layer in an Sconnect statement. This layer must also appear in a [Connect](#) statement, but not as a contact (BY) layer.

- **-checkname "name"**

Optional parameter set that specifies the name of the rule check to use for the results databases. By default, a name is chosen by the tool as shown in the section "[Report Soft Connections Using Contact Counts](#)" on page 294.

- **-comment "string"**

Optional parameter set that specifies a user comment to annotate the results.

- **-contact_layer *layer_name***

Optional parameter set that specifies the name of a contact (BY) layer in an Sconnect statement.

- **-softchk_rdb *filename***

Optional parameter set that specifies the name of the output ASCII DFM RDB database. The default database name is *softchk.rdb*.

- **-debug**

Optional parameter that specifies to output debugging layers to the ASCII DFM RDB database.

- **-help | ?**

Optional parameters that print a usage message.

Description

Specifies to perform soft-connection reporting based upon contact counts of nets rather than vertex counts, which is what [LVS Softchk](#) uses for reporting. This is a TVF function and must be used in a compile-time TVF rule file. The function name and parameter names are case-sensitive. The *layer_name* and *filename* parameters are not.

This function determines which net prevails after an Sconnect stamping conflict is detected. When **-contact_layer** is specified, the net that contains the most discrete contacts that are shared between **-upper_layer** and **-lower_layer** polygons is chosen as the prevailing net. Contacts from conflicting nets that are not selected as the prevailing net are output. These are contacts that intersect **-lower_layer** polygons.

If **-contact_layer** is not specified, the function performs the analysis assuming a direct connection between **-lower_layer** and **-upper_layer** polygons and reports violations at the intersections of the two layers.

Results output is to an [ERC Results Database](#) file (this statement must be specified when `lvs::softchk` is used) and to an ASCII DFM RDB named `softchk.rdb` by default. The latter database name can be changed using the **-softchk_rdb** parameter. Results are polygons from nets involved in a stamping conflict with **-lower_layer** polygons, where the nets are not the prevailing nets. The output formats are discussed under “[Report Soft Connections Using Contact Counts](#).”

When the **-debug** option is used, the ASCII DFM RDB database contains additional information that can be used for debugging. The additional rule checks and comments are preceded by the word DEBUG in the database. [Table 9-3](#) shows the DEBUG check annotations and what they mean.

Table 9-3. DEBUG Results Annotations

Check Annotation Name	Description
CONTACT	All contacts shared between upper layer and lower layer polygons.
CONTACT_CONFLICT	All shared contacts intersecting a lower layer polygon that is involved in a stamping conflict.
CONTACT_NETID	Similar geometrically to CONTACT_CONFLICT but also includes contact net ID properties.
LOWER	All lower layer polygons.
LOWER_CONFLICT	All lower layer polygons involved in a stamping conflict.
LOWER_NETID	Similar to LOWER_CONFLICT but also includes the assigned net ID property based upon net vertex count.
LOWER_STAMPED	Similar to LOWER_CONFLICT but also includes the assigned net ID property based upon net shared contact count.

There is no consistency check to ensure that the arguments to lvs::softchk match an Sconnect statement. The lvs::softchk function assumes that the input layers are properly specified and only considers their existing connectivity.

Layer shielding, such as with Connect or Sconnect statements having more than two interconnect layers, is not supported. So, lvs::softchk does not provide a way to specify this:

```
SCONNECT w x y BY z
```

where polygons on x can shield polygons on y from receiving connectivity from polygons on w.

You may only specify the equivalent of this:

```
SCONNECT a b BY c
```

where no shielding can occur.

Examples

```
#!tvf
tvf::VERBATIM {
// SVRF code
ERC RESULTS DATABASE erc.db
ERC SUMMARY REPORT erc.report HIER
LVS EXECUTE ERC YES
...
}
...
# Tcl code
package require CalibreDFM_SOFTCHK
lvs::softchk -upper_layer ntap -lower_layer nwell -contact_layer cont
lvs::softchk -upper_layer ptap -lower_layer pwell -contact_layer cont
```

Related Topics

[Detection of Soft Connections](#)

Short Isolation

A text short occurs when a layout net has two different connectivity text objects associated with it. Such shorts can lead to problems with net name assignment, which impacts things like ERC checks and LVS comparison results. Short isolation is the process of identifying and fixing text short discrepancies.

Text shorts are always reported in the run transcript when circuit extraction is performed. They also appear in a circuit extraction report when one is produced. Most importantly, Calibre offers three flows for finding shorts. The first is implemented through the LVS Isolate Shorts specification statement. The second is implemented through the Mask SVDB Directory SI option, and the third is implemented through the -recon -si command line options. The latter two

methods support interactive short isolation in the Calibre Query Server Tcl shell. All flows support Calibre RVE short debugging.

LVS Isolate Shorts Flow

To simplify the process of isolating shorts, you can specify this in the LVS rules:

```
LVS ISOLATE SHORTS YES BY LAYER BY CELL
```

This statement isolates a short by finding a polygonal path between two or more conflicting text objects. The shorts are output to an ASCII results database similar to the ones produced in DRC runs.

Note

 Short isolation occurs only at the top level by default. If the default is used and no shorts occur at the top level, then no shorts database is produced. However, text shorts from all levels of the hierarchy always appear in the circuit extraction report.

You should always use the BY LAYER keyword because it enables easier debugging. It organizes results per-short, per-layer.

Specifying BY CELL with BY LAYER causes the results to be written on a per-short, per-layer, per-cell basis, which can be very useful in debugging more complex shorts involving global signals or supply nets. The cell of origin is reported using the “CN” property in the results database.

The BY CELL and BY LAYER keywords have an optional ALSO keyword. Specifying ALSO causes the results to be written from multiple perspectives.

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 at the level of hierarchy specified by the statement, short isolation algorithm operates hierarchically and shapes are not flattened.
3. Short isolation results are written out to disk in an ASCII database format file, which is viewable using a layout editor and Calibre RVE. The name of this database is *lvs_report_name.shorts*, where *lvs_report_name* is the name of the [LVS Report](#). If no LVS Report is specified, then the database is called *lvs.layout_primary.rep.shorts*.

4. Hierarchical device recognition, hierarchical SPICE netlist generation and, if requested, hierarchical LVS comparison.

To execute short isolation in hierarchical mode, any of these command lines may be used:

```
calibre -spice layout.net -turbo -hyper rules  
calibre -spice layout.net -lvs -hier -hcell cells -turbo -hyper rules  
calibre -lvs -hier -hcell cells -turbo -hyper rules
```

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

```
calibre -lvs -flatten rules  
calibre -lvs -tl layout.cnet rules
```

You can view short isolation results before the run terminates by altering the command line in the following way:

```
calibre -lvs ... >& logfile  
tail -f logfile
```

or

```
grep -i "SHORT ISOLATION" logfile
```

The short isolation module returns the following:

```
SHORT ISOLATION started.  
SHORT ISOLATION completed. CPU TIME = 0 REAL TIME = 0 ...  
SHORT ISOLATION RESULTS DATABASE = lvs.rep.shorts
```

You can then open the short isolation database in Calibre RVE and begin debugging. The short isolation database is much like the DRC ASCII results database with some additional information.

If the run produces no shorts and LVS Isolate Shorts YES is specified, the tool automatically removes an existing short isolation results database.

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](#).

In the shorts database, 2x2 dbu squares are added to the polygonal paths between shorted text objects. The text object base points intersect these squares. This is to ensure text objects in the shorts database are not isolated from short paths. The squares are on the layers that the text objects are attached to, and the squares abut or overlap other shapes in a short path.

In Calibre nmLVS-H, short isolation does not process target layers specified by the `Sconnect` statement unless those layers contain texted 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.

You can cause text shorts to produce LVS comparison discrepancies by specifying [LVS Report Option ES](#).

See “[Invoking Short Isolation](#)” in the *Calibre Interactive User’s Manual* to see how to set up a short isolation run in the interactive mode.

The short isolation (`.shorts`) database can be viewed in Calibre RVE for LVS. You can debug a short, virtually repair the short by marking polygons for removal and assigning net labels within Calibre RVE, and run short verification to verify that the changes fix the short. See the following topics in the *Calibre RVE User’s Manual* for more information: “[Using Calibre RVE for LVS](#)”, “[Interactive Short Isolation](#)”, “[Verifying Short Repairs](#).”

Mask SVDB Directory SI Flow

This flow facilitates short isolation using the Query Server Tcl shell. The `Mask SVDB Directory` SI keyword ensures the SVDB directory has the necessary information to perform interactive short isolation using Query Server Tcl commands. For more information about this, see “[Using Short Database Commands](#)” in the *Calibre Query Server Manual*.

This flow allows you to do short isolation independently from other LVS tasks once the SVDB containing the shorts data exists. See “[Performing Only Short Isolation](#)” in the *Calibre Solutions for Physical Verification Manual*.

Calibre nmLVS Reconnaissance Short Isolation

The Calibre nmLVS Reconnaissance Short Isolation flow (or *Calibre nmLVS Recon SI* for short) supports stand-alone short isolation. This flow is used outside of LVS and is well suited toward fixing shorts early in the design cycle to avoid LVS problems later.

This flow uses the calibre `-recon -si` command line. This is an abbreviated form:

```
calibre -recon -si[={ALL | DB | IO | PG | <tcl_script>}] -turbo \
{<rule_file> | {-svdb <svdb_dir> [<top_cell>]}}
```

The `-recon` and `-si` options are both required. If `rule_file` is specified, then the rule file flow is used (again, these are LVS circuit extraction rules). If `-svdb` is specified, then the Mask SVDB Directory flow is used. The run is hierarchical and multithreading is supported. Hyperscaling is supported when a rule file is specified.

The rule file flow performs short isolation using LVS circuit extraction rules and a layout as inputs. The Mask SVDB Directory flow requires the `-svdb` option and loads an SVDB from a previous LVS circuit extraction run and performs short isolation on that data. Both flows

depend on a Mask SVDB Directory statement in the rules that produces a persistent hierarchical database (PHDB). Some of the common keyword options that do this include CCI, QUERY, or SI.

If -si is specified without the equals sign suffix, then all nets at the primary level of the design are processed for short isolation. Results are presented in the same way as when BY CELL BY LAYER are specified with LVS Isolate Shorts. If the equals sign is appended to -si, then an additional modal argument must be specified that controls which nets are processed for short isolation. These are either nets at all levels of the hierarchy, I/O signal nets at the primary level, or power and ground supply nets at the primary level. If the *tcl_script* is specified, this is a Query Server Tcl shell script that runs commands from that interface on the PHDB data. Again, the DB modal operator produces only a PHDB for short isolation in subsequent runs on the SVDB.

The DB modal operator is a special case for use in a two-step flow. The first run uses -si=DB and creates just a PHDB in either a new or existing SVDB. Subsequent runs perform short isolation on the data in the SVDB (-si without “=DB” and -svdb are used). This can be faster than doing everything in a single run in certain cases, especially for larger designs or when there are many shorts anticipated in the layout.

In runs other than an -si=DB run, only connectivity extraction and short isolation are performed (no device extraction or anything dependent upon it, but Device statements are used to identify pin layers). These flows support interactive short isolation immediately upon loading the generated SVDB data into either Calibre RVE or the Calibre Query Server Tcl shell. This interactive capability is provided automatically and requires no additional rule file setup. For more information about interactive short isolation in Calibre RVE, see “[Interactive Short Isolation](#)” in the *Calibre RVE User’s Manual*. For more information about this topic in the Query Server context, see “[Using Short Database Commands](#)” in the *Calibre Query Server Manual*.

Here are some command line examples with brief descriptions:

```
# isolate shorts at the top level of the design
calibre -recon -si -turbo rules

# isolate shorts in all cells. do not allow LVS Isolate Shorts NAMES
# keyword to override short isolation on all nets.
calibre -recon -si=ALL -turbo rules

# isolate top-level supply shorts using an SVDB containing a PHDB
calibre -recon -si=PG -turbo -svdb svdb
```

An [LVS Isolate Shorts](#) NO statement, specified either explicitly or by default, is ignored in this flow. LVS Isolation Shorts YES secondary keywords are observed if specified in the rules and are applied to the run if they do not conflict with the command line option behavior.

There is also a [LVS SI Select Connects](#) specification statement and a [short_db::select_connects](#) Query Server command that may be used to configure which Connect and Sconnect operations

are processed prior to short isolation. By default, all such operations that produce connectivity to the level of the device pins are processed.

Using this flow requires a Calibre nmLVS Reconnaissance license. A hierarchical LVS license pair is required if a rule file is specified. A Query Server license is required when `-recon -si=tcl_script` is used.

Chapter 10

Device Recognition

Device recognition extracts instances of devices from collections of shapes in the layout, computes specified properties of these instances, and prepares the results for use by other processes.

Device Recognition Overview	307
Concepts and Terminology	308
Recognition Logic	310
Device Signatures	315
DFM Property in ADP Device Recognition	318
Bad Device Reporting	320
Hierarchical Device Recognition	322
Property Computation	325

Device Recognition Overview

DEvice statements govern how devices are recognized and classified, and how properties are computed.

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 extraction of a layout SPICE netlist.

Following device recognition are other client processes that use the extracted device information. Which of these clients is active depends on the command being performed. One client is LVS comparison, for which device recognition prepares a netlist and related information. LVS then compares this layout netlist to a second netlist, usually from a schematic, and reports any discrepancies. See the “[LVS Circuit Comparison](#)” chapter for details.

Another 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.

See “[LVS Best Practices](#)” in the *Calibre Solutions for Physical Verification* manual for rule file setup instructions.

Concepts and Terminology

Device recognition attempts to match device instances in the layout based upon DEVice statements you provide in your rule file. DEVice statements are patterns, or templates, which the device recognition algorithms use to classify device instances and compute their properties.

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 a collection of shapes that can classify an instance of the device in the 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:

Element name — Identifies the specific kind of device, which corresponds to the element name (also known as component type) in a schematic. Some names are special to the device recognition module, and these are discussed later in this chapter. Examples are MN and MP for MOS N- and P-type transistors, and C for a capacitor. You can also create your own names for devices.

Model name — Identifies the component subtype (by default, the MODEL property) in the schematic.

Device layer — Layer that contains device body, or seed, shapes.

Pin layers — Layers on which device pins are to be found. One or more pins must be present.

Pin names — Pins names of the device.

Swap lists — Identify pin swap groups, which allow interchangeable connectivity of device pins or pin groups. 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.

Auxiliary layers — Layers containing shapes that are not pins and that aid in the recognition of device instances. These come in two types: normal auxiliary layers and annotation auxiliary layers. Normal auxiliary layers are used only for device classification. Annotation auxiliary layers have DFM properties attached to them and are used with a property annotation program.

Signatures — Layers in the vicinity of a device seed shape that are used for identification of a device.

Text model layer — Specifies a text layer in the layout used to determine model names for device instances.

Property specification — Optional floating-point number or user-defined program that defines how property values are computed. A program overrides all default property computations for built-in device types.

Property annotation specification — Optional user-defined program used in the Pushdown Separated Properties (PDSP) flow to calculate simulation properties. Can be used with its own auxiliary annotation layer.

Here is an example of a DEDevice statement with labeled elements:

Element name	Model name	Device layer	Pin layers and names
<pre>DEVICE MN(nch) gate poly(g) sd(s) sd(d) well(b) [</pre>			
<pre>PROPERTY W, L WEFFECT = effective_width_constant //user-supplied W = .5*(perim_co(S,seed) + perim_in(S,seed) + perim_co(D,seed) + perim_in(D,seed)) L = AREA(seed) / W if ((BENDS(seed) != 0) && (WEFFECT != 0)) { if (W > L) W = W - WEFFECT * bends(seed) * L else L = L - WEFFECT * bends(seed) * W }</pre>			

A device instance inherits the element and model names from the DEDevice statement definition of which it is an instance. An instance 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.

Device Template Selection by Seed Layer

The LVS [[Un](#)]Select Device By Seed Layer specification statements allow configuration of device recognition based upon specified seed layers. These statements constrain device recognition to a subset of DEDevice statements in the rules.

Recognition Logic

Device instance recognition identifies sets of interacting device seed and pin shapes that match the specifications in a DEVICE statement.

The process proceeds as follows:

- Scan each layer that appears as a device layer in one or more DEVICE statements, together with all relevant auxiliary and pin layers.
- For each seed shape, identify an initial set of auxiliary shapes (if any), signatures (if any), 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 statements; 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	310
Pin Relations	311
Fill-In Algorithm	313
Ill-Formed Devices	314

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 statements 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, which compete to classify each shape on layer dev_lay:

```
DEVICE D1 dev_lay pin_lay_1(A) pin_lay_2(B)
DEVICE D2 dev_lay pin_lay_1(A) pin_lay_2(B) pin_lay_2(C)
DEVICE D3 dev_lay pin_lay_3(X) pin_lay_4(Y)
```

The interpretation of a DEVICE statement depends on the existence of other statements sharing the same device layer.

In this 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 statements and is classified as a BAD DEVICE in the circuit extraction report.

However, if the statement for DEVICE D3 were eliminated, the shape would be classified as a D1 device. The reason is, in the absence of the DEVICE D3 statement, only pin layers

`pin_lay_1` and `pin_lay_2` are scanned for pins, and the connection with a potential pin on `pin_lay_3` would not be recognized.

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 DEDevice statement. 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 DEDevice statement 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 DEDevice statements. If not, then the device is classified as bad.

Related Topics

[Recognition Logic](#)

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:

- The nets assigned to the shapes on a pin layer, not the shapes themselves, determine the initial pins on a layer when BY NET is specified either explicitly or by default.
- Each shape on a pin layer is treated as a separate pin if the secondary keyword BY SHAPE is specified in the DEDevice statements.

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 pin fill-in phase.

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 DEDevice 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 DEDevice statement.

The exact match rule allows different devices sharing the same device layer to be recognized on the basis of the layers where their pins appear. The rule file compiler prevents two DEDevice statements from having identical sets of pins and layers, so an exact match is guaranteed to be unique.

Device recognition uses a fill-in algorithm to attempt a match if BY NET is specified, either explicitly or by default.

The instance is classified as ill-formed if the exact match rule fails and BY SHAPE is specified.

Property Distribution for Shorted Device Pins

The DEDevice operation allows specification of multiple pins on the same layer (source and drain pins of MOSFET transistors are a common example of this). The DEDevice BY NET keyword (the default) replicates a single net attached to one of these pins when not enough independent nets were found. In the case of the source and drain pins for MOSFET devices, this allows recognition of devices with shorted source and drain pins, even though only one unique net is present for the device.

The pin replication feature assigns correct properties to each pin if it can determine a direct polygon-to-pin mapping. In order for this to occur, the number of polygons making up the shorted pins must exactly match the number of shorted pins. If there are fewer or more shapes than there are defined pins, the properties are distributed as before, so that one pin has the accumulated properties for all interacting polygons and the other has 0 values assigned.

For example, consider the following device statement:

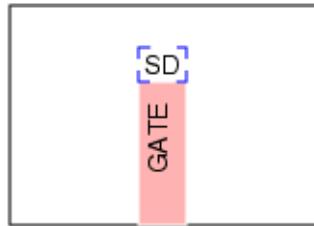
```
DEV MP(S0) PGATE PGATE(G) SD_P(s) SD_P(d) BY NET
[
    PROPERTY AS, AD, PS, PD
    PS = PERIM(s)
    PD = PERIM(d)
    AS = AREA(s)
    AD = AREA(d)
]
```

As long as the device is formed from exactly one source and one drain polygon, the properties are properly distributed between the two pins:

```
M0 5 1 5 P AD=1.516e-15 AS=2e-15 PD=2.88e-07 PS=3.02e-07 $X=-291 $Y=85
$D=0
M1 6 2 6 P AD=1.024e-15 AS=7.83e-16 PD=1.7e-07 PS=1.64e-07 $X=-253 $Y=171
$D=0
```

For a case where the number of polygons do not match the number of pins, consider [Figure 10-1](#):

Figure 10-1. Gate with Shorted S/D Pins



Here, the source/drain pin is a single polygon. When BY NET device recognition is in effect, this forms a valid device because only one net is present, and pin replication allows formation of shorted source and drain pins. Properties for this device, however, are not distributed because source and drain pin polygons are not unique.

Similar results occur where two or more shorted polygons are used for the SD regions of the device.

Results for devices that have exact pin-to-polygon matching are the same as those that use the BY SHAPE device recognition option, with the added benefit that BY NET device recognition offers much better performance than BY SHAPE device recognition.

This capability negates the most common reason for use of the BY SHAPE device recognition feature. Rule files using BY SHAPE in order to avoid 0 properties on source and drain pins should use BY NET instead.

Related Topics

[Recognition Logic](#)

Fill-In Algorithm

The fill-in algorithm tries to find a unique match to a DEvice statement 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 certain conditions are true.

Provisional pin matching occurs under these conditions:

- The pins and layers of the initial set can be matched with a subset of the pins and layers in the DEDevice statement.
- The initial pin set contains at least one pin on each pin layer of the DEDevice statement.
- The initial pin set contains exactly one pin on each pin layer of the DEDevice statement 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 DEDevice statement generates a provisional match.

The instance is classified as *ill-formed* if more than one DEDevice statement generates a provisional match.

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

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

A four-pin device instance, with one pin each on layers coll, emitt, base, and sub, is classified as a BPL 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 DEDevice statement are considered shorted together for this instance because they are from the same layer.

III-Formed Devices

When device recognition classifies a shape on a device layer as ill-formed, they are identified as discrepancies.

These things occur when a device is ill-formed:

- Device reduction ignores any pin and auxiliary shapes the ill-formed shape touches.
- The circuit extraction report lists the ill-formed (BAD) devices by the (x,y) location of the lower-left corner of the failed device shape, along with the element names from all DEDevice statements using that device layer.

Example 10-1. Device Classification

As an example of several DEvice statements competing to classify a device shape, consider the following statements 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 solely 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 preceding 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.

Related Topics

[Bad Device Reporting](#)

[Recognition Logic](#)

[Fill-In Algorithm](#)

Device Signatures

Device signatures are managed by the DEvice statement’s SIGNATURE keyword. A signature is a set of polygons that is in proximity to a device seed shape and is used to identify the device type of a layout instance.

Device signatures are used in conjunction with so-called *golden device libraries*, where the polygons surrounding certain devices must be considered in the identification of layout instances. This can be of particular concern in analog layouts.

There are two modes for using the SIGNATURE keyword:

Signature Generation — This mode is used to create a device signature to include in a rule file for future use in device extraction.

Signature Comparison — This mode is used during device extraction when a device instance in the layout is checked to see if it matches a signature in the rule file.

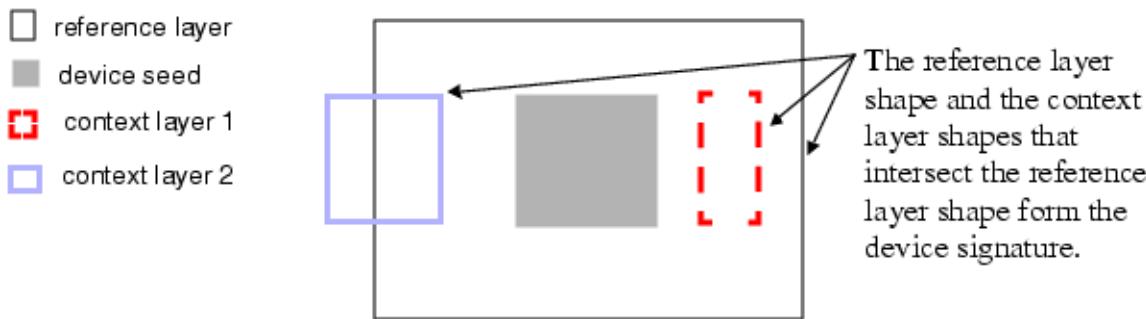
Use of the SIGNATURE keyword is part of the Calibre Advanced Device Properties (ADP) package and requires an ADP license in addition to your LVS license. If using multithreaded mode, you need additional licenses according to the usual licensing scheme for such applications.

Signature Generation

In order to generate a device signature, you must specify a *reference layer* (or a halo layer) and any number of *context layers*.

The reference layer defines the neighborhoods around seed shapes from which signatures are derived. Each reference layer polygon intersects (the Boolean AND is non-empty) a device seed shape and the context layer shapes to be used in generating the signature. The reference layer shape (one per seed shape), along with the context layer shapes that intersect it, form the signature. Text on reference and context layers is ignored. See [Figure 10-2](#).

Figure 10-2. Device Signature



Unless specifically added as context layers, the DEvice seed layer, pin layers, and any auxiliary layers do not contribute to the signature calculation.

The signature matches any orientation of geometry that includes rotation (in 90-degree increments) as well as reflection. Therefore, if your pattern of context layer polygons is not symmetrical in both x and y directions, you should include enough of the device-forming layers in your context layer list to ensure that the orientation of the signature layers relative to the device can obtain a match.

Signature layers that cannot be represented exactly in database coordinates may not produce consistent signature matches in the device recognition module. This is of concern when using derived layers containing angled edges that are subject to grid snapping. Polygons on such layers may not snap to grid consistently in all instances and may result in different signature matches than what is desired. For this reason, you should avoid specifying signature layers that are subject to grid snapping.

To generate a signature, you specify a DEDevice statement similar to this one in a circuit extraction run:

```
DEVICE C(cmod) ... SIGNATURE "?" ref_layer c1 c2
```

The “?” indicates you want to generate a signature. The ref_layer is the reference layer. The layers c1 and c2 are the context layers. The reference layer and the context layers are any layers of your choosing, although the reference layer must intersect the device seed shape and any context layers. Once a reference layer is used with a specific set of context layers, it may only be used with that set of context layers in the rule file.

Using the previous DEDevice statement in your rule file, you then run Calibre with the -siggen option (see “[Calibre nmLVS and Calibre nmlVS-H Command Line](#)” on page 45 for details) as follows:

Example 10-2. Device Signature Generation

```
calibre -spice layout.spi -flatten -siggen rules # flat extraction
calibre -spice layout.spi -siggen rules           # hier extraction
calibre -lvs -hier -siggen ... rules            # LVS-H with geometric
                                                # Layout System
```

This causes a device signature to be generated in the run transcript, like this (with no line breaks):

```
DEVICE C (cmod) cap pos neg SIGNATURE
:AcZSuedavurxCm8B.C50lgI8IJZ:x5gUdvBv0kjEFns2ggba0PecBn:C5ro.FZ1HVevEiqN1
N
::BNDmGj4kZoo3Ev:7DJ4g4JEpHi2BnQ632KoAcDeltKFsnbVfJy.LeJUNzrsYv82.2KWmv2Q
J
:T1GfdfHM2P32aO6F5:tMAVL8do! ref_layec c1 c2
HALO SIGNATURE: cell1 TOPCELL has halo signature of
AcZSuedavurxCm8B.C50lgI8IJZ:x5gUdvBv0kjEFns2ggba0PecBn:C5ro.FZ1HVevEiqN1N
:BNDmGj4kZoo3Ev:7DJ4g4JEpHi2BnQ632KoAcDeltKFsnbVfJy.LeJUNzrsYv82.2KWmv2QJ
T1GfdfHM2P32aO6F5:tMAVL8do!
```

The geometric signature is encoded as an alphanumeric string. The generated Device statement can then be copied and pasted into the rule file and used for signature comparison. It must be copied exactly (do not copy the HALO SIGNATURE line), with no line breaks or other modifications.

You must ensure that the reference layer extents are representable in 32-bit coordinate space.

Signature Comparison

Signature comparison uses the signature string from the Signature Generation phase.

The alphanumeric string is pasted into the DEDevice statement in place of the “?” character, like this (without line breaks in the signature):

```
DEVICE C(cmod) ... SIGNATURE
"AcZSuedavurxCm8B.C50lgI8IJZ:x5gUdvBv0kjEFns2ggb0PecBn:C5ro.FZ1HVevEiqNl
N:BNDmGj4kZoo3Ev:7D4g4JEpHi2BnQ632KoAcDeltKFsNbVfJy.LeJUNzrsYv82.2KWmv2Q
JT1GfdfHM2P32aO6F5:tMAVL8do!" ref_layer c1 c2
```

Signature comparison occurs during device extraction. The signature string is decoded during runtime and compared to devices in the layout to see if they match the geometric signature represented by the string. If a device instance matches the signature, it is netlisted as a device that matches the corresponding DEDevice statement. Signature comparison does not require the use of the -siggen command line option.

It is important that the reference layers and context layers specified in the DEDevice statements match, *including the order* in which they are specified, in the [Signature Generation](#) and comparison runs, otherwise the signature does not find a match.

DFM Property in ADP Device Recognition

The DFM Property operation is used in Advanced Device Properties (ADP) applications. This operation requires care in handling undefined mathematical expressions, such as division by zero.

[DFM Property](#) enables the assignment of property values to shapes, which enables classification and property calculation for certain devices. See “[Example Device Annotation Programs Using DFM Property](#)” in the *SVRF Manual* for a discussion of this capability.

When using DFM Property, it is important to protect against *expression evaluation failure*. Expression evaluation failure occurs in any situation where DFM Property cannot assign an appropriate property to polygons. Examples include numerical errors, lookups on non-existent properties, out-of-range properties, and so forth. Here is an example:

```
prop_layer = DFM PROPERTY gate_layer ml2 OVERLAP MULTI
[ a = area(ml1)/count(ml2) ] // possible divide-by-zero
```

If `count(ml2) == 0`, there is division by 0 and the property “a” cannot be assigned a value. Consequently, a polygon on `gate_layer` cannot be copied to `prop_layer`, and `prop_layer` will have fewer polygons than the original `gate_layer`.

In Calibre ADP applications, `prop_layer` is usually an auxiliary layer in a DEDevice statement. The `gate_layer` contains the seed shape. If the aux layer polygon is missing, the device on the

gate_layer is classified as a BAD DEVICE. If prop_layer is supposed to contain the seed shape, then the device is not extracted.

Partial copies of device layers are problematic in the ADP flow. Devices in the layout are not extracted and it is difficult to determine the cause. DFM Property issues a warning in the transcript if the operation did a partial copy due to expression evaluation failure. You should look for such warnings if you suspect this problem. Also, compare the FGC (flat geometry count) for the gate_layer to the FGC of the prop_layer to confirm whether polygons were lost. Here is an example from a transcript:

```
gate_layer (HIER TYP=1 CFG=1 HGC=4578 FGC=316325 HEC=137654 FEC=37475542
VHC=F VPC=F)
```

If this FGC number does not match the prop_layer, there is a problem in getting the properties onto the correct polygons.

All ADP rule files can avoid this problem by including *fall-through expressions* for each property value where a problem might occur. A fall-through expression is a second expression that applies if the first expression results in an exception. Using a fall-through expression ensures preservation of all polygons from the first input layer. For example:

```
prop_layer = DFM PROPERTY gate_layer ml2 OVERLAP MULTI
    [ a = area(ml2)/count(ml2) ] // possible divide-by-zero
    [ a = -1 ] // error value for divide-by-zero case
```

When count(ml2) == 0, the property is assigned the value -1. The Device built-in language program should check for this error value and handle it appropriately. Of course, the error value could be anything. You must determine the convention to use and ensure it is handled appropriately in the device property calculation.

You can also set defaults for vector properties. Consider this example:

```
prop_layer = DFM PROPERTY gate_layer ml2 OVERLAP MULTI
    [ p = VPROPERTY(ml2,"p",3) ]
    // possible out-of-range ordinal lookup
    [ p = vector(,,) ]
    // handle out-of-range ordinal by creating an
    // empty vector with tuple_size==3
```

With vector properties, the fall-through expression must generate a vector with the same tuple size as the first expression would generate.

DFM Property checks that all subsequent references to the same output property are of the same type (and for vectors, tuple size is included in that type checking).

Bad Device Reporting

Bad (or ill-formed) devices are reported in the circuit extraction report file and the run transcript.

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.

The device recognition phase of circuit extraction may report bad devices in the circuit extraction report when the following conditions are met:

- DEVice operations are specified with a TEXT MODEL LAYER.
- More than one text object touching a seed shape is present on the TEXT MODEL LAYER, and those text objects are not equivalent according to string comparison.
- Layout Preserve Case YES is specified.

Note that equivalence of multiple text objects determined by string comparison is subject to the setting of the Layout Preserve Case statement. If Layout Preserve Case NO is specified (this is the default), then text strings that differ only by case are treated as identical and the device recognizer forms a *valid* device.

For example, two text objects “abc” and “ABC”, present at different locations on the same TEXT MODEL LAYER and touching the same seed shape are considered a valid device. The model name of the device in the extracted SPICE netlist is either “abc” or “ABC”, chosen arbitrarily.

When Layout Preserve Case YES is specified, text objects on the TEXT MODEL LAYER must match exactly, including case, to be considered equivalent. In the previous example, the text objects “abc” and “ABC” are considered non-equivalent and classified as a bad device in the circuit extraction report with the message “Too many Text Model Layers.”

- 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 DEDevice statements in the rule file that use that device recognition layer.

These are possible reasons for classifying a device as bad:

- (No matching DEDevice statement).

This usually means that some pins or auxiliary shapes are missing.

- (Too many pins).

More physical pins exist than exist in a DEDevice statement.

- (Too many matching DEDevice statements).

This usually means that pins on one or more pin layers were present in insufficient numbers, and more than one distinct DEDevice statement could be matched by the pin fill-in algorithm.

- (Too many TEXT MODEL LAYER texts).

More than one TEXT MODEL LAYER text object intersects the seed shape.

- (Too many TEXT PROPERTY LAYER texts on same layer).

More than one TEXT PROPERTY LAYER text object on the same layer intersect the seed shape.

- (DEVICE signature mismatch).

There is a mismatch between the layout device and its signature in the corresponding DEDevice statement. This message is issued only if all required pin and auxiliary shapes are present and the only reason the instance is rejected is the signature mismatch.

- (User device program marked device bad).

A DEDevice TVF Function has a bad_device command that classified a device instance as bad.

You can specify that bad devices cause an INCORRECT comparison status by using [LVS Report Option EB](#).

Calibre nmLVS-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 545) is added.

Examples:

```
WARNING: BAD DEVICE on layer badseed at location (5,-7.5) in cell D
          (No matching DEVICE operation).
          Found 0 interaction(s):

          No interacting pins, auxiliary shapes or texts were observed.

WARNING: BAD DEVICE on layer Pseed at location (2,-6.75) in cell D
          (Too many pins).
          Found 4 interaction(s):
          Pin on layer poly, net 5 at location (2,-6.625)
          Pin on layer SD, net "VCC" at location (2,-6.625)
          Pin on layer SD, net 12 at location (2,-6.125)
          Pin on layer SD, net 13 at location (2,-5.5) (extra pin)

WARNING: BAD DEVICE on layer 901 at location (0,0) in cell zcel
          (Too many matching DEVICE operations).
          Found 1 interaction(s):
          Pin on layer 903, net 1 at location (0,5)

WARNING: BAD DEVICE on layer Pseed at location (3,-8.5) in cell D
          (Too many TEXT MODEL LAYER texts).
          Found 4 interaction(s):
          Pin on layer poly, net 7 at location (3,-8.25)
          Pin on layer SD, net 15 at location (3.5,-8.25)
          Text "m2" on layer metal at location (3.123,-8.375)
          Text "m3" on layer metal at location (3.123,-8.125) (extra model
text)

WARNING: BAD DEVICE on layer POLY at location (0.479,0.836) in cell
TOPCELL (DEVICE signature mismatch).
          Found 4 interaction(s):
          Pin on layer POLY, net 1 at location (0.479,0.907)
          Pin on layer SD, net 2 at location (0.479,1.546)
          Pin on layer SD, net 3 at location (0.751,1.264)
          Reference shape on (reference layer) at location (0.479,0.907)
          Possible Element Names: M
```

Flat LVS — When -flatten is used, a circuit extraction report is produced as in hierarchical LVS. The formatting is essentially the same as in hierarchical LVS discussed previously, but the reporting is from a top-level perspective.

Hierarchical Device Recognition

The DEVICE operation includes two secondary keywords that apply to hierarchical applications: BY NET and BY SHAPE. These have effects on how devices are recognized in a hierarchical run. Seed promotion occurs in certain cases when a device instance cannot be recognized at the original hierarchical level that a seed shape appears. It is important to understand the causes for this when it occurs.

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 — Hierarchical device recognition promotes each device up the hierarchy until all pin shapes are fully merged, which may be compute intensive.

Seed Promotion

Seed promotion occurs in hierarchical runs when devices cannot be formed at the same level as the seed shape. In certain cases, device seed shapes have to be promoted out of their cells in order to fully form devices.

In many cases, promoted seed shapes can be pushed back down the hierarchy into their original cells, and the device instances are recognized there. The TENTATIVE SEED PROMOTIONS section of the run transcript shows initial seed promotion before device pushdown is attempted. The [LVS Push Devices](#) statement controls this pushdown behavior.

Tip

 To mitigate the effect of seed promotion, layer operations that tend to preserve layout hierarchy should be used when deriving device seed layers. See “[Hierarchy Preservation by Layer Operations](#)” on page 491.

If device pushdown cannot be completed, the final seed promotions are shown in the SEED PROMOTIONS section of the run transcript. You will also see subcircuits in the extracted netlist that contain the *.SEEDPROM statement, such as this example:

```
.SUBCKT cell_1 1 2 4
** N=4 EP=3 IP=0 FDC=1
*.SEEDPROM
M0 4 1 2 2 n L=5e-08 W=1.25e-07 $X=1025 $Y=850 $D=0
.ENDS
```

The subcircuit cell_1 has had an NMOS seed shape promoted out of it.

Device pushdown can be unsuccessful for four primary reasons:

- Only a portion of promoted seed shapes form good device instances. The remainder are classified as bad devices.
- Some of the promoted seed shapes result in instances recognized by one DEViCE operation, while the remainder are recognized by another DEViCE operation. This can occur, for example, when the instances differ by pin count, and the DEViCE statements are sensitive to this difference.
- Promoted seed shapes result in instances that differ by pin connectivity.

- Promoted seed shapes result in instances that differ by property values. (Use LVS Push Devices SEPARATE PROPERTIES if you are using the pushdown separate properties flow with Calibre xRC, and seed promotion occurs due to property differences.)

Seed promotion out of hcells can cause discrepancies in the hierarchical comparison phase. This is because the subcircuits where promoted devices are instantiated differ between layout and source. The usual way to alleviate this problem is to specify [LVS Expand Seed Promotions YES](#).

When layer derivation with the [AND](#) operation is responsible for device pins forming in undesired locations in the hierarchy, an alternative is to use [LVS Use Careful AND YES](#), or the LVSCAREFUL keyword of the AND operation. Device promotion due to layer derivation may or may not lead to *.SEEDPROM statements in the extracted netlist. It depends on when the layer promotion occurs—during hierarchical database construction or during device recognition.

See “[Seed Promotion and Hcells](#)” on page 481 for a more discussion of this subject.

Property Computation

Calibre nmLVS uses computed property values when comparing the layout netlist to the schematic netlist. 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.

For each DEvice statement 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 330).

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 and is also considered built-in from the standpoint of device recognition, but no default properties are computed for this device.

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 0 (zero) for the element names C and R.
 - No properties are computed for any other element name.
- A list of floating-point numbers for default property computations can only be used with the reserved element names; C, R, MN, MP, MD, and ME. In these cases, the numbers specified in the DEvice statement are used by an internal program to calculate default property values. 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 default property computations are shown in the “Examples” section of the [DEvice statement description in the SVRF Manual](#).

- The user-defined 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 section “[Default Property Computations](#),” discusses the default property computations available for element names C, D, R, MN, MP, MD, and ME. The sections “[User-Defined Property Computation](#)” through “[Property Computation Debugging](#)” discuss the use of the special property computation language.

Default Property Computations

Default property computations are available for devices with reserved element names.

The default traceable properties and the corresponding property names are shown in the third and fifth columns, respectively, in [Table 10-1](#).

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

Table 10-1. Default Device Properties

Name	Meaning	Default traceable netlist properties	Property parameters in DEDevice statement	Trace Property names
C	capacitor	capacitance (numeric)	area_cap, perim_cap	C
D	diode	AREA, PJ	(none)	A, P
MN	transistor	W, L	effective_width_factor	W, L
MP	transistor	W, L	effective_width_factor	W, L
MD	transistor	W, L	effective_width_factor	W, L
ME	transistor	W, L	effective_width_factor	W, L
R	resistor	resistance (numeric)	resistivity	R

For each DEDevice statement containing a reserved element name from [Table 10-1](#), you can supply appropriate property parameters. An exception is the diode, which takes no parameters. If the property parameters are omitted, they default to 0. 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 [DEDevice](#) section in the *SVRF Manual*.

The following covers the default property calculations.

- **Capacitors** — Capacitance is computed for C devices by default. Here is an example device statement:

```
DEVICE C cap_layer pos_layer neg_layer [ 1.6 0.07 ]
```

Two numeric 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 statement, they must both be present, and the area constant must appear first. Both numbers must evaluate to non-negative, floating-point numerics. Specifying either value as 0 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 0 and a capacitance of 0 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 per square mil and the perimeter capacitance to be 10 nanofarads per mil:

```
UNIT CAPACITANCE nf
UNIT LENGTH mil
DEVICE C cap_layer pos_layer neg_layer [ 300 10 ]
```

The formula is this:

$$\text{capacitance} = (\text{ca} * \text{area} + \text{cp} * \text{perimeter})$$

where area and perimeter are computed as for diodes, and ca and cp are the parameters supplied by the property parameters in the DEvice statement. Although the area and perimeter are computed internally for device reduction calculations, they are not made available as property values for device instances.

The following user-defined properties can be calculated and traced for capacitors: C, A, P, L, and W. They are netlisted as a numeric value, \$A, \$P, L, and W respectively.

- **Diodes** — Area and perimeter are computed internally for D devices by default. Here is an example device statement:

```
DEVICE D dio_layer pin_lay_1 pin_lay_2
```

Because no numeric parameters are required for this computation, no property parameters (in brackets) are specified in the statement. The calculated property values are expressed in square meters and meters, respectively.

Area is netlisted as AREA and perimeter is netlisted as PJ. However, these properties should be traced as A and P, respectively. The same holds true if these are calculated as user-defined properties.

- **MOS Transistors** — Effective width and effective length of the device are computed with compensation for bends in the device area. Here is an example device statement:

```
DEVICE MP dev_lay gate source drain [ 0.5 ]
```

You can specify one parameter—the effective width (*weffect*) constant—in the DEVICE statement. If omitted, it defaults to 0, 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 0. The parameter must evaluate to a non-negative, floating-point numeric.

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 statement.

- The *weffect* is 0: 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:

$$I = S / 90$$

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:

$$W - weffect * I * L$$

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

$$L - weffect * I * W$$

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

The following user-defined properties can be calculated and traced for MOS devices: L, W, AS, AD, PS, PD, NRS, NRD. They are netlisted using the same names.

- **Resistors** — Resistance is computed for resistors. Here is an example device statement:

```
DEVICE R res_layer pin_lay pin_lay [ 1.1 ]
```

One parameter is needed: the resistivity in units of resistance per square. This parameter must evaluate to a non-negative number. If omitted, the value defaults to 0 and a resistance of 0 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 per square:

```
UNIT RESISTANCE kohm
DEVICE R res_layer pin_lay pin_lay [ 0.1 ]
```

The formula is this:

$$\text{resistance} = r * (L / W)$$

where, L is the length and W the width of the resistor, and r is the resistivity parameter supplied by the property parameter in the DEVICE statement.

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 actual 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 statements to alter the geometry prior to computation and then to specify the computation using the built-in property language.

The following user-defined properties can be calculated and traced for R devices: R, L, and W. These are netlisted as a numeric value, L, and W.

User-Defined Property Computation

The DEvice statement allows you to define your own property computation algorithm for any device. 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, including any properties that would have been computed by default.

The optional property computation section of the DEvice statement uses a built-in language. This language is described under “[Device Property Computation Built-In Language](#)” in the *SVRF Manual*.

Your property names should follow SPICE conventions for allowed characters. See “[SPICE Syntax Conventions](#)” on page 634. The rule file compiler allows you to specify property names in a DEvice statement that are not SPICE compliant, so care should be used to specify names that are permitted in SPICE.

The SPICE formats for built-in devices are shown under “[Element Statements](#)” on page 677. Property names generally follow the canonical properties shown in that section for built-in devices.

For well enclosure stress effect property computation, use the “[Device Annotation Built-In Language](#)” described in the *SVRF Manual*. This language is used in conjunction with the Calibre Connectivity Interface of the Query Server for extracting properties commonly calculated for BSIM4 device models.

Comment-Coded Properties for C and Q Elements	330
Units of Measurement	331
Property Computation Structure	332
Coding Efficiency Considerations	334

Comment-Coded Properties for C and Q Elements

If you calculate A and P properties for capacitors (C element), by default, capacitor instances have comment-coded properties \$A and \$P written for them in the extracted layout netlist. These correspond to area and perimeter.

Similarly, bipolar devices (Q element) have comment-coded properties \$L and \$W written for them if you calculate properties L and W. These correspond to length and width.

You can change the comment-coded properties to actual properties by specifying [LVS Netlist Comment Coded Properties NO](#).

Units of Measurement

You should consider the selection of appropriate units when computing numeric properties that represent physical quantities.

Consider the case of computing the capacitance for a capacitor. The basic formula might be this:

$$C = \text{perim_factor} * \text{perim(device)} + \text{area_factor} * \text{area(device)}$$

To compute a proper value for C, the following seven questions must be answered:

1. In what units is perim(device) delivered?
2. In what units is area(device) delivered?
3. In what units should C be expressed?
4. In what units should area_factor be expressed?
5. In what units should perim_factor be expressed?
6. What is the appropriate value for area_factor?
7. What is the appropriate value for perim_factor?

The standard units for representing the values of time, length, area, capacitance, and resistance are these:

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 per meter, since the perimeter is in meters and the capacitance is in farads.
5. The units of area_factor should be expressed in farads per 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:

$$0.05 \text{ pf/um} = 0.05e+6 \text{ pf/m} = 5e-8 \text{ f/m}$$

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.

Related Topics

[User-Defined Property Computation](#)

Property Computation Structure

At the time a rule file is loaded, each DEvice statement containing a property computation causes the generation of a *value array*. The value array is an array of properties indexed from zero through a maximum number.

Each property in the array has one of the following types:

- double-precision floating-point number
- string
- complex type (usually an array of numbers)

There is only one value array for each DEvice statement. The entries in this array are used during the computation of properties for each instance of the device. The same array is used 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 10-2. Value Array Listing

Index	Type	Name (as shown in debugging output)
0	property value	w
1	property value	l
2	property value	as
3	property value	ad
4	local variable	weffect
5	constant	0.5
6	data value	area(S)
7	data value	area(D)
8	data value	perimeter_coincide(G, diff)
9	data value	perimeter_inside(G, diff)
10	constant	2
11	data value	perimeter_outside(G, diff)
12	data value	bends(G)
13	constant	0
14	temporary variable	temp1
15	temporary variable	temp2

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.

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.

When Device TVF functions are used for property computations (see “[TVF Functional Interface for Device Property Calculation](#)” in the *SVRF Manual*), the TVF Function call is loaded into a Tcl interpreter and run with dummy values at compile time to check for Tcl syntax errors. It may run several times with dummy values before actual device instances are processed. At device recognition time, the requested parameters are passed in to the Tcl proc as it is called. The value returned by Tcl proc is assigned to the property array upon completion of the proc.

Related Topics

[User-Defined Property Computation](#)

Coding 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.

The rule file compiler attempts to move parts of the computation from the evaluation phase to the initiation phase. (For information on the initiation and evaluation phases of computation, refer to “[Property Computation Structure](#)” on page 332.) 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 the computations do not depend on instance data, they are performed only once during initiation and the results used many times during evaluation.

The source of instance-dependent data is the following functions:

All AREA functions	X_LOCATION()
All PERIMETER functions	Y_LOCATION()
COUNT()	PIN_NET()
BENDS()	INSTANCE()
All ENCLOSURE_* functions	All DFM_* functions
All TVF_* functions	

All other functions, including NAMED_NET(), deliver instance-independent data. (For reference, the complete descriptions of device property calculation functions are found under “[Device Property Computation Function Reference](#)” in the *SVRF Manual*.)

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 sub-expressions that are instance-independent. It moves the computation of these sub-expressions 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 sub-expressions.

In examining sub-expressions, the compiler treats constants, process variables, and instance-independent function calls as instance-independent. However, the compiler is 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 preceding concepts. In this example, you should assume that P and Q represent pin names, and that normal_adjustment is a process variable.

```

1  [
2      property ap, aq, inst
3      power_adjustment = 0.04 + normal_adjustment
4      power_net = named_net("VDD")
5      if (pin_net(P) == power_net)
6          pin_adjustment = power_adjustment
7      else
8          pin_adjustment = normal_adjustment
9      ap = area(P) + pin_adjustment
10     aq = area(Q) + 2 * normal_adjustment
11     inst = instance()
12 ]

```

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 a value array during initiation. The evaluation phase obtains these variables directly from the value array. Also, the statement sub-expression:

$2 * \text{normal_adjustment}$

on line 10 is instance-independent. A variable “init1” is created in the value array to store the value of this sub-expression. Line 10 is then treated as if it read like this:

$\text{aq} = \text{area}(Q) + \text{init1}$

Note that the compiler's two strategies do not find all of the instance-independent computations that could be moved. In the preceding example, if the statement:

```
inst = 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 sub-expressions:

```
0.04 + normal_adjustment
```

and

```
2 * normal_adjustment
```

However, if you use the following guidance, you can ensure that all instance-independent computations are computed only once in the initiation phase.

Here are some tips on writing clear and efficient property specification programs.

- **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 (computations that do not depend upon instance properties) are performed and stored in local variables. This portion is performed only once in the initiation phase. If you intermix independent and dependent portions, the compiler still attempts to move independent sub-expressions to the initiation phase, but it is not 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 bend_effect = 0.5 during initiation. Then during each evaluation, it accesses the bend_effect value just as efficiently as the first program accesses the constant 0.5 value.

```
[ // Use of unnamed constant. (OK)
  property w
  w = perimeter_coincident(G,pin_layer) - 0.5 * bends(G)
]

[ // Use of named constant. (Just as fast, but possibly
  // more meaningful.)
  property w
  bend_effect = 0.5 // This takes place only once at
                     // initiation time.
  w = perimeter_coincident(G,pin_layer) - bend_effect*bends(G)
]
```

- **Parenthesize instance-independent sub-expressions** — You might have to use parentheses for instance-independent expressions. For example, consider the following assignment statement where var is a process variable:

```
a = area(P) + 0.5 + var
```

The compiler does not find the instance independent sub-expression $0.5 + \text{var}$ in the preceding statement because it does addition from left to right by default. Thus, the expression is treated as if were parenthesized as follows:

```
a = (area(P) + 0.5) + var
```

To optimize the original statement, you can introduce your own parentheses as follows:

```
a = area(P) + (0.5 + 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 than one place in your program, you are simply accessing the pre-computed value, not causing it to be computed each time you reference it. Therefore, you do not have to store the value in a local variable. 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.

```
[ // Use of function values indirectly through local
  // variable. (SLOWER)
  property w, l
  common_perim = perimeter_coincident(G, pin_layer)
  w = common_perim
  l = perim(G) - common_perim
]

[ // Use of function values directly. (FASTER)
  property w, l
  w = perimeter_coincident(G, pin_layer)
  l = perim(G) - perimeter_coincident(G, pin_layer)
]
```

- **Use local property measurements** — Use seed layers for property calculations instead of pin layers or pin names. For example, if you have this Device statement:

```
DEVICE MP pgate poly(G) psd(S) psd(D) nw(B) <aux> ...
```

You should use a function call like this:

```
z = perimeter_coincide(pgate, aux)
```

rather than using poly or G for pgate. Even though pgate is likely derived from poly and is coincident with the G pin, using the local layer (the seed layer) prevents expansion of the poly layer during processing.

To further illustrate this idea, consider the following.

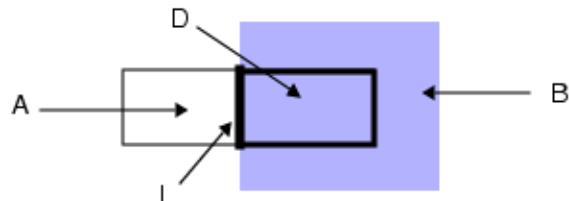
Suppose the devices to be recognized have a pin “P”, which is known to lie strictly within the area of the seed shape on layer dev_lay. Therefore, the two functions shown next are numerically equivalent. However, the area requested in the first function is not known by the compiler to lie entirely within 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.

```
AREA(P) // slower
AREA_COMMON(P, dev_lay) // faster, because dev_lay is a seed layer
```

In [Figure 10-3](#), 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.

Figure 10-3. Efficient Function Choice



<code>L = perim_in(B,A)</code>	//slower
<code>L = perim_in(D,A)</code>	//faster because D is a device seed layer

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.

- **Avoid time-consuming geometry functions** — 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.
 - 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.
 - 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.
 - Some of 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.
 - The most time-consuming functions are the ENCclosure_* functions that make collections of pin-specific geometric measurements. See “[Historical Well Proximity Calculation Methods](#)” in the *SVRF Manual* for a summary of these.

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.

- **Fastest functions** — Geometry on or within seed shape:
 - area(dev_lay)
 - area_common(<args>)
 - perimeter(dev_lay)
 - perimeter_outside(dev_lay, pin_lay)
 - perimeter_<variant>(<args>) all coincide and inside variants
 - bends(dev_lay)
 - x_location(<arg>)
 - y_location(<arg>)

- **Intermediate 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)
- *Slower functions* — Interaction between non-device layer shapes:
area_common(pin_lay_1, pin_lay_2)
perimeter_<variant>(pin_lay_1, pin_lay_2) all coincide, inside, and outside variants
- **Slowest functions** — These should generally be avoided because other device property APIs offer much better performance:
enclosure_parallel(<args>), enclosure_perpendicular(<args>),
enclosure_vector(<args>)

The complete descriptions of device property calculation functions are found under “[Device Property Computation Function Reference](#)” in the *SVRF Manual*.

Property Computation Debugging

By using the DEBUG command in a property computation program, you can obtain a detailed analysis of the property computation for any device instances you choose.

Suppose you have just written a built-in property computation and 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, run Calibre again to find the next error and repeat this process until the rule file loads successfully.

Having successfully loaded the rule file, you can now perform LVS comparison. Property computation programs are typically written during rule file development. For debugging purposes, it is best to run LVS flat on cells containing devices of interest when validating your programs.

A device property discrepancy appears in a PROPERTY ERRORS section in the LVS Report. The causes of some of these discrepancies may be apparent, which you can find by studying the property numbers produced in the report and comparing them to the rule file. Other property errors might not be so easily debugged.

For the complete list of Device property compiler errors, see “[Device Property Specification Errors](#)” in the *SVRF Manual*.

Debug Example	341
Hierarchical Run Considerations.....	343
Debug Statement Placement.....	344
Debug Output.....	344

Debug Example

The DEVice property computation language employs a DEBUG statement to facilitate debugging of property computation programs.

In the following code, a DEVice statement appears on lines 23-37 of the rule file. The line numbers are not part of the DEVice statement itself but are used in debugging output to identify which line of the program is responsible for each step in the analysis.

```
23     device mp(pmos) gate gate(G) diff(S) diff(D) base(B)
24     [
25     property W, L
26     bend_effect = 0.5
27     W = (perimeter_coincide(G, diff) +
28           perimeter_inside(G, diff)) / 2
29     L = perimeter_outside(G, diff) / 2
30     if (bends(G) > 0)
31     {
32       if (W > L)
33         W = W - bend_effect * bends(G) * L
34       else
35         L = L - bend_effect * bends(G) * W
36     }
37 ]
```

Suppose the rule file containing the DEVice statement shown previously generates results in the following PROPERTY ERRORS section of a flat LVS Report:

PROPERTY ERRORS			
DISC#	LAYOUT	SOURCE	ERROR
1	5 (34.125,51.000) mp	1/0/m0 mp (P)	
	1: 0.875 u	l: 1.75 u	50%
	w: 40 u	w: 20 u	100%

The property values for L and W do not compare correctly. The device instance that caused the problem is identified on the first line of the discrepancy, where we have these values:

- “1” is the discrepancy number.
- “5” is the device instance number.
- “34.125” is the x-coordinate.
- “51.000” is the y-coordinate.
- “mp” is the element name of the device.

In any flat run of device recognition, all device instances are numbered sequentially beginning with zero (the instances do not necessarily appear consecutively, however). This numbering is over all device types and subtypes. Hence, in flat LVS, determining which device instance number to debug is relatively easy.

In this case, layout instance 5 has the problem, so that is the instance number you use in a **DEBUG** statement.

Hierarchical Run Considerations

In a hierarchical run, device instances are generated on a per-cell basis. This makes finding the exact instance number to debug more challenging. It's often better to run small layout test cases flat when debugging property computations because finding the instances to debug is easier.

Here is an example:

***** PROPERTY ERRORS *****			
DISC#	LAYOUT	SOURCE	ERROR
1	M0 (5.125,51.000) MP (P)	M0 MP (P)	
	l: 0.875 u	l: 1.75 u	50%
	w: 40 u	w: 20 u	100%

Notice there is no layout instance number.

In order to use the [DEBUG](#) command effectively, you must know the instance ID to debug.

To obtain an instance ID number to use in a DEBUG command, you can use the [instance\(\)](#) function in the Device property computation to generate instance number properties in the extracted netlist.

For example, if you do this:

```
device mn ngate ngate(g) nsd(s) nsd(d) pwell(b) [
    PROPERTY W,L,I
    I = inst()
    ...
```

The extracted netlist will show “i” properties like this:

```
M2 7 IN1 GND GND n L=6.25e-07 W=3e-05 i=0 $X=5375 $Y=14000 $D=0
```

The property values can be used in DEBUG statements for tracing.

Because these instance IDs are not unique in a hierarchical run, all instances having the specified ID number are reported in the transcript debugging trace. You can find the appropriate cell by looking for this in the transcript:

```
EVALUATION: (cell: nand) Device: mn (line 36 of rule file)
```

To narrow down the scope of the output, you can change the [Layout Primary](#) to the parent cell of the devices of interest.

You must also determine which DEvice statement generated the instance with the property discrepancy so you know where to place the DEBUG statement. This is discussed under “[Debug Statement Placement](#)” on page 344.

Debug Statement Placement

You can use the \$D property to determine which DEVice statement in the rules is responsible for classifying the device.

Consider this instance:

```
M3 OUT IN2 7 GND p L=1.25e-06 W=1.5e-05 $X=13375 $Y=14000 $D=0
```

The \$D properties are indexed from 0, which corresponds to the first active DEVice statement in the rule file. Once you know the DEVice statement responsible for the classifying the device, you can place a DEBUG command into the relevant property computation program.

Assume that \$D=0 corresponds to the following DEVice statement. You can then place the DEBUG command for the instance number of interest into the property computation program, like this:

```
23     device mp(pmos) gate gate(G) diff(S) diff(D) base(B)
24     [ DEBUG 5
25     property W, L
...

```

The EVALUATION debug output line only appears for the appropriate instance in the flat case (it is often easier to debug property computation programs from small layouts containing your devices and running circuit extraction flat), along with the appropriate line of the rule file:

```
EVALUATION: Device: mp (line 96 of rule file)
Instance: 5 x: 34.1 y: 51
```

Debug Output

DEBUG statement output appears as comments in the run transcript.

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 statement 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).

Example 10-3. Device DEBUG Statement Output

```
INITIATION: Device: mp  (line 23 of rule file)
  Values before:
    0: w      0
    1: l      0
    2: i      0
    3: INST()      0
    4: bend_effect  0
    5: 0      0
    6: 0.5   0.5
    7: perim_co(psd,pgate)  0
    8: perim_in(psd,pgate)  0
    9: AREA(pgate)  0
   10: BENDS(pgate)  0
   11: temp1      0
   12: temp2      0
   13: temp3      0

  Interpreter called.
  Values after:
    0: w      0
    1: l      0
    2: i      0
    3: INST()      0
    4: bend_effect  0
    5: 0      0
    6: 0.5   0.5
    7: perim_co(psd,pgate)  0
    8: perim_in(psd,pgate)  0
    9: AREA(pgate)  0
   10: BENDS(pgate)  0
   11: temp1      0
   12: temp2      0
   13: temp3      0
```

Each value line 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 initialized to 0 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 computations. These left-over values do not affect the course of future computations because they are overwritten before they are used.

The rule file compiler determined that the assignment of bend_effect could be executed at initiation time since it was independent of any instance data values. The display shows this assignment taking place to 0.5. See the section “[Coding Efficiency Considerations](#)” on page 334

for more information on how the compiler optimizes the computation by placing portions of it into the initiation phase.

The following 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 statement 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 values displayed is 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: (cell: inv) Device: mp   (line 23 of rule file)
Instance: 1  x: 5.12  y: 51
Values before:
 0: w      0
 1: l      0
 2: i      0
 3: INST()      1
 4: bend_effect  0
 5: 0      0
 6: 0.5    0.5
 7: perim_co(psd,pgate)  3.9999999999999965e-05
 8: perim_in(psd,pgate)  0
 9: AREA(pgate)  3.499999999999989e-11
10: BENDS(pgate)  0
11: temp1      0
12: temp2      0
13: temp3      0      Interpreter called.
98: i
      = INST()
      = 1
99: bend_effect
      = 0
      = 0
100: temp3
      = perim_co(psd,pgate) + perim_in(psd,pgate)
      = 3.999999999999965e-05 + 0
      = 3.999999999999965e-05
100: temp2
      = temp3 + perim_co(psd,pgate)
      = 3.999999999999965e-05 + 3.999999999999965e-05
      = 7.99999999999993e-05
100: temp1
      = temp2 + perim_in(psd,pgate)
      = 7.99999999999993e-05 + 0
      = 7.99999999999993e-05
100: w
      = 0.5 * temp1
      = 0.5 * 7.99999999999993e-05
      = 3.999999999999965e-05
101: l
      = AREA(pgate) / w
      = 3.499999999999989e-11 / 3.999999999999965e-05
      = 8.7499999999999775e-07
102: ?
      BENDS(pgate) != 0
      0 != 0
      FALSE

```

```
Values after:  
0: w      3.9999999999999965e-05  
1: l      8.7499999999999775e-07  
2: i      1  
3: INST()      1  
4: bend_effect  0  
5: 0      0  
6: 0.5    0.5  
7: perim_co(psd, pgate)  3.9999999999999965e-05  
8: perim_in(psd, pgate)  0  
9: AREA(pgate)  3.499999999999989e-11  
10: BENDS(pgate) 0  
11: temp1      7.999999999999993e-05  
12: temp2      7.999999999999993e-05  
13: temp3      3.9999999999999965e-05
```

Each step of the computation 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 “ $1.2999999999999996 < 1.299999999999998$ ” is TRUE.

In evaluating a compound logical expression such as $(a < b \parallel c < d)$, the interpreter does not explicitly perform logical statements 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 statement 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 tempn, where n is a small integer. In the preceding example, the statement on line 100 of the rules is this:

```
100  W = (perimeter_coincide(G, diff) + perimeter_inside(G, diff)) / 2
```

In the trace, this breaks down into three separate steps, each labeled with line number 100, which uses the tempn variables to store intermediate values of sub-expressions. The temporary variables are reused from one statement to the next.

Although this example does not show it, there is a second kind of temporary variable whose name is of the form init<n>, where <n> is a small integer. These variables are used to store the results of sub-expressions 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:

```
// END EXTRACTED DEVICE PROPERTY COMPUTATION DEBUG OUTPUT
```

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 preceding example, only one DEDevice statement contained a DEBUG statement and that statement 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 a DEBUG statement in a second DEDevice statement, the order of the output would depend on whether the DEDevice statements used the same or different device (seed) layers. If the statements used the same device layer, then both initiations would appear, followed by a mixture of evaluations for both types of device. This occurs because DEDevice statements 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. Each DEDevice statement has its own value array.

Related Topics

- [User-Defined Property Computation](#)
- [Property Computation Debugging](#)

Chapter 11

Electrical Rule Checks

Electrical rule check (ERC) operations in Calibre perform geometric checks based upon connectivity. ERC checking can be accomplished in two ways: in an LVS circuit extraction run or using Calibre nmLVS Reconnaissance ERC.

In many respects, ERC is similar to DRC and can be thought of as DRC-type checks performed as a part of LVS. Path checking operations are unique to ERC, and are typical of the types of checks performed in this module. ERC checks have their own results database and summary report.

When performed as a part of LVS circuit extraction, ERC is enabled by specifying [ERC Results Database](#), [ERC Select Check](#), and [LVS Execute ERC YES](#) (the default) in the rule file. Rule checks must also be specified, which often use the [Pathchk](#) layer operation. The [ERC Pathchk](#) statement may also be used, but it is less flexible than Pathchk.

Calibre nmLVS Recon ERC is a stand-alone mode separate from LVS and is not discussed in this chapter. This mode uses the calibre -recon -erc command line.

ERC Statements and Operations	351
Execution of ERC Operations in LVS	354
ERC Results Database.....	359
ERC Auxiliary Files.....	359
ERC Summary Report	361
Trivial Pin Removal Behavior	361
ERC Examples	363
ERC Operations in Calibre nmDRC	364

ERC Statements and Operations

ERC has its own set of specification statements and operations.

These are specification statements related to ERC.

ERC Cell Name	ERC Check Text
ERC Keep Empty	ERC Maximum Results
ERC Maximum Vertex	ERC Path Also
ERC Pathchk	ERC Results Database

[ERC Select Check](#)

[ERC Summary Report](#)

[ERC Unselect Check](#)

[LVS Execute ERC](#)

[LVS Abort On ERC Error](#)

You will notice that many of these statements are similar to ones used for DRC.

These are the ERC-specific layer operations.

[Pathchk](#)

[Device Layer](#)

ERC supports all the layer operations used in typical DRC checks. ERC does not support incremental connectivity, however.

At a minimum, your rule file needs to contain the following for ERC to occur:

- LVS Execute ERC YES (default)
- ERC Select Check *rulecheck*
- At least one rule check (see “[Rule Check Statements](#)” on page 112)
- ERC Results Database *filename*

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

ERC Layer Operations

ERC operations are typically used to check whether polygons on an unlabeled net have a path to some labeled net or not. Often these are power, ground, or pad nets.

Definition of path: A path is a set of electrically-connected objects of specific types. For ERC, a path goes through the source and drain pins of built-in MOS devices (M, MD, ME, MN, MP, LDD, LDDD, LDDE, LDDN, LDDP, and equivalent devices specified with [LVS Device Type](#)) and through the pos and neg pins of built-in resistor devices (R and equivalent devices) by default.

Other devices can be included in the definition of an ERC path by using the [ERC Path Also](#) statement. See “[User-Defined Devices in Path Check Operations](#)” for additional information about extending the definition of path.

The layer operation most frequently used for ERC checks is [Pathchk](#). Results of the Pathchk operation are optimized for the different outputs the operation can produce. The output layer of a Pathchk operation that has an input layer specified preserves hierarchy as much as possible, resulting in more compact data with less promotion. Such operations are commonly used as part of a more complex layer derivation and rule check scheme, where the result layer serves as an input layer for other layer operations.

The input layer parameter is most commonly used for results presentation. However, the output layer of a Pathchk operation that does not use an input layer parameter is optimized for readability and clarity. For each net selected by Pathchk which is a pin of a cell, the output layer of a Pathchk operation without an input layer promotes all polygons to the containing cell and reports in the cell's context as part of the parent net.

In cases where there is a failed Pathchk operation (this is due to a specified net type not being present in the layout design), the rule check entry in the results database shows this comment:

```
rule1
0 0 3 Sep 13 13:08:48 2012
ERROR IN PATHCHK
```

Results generated by the [ERC Pathchk](#) and Pathchk PRINT POLYGONS statements are optimized in the same way since they are also intended for results viewing.

Pathchk and Device Layer operations are layer constructors and can be used to derive layers for rule check output or for other operations. In addition, Pathchk operations may generate auxiliary result files. See “[ERC Auxiliary Files](#)” for more information.

The [Device Layer](#) operation is frequently used for deriving device seed layers in ERC checks.

ERC checking can utilize many Calibre layer operations. Hence, you can use typical DRC layer operations as a part of ERC.

Execution of ERC Operations in LVS

ERC is controlled by the ERC Select Check and LVS Execute ERC specification statements.

Note

 If any rule checks are selected, Calibre nmDRC licenses are acquired in addition to LVS licenses. See “[ERC Licensing](#)” on page 355. For further information about licensing, refer to the [Calibre Administrator’s Guide](#).

Most rule check statements in the rule file are candidates for use in an ERC Select Check statement. Rule checks may contain any layers and are not restricted to Pathchk and Device Layer operations. (For a detailed description of rule check statements, refer to “[Rule Check Statements](#)” on page 112.)

You can suppress execution of all selected rule checks in Calibre LVS by using the LVS Execute ERC NO specification statement. Calibre nmDRC 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 nmDRC.

ERC is performed in the circuit extraction stage of LVS.

The following commands perform ERC in LVS when the appropriate ERC specification statements are present:

- `calibre -spice ... rules`
ERC runs hierarchically.
- `calibre -lvs -hier ... rules`
ERC runs hierarchically when the [Layout System](#) is geometric.
- `calibre -spice ... -flatten ... rules`
ERC runs flat, and the Layout System statement must specify a geometric layout, such as GDSII or OASIS. The transcript is similar to when a flat design is input when using the hierarchical engine.
- `calibre -lvs -tl ... rules`
ERC runs flat, and the Layout System statement must specify a geometric layout, such as GDSII or OASIS.

The -hyper command line option enables hyperscaling during circuit extraction and is recommended for multithreaded runs. The -hyper pathchk keyword enables concurrency of Pathchk operations in HDBs. This keyword can be useful when Pathchk operations consume much time during ERC.

All these commands use Calibre nmDRC licenses in addition to LVS licenses if ERC checks are run.

ERC operations are not performed in Calibre xRC.

ERC Licensing	355
Rule Check Selection in LVS	356
Abort an LVS Run Due to ERC Errors	356
Case Sensitivity of Supply Net Names	356
User-Defined Devices in Path Check Operations	357

ERC Licensing

When ERC is used in flat LVS, flat DRC and flat LVS licenses are used. When used in hierarchical circuit extraction (calibre -spice), four licenses are used: Calibre nmLVS, Calibre nmLVS-H, Calibre nmDRC, and Calibre nmDRC-H.

In -turbo mode, Calibre nmLVS-H uses additional Calibre nmDRC/Calibre nmDRC-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 Calibre nmLVS/Calibre nmLVS-H licenses, and Calibre is run as calibre -spice -turbo, the tool attempts to use four Calibre nmDRC and Calibre nmDRC-H licenses as well. If less than four are available, then the number of threads is reduced, and the remaining Calibre nmLVS-H licenses are released.

If Calibre is invoked as calibre -spice ... -lvs -hier, all Calibre nmDRC and Calibre nmDRC-H licenses are released before LVS comparison begins. Calibre nmDRC 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.

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

Related Topics

[Electrical Rule Checks](#)

Rule Check Selection in LVS

LVS uses a three-step process to enable rule checks.

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.

If no checks are explicitly selected, ERC is not performed.

Abort an LVS Run Due to ERC Errors

You can abort a Calibre nmLVS run due to ERC errors by specifying the ABORT LVS option in an ERC Select Check statement.

```
ERC SELECT CHECK ABORT LVS power_check
LVS EXECUTE ERC YES           //default setting
LVS ABORT ON ERC ERROR YES //default setting
```

If the power_check rule check causes an ERC error, then LVS terminates after performing circuit extraction, device recognition, and ERC operations, but before the circuit comparison step. All errors and warnings generated by circuit extraction are reported normally. A warning is issued in the transcript for each ERC check that generates results, like this:

```
WARNING: Results generated by an ERC check with ABORT LVS
(check <check name>) - aborting.
```

With this message issued at the end of the transcript:

```
ERROR: Results generated by one or more ERC checks with ABORT LVS
(check <check name>) - aborting.
```

If you have numerous [ERC Select Check](#) statements with the ABORT LVS option, you can disable the ABORT LVS behavior by specifying this:

```
LVS ABORT ON ERC ERROR NO
```

All ERC Select Check statements are still used, but ERC errors do not terminate LVS in this case.

Case Sensitivity of Supply Net Names

Text case of nets is not considered by ERC path checking by default. However, this can be changed.

If either the [LVS Compare Case YES](#) or LVS Compare Case NAMES statement is specified, then [Pathchk](#) and [ERC Pathchk](#) operations treat [LVS Power Name](#) and [LVS Ground Name](#) nets

in a case-sensitive manner. This case sensitivity behavior also applies to POWERNAME, GROUNDNAMES, LABELNAME, and BREAKNAME keyword overrides of Pathchk. If case-sensitivity is requested in the rule file, then net names are traced based upon text case.

Note that the [Layout Preserve Case YES](#) statement is not required in order to have meaningful case-sensitive comparisons of layout names and power or ground names because layout names retain their original case. However, if the layout contains names that differ only in their cases, then the Layout Preserve Case YES statement must be specified if you want Calibre not to treat such names as equivalent.

Consider the following example. If the layout contains only name VDD (in uppercase) and the rule file contains these statements:

```
LVS POWER NAME "vdd"
x = PATHCHK !POWER poly
...
poly_no_power {COPY x}
...
```

Calibre considers VDD (as well as any other variant of this string that differs only by case) to be a power name when it executes the Pathchk operation. However, if the rule file contains these statements:

```
LVS COMPARE CASE YES
LVS POWER NAME "vdd"
x = PATHCHK !POWER poly
```

Calibre does not consider VDD to be a power supply name for the purpose of the Pathchk operation. If, instead, the layout has only name vdd (in lowercase), it is considered a power supply name. If the layout has both names vdd and VDD on different nets, an open circuit is detected and one of the two names is chosen randomly (the other net becomes unnamed). However, if the rule file is modified as follows:

```
LAYOUT PRESERVE CASE YES
LVS COMPARE CASE YES
LVS POWER NAME "vdd"
x = PATHCHK !POWER poly
```

both vdd and VDD names are retained, no open circuit is found, but only the name vdd is used by Pathchk as a power supply name.

User-Defined Devices in Path Check Operations

User-defined devices are not considered in ERC path checking by default. However, this behavior can be changed.

The [Pathchk](#) and [ERC Pathchk](#) operations can take into account user-defined devices in addition to built-in devices when tracing paths. User-defined devices are mapped to built-in devices using the [LVS Device Type](#) specification statement.

By default, path checking only considers routes through the S (source) and D (drain) pins of MOS devices (including all “M” types and “LDD” types), and the POS and NEG pins of resistor devices. Capacitors, diodes, and bipolar devices can be added to path checking by using the [ERC Path Also](#) statement. The following table shows the device types, pin names, and tracing modes used by path checking.

Table 11-1. Path Tracing

Built-In Type	Series Pins	Enable Tracing
BIPOLAR	C, E	ERC Path Also BIPOLAR
CAPACITOR	POS, NEG	ERC Path Also CAPACITOR
DIODE	POS, NEG	ERC Path Also DIODE
MOS types	S, D	default
RESISTOR	POS, NEG	default

User-defined devices mapped to built-in devices using LVS Device Type are treated as built-in by the path check functionality in all ways, including unused device filtering, if it is enabled by the EXCLUDE UNUSED option of the path check SVRF statements.

The syntax of the [LVS Device Type](#) specification statement allows mapping of device types and pin names. Here is a conceptual example:

```
LVS DEVICE TYPE built_in_type user_name
[built_in_pin1=name1 built_in_pin2=name2] LAYOUT SOURCE
```

If the SOURCE keyword is specified alone, then the statement applies only to LVS comparison, not to ERC.

The pin map section of LVS Device Type is optional and can be placed anywhere on the line after the *built_in_type* parameter. The following are the built-in devices and pin names.

Table 11-2. Built-In Device Pin Names for Mapping

Built-In Type	Pin Names
BIPOLAR	C, B, E
CAPACITOR	POS, NEG
DIODE	POS, NEG
MOS types	S, G, D
RESISTOR	POS, NEG

If the pin mapping section is omitted, then the user-defined device must already use built-in pin names. The order of the pin names POS and NEG is arbitrary. If the mapping for one of the pins is specified for any device, then the other pins must be mapped as well.

For example, suppose you have a user-defined device with element name RDEV and pins named PLUS and MINUS. This device can be mapped to a resistor for the purposes of path check operations using the following statement (note that the square brackets are literal):

Example 11-1. ERC Check for User-Defined Device

```
LVS DEVICE TYPE RESISTOR RDEV [ POS=PLUS NEG=MINUS ]  
no_power {PATHCHK !POWER}
```

Here is another example:

```
DEV poly_cap cap_layer anode(PLUS) cathode(MINUS)  
...  
// map user-defined capacitors  
LVS DEVICE TYPE CAPACITOR poly_cap [ POS=PLUS NEG=MINUS ]  
  
// include capacitors in the definition of "path"  
ERC PATH ALSO CAPACITOR  
  
// find nets with no power or ground  
no_vdd_vss {PATHCHK !POWER && !GROUND  
}
```

The built-in device type names are not case-sensitive, the user-defined device names have the same case as the [DEVice](#) statements, and pin names are case-insensitive.

ERC Results Database

The ERC Results Database statement generates a results database similar to the ASCII type generated during a DRC run. This database can be loaded into Calibre RVE for cross-probing into your layout, just as with a DRC Results Database.

Note

 You can create only an ASCII format database for an ERC run.

In no case are the same polygons reported in both the child and parent cells.

Related Topics

[ERC Results Database \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[ASCII DRC Results Database Format](#)

ERC Auxiliary Files

ERC operations contain secondary keywords that instruct Calibre to produce auxiliary files. These are in addition to, or instead of, results sent to the ERC results database.

These secondary keywords contain the keyword PRINT. The following sections describe the auxiliary files created with the PRINT keyword, specifically for the [Pathchk](#) operation.

- **PRINT POLYGONS *filename*** — Calibre creates an auxiliary file when you include this syntax in a Pathchk operation. The file contains polygon output of all nets that satisfy the Pathchk condition.

Calibre writes the polygons in DRC ASCII format with the following syntax:

```
PATHCHK condition [ in cell cellname ] [ (layer_name) ]
```

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.

You can perform a PRINT option without writing the information to the [ERC Results Database](#) by performing an AND operation on the ERC operation and an empty layer. In the following example, the rule check X creates file1 but does not write any data to the ERC Results Database (subject to the use of the [ERC Keep Empty](#) statement):

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

- **PRINT NETS *filename*** — Calibre creates an auxiliary file when you include this syntax in a Pathchk operation. The file includes a list of nets that satisfy the Pathchk condition.

Hierarchical Calibre writes the nets to filename in text format with the following syntax:

```
PATHCHK REPORT for layout_primary  
    PATHCHK condition  
        cell name (placement): net1, net2, ...  
        cell name (placement): net1, net2, ...
```

and flat Calibre writes the nets to filename in text format with the following syntax:

```
PATHCHK REPORT for layout_primary  
    PATHCHK condition  
        net1, net2, ...
```

layout_primary — Specifies the top-level cell as defined in the [Layout Primary](#) specification statement.

condition — Specifies the condition of the Pathchk statement, such as “!POWER” or “GROUND && !POWER”.

cell name (placement) — Specifies the name of the cell that contains the reported nets, followed by the placement of that cell, enclosed in parentheses.

net1, net2, netN — Lists the net names or numbers in the specific cell that satisfy the Pathchk condition.

ERC Summary Report

When the ERC Summary Report statement is specified, a report file is generated similar in most respects to the DRC Summary Report.

In cases where there is a failed Pathchk operation (this is due to a specified net type not being present in the layout design), the [ERC Summary Report](#) contains special annotations for rule checks affected by such failed operations.

If there is a failed Pathchk operation in the rule itself, like this:

```
// fails if "pwr" is not in the design
rule1 { PATHCHK POWER metall1 POWERNAME "pwr" }
```

Then the summary report shows this:

```
RULECHECK rule1 ..... NOT EXECUTED DUE TO PATHCHK ERRORS
```

This occurs when the operation has output directly to the [ERC Results Database](#).

If a rule uses a layer that is derived from a failed Pathchk operation, like this:

```
// fails if "A" is not in the design
poly_A = PATHCHK LABELED poly LABELNAME "A"
pins_A = sd TOUCH poly_A

rule2 { NET pins_A "vdd" }
```

Then the summary report shows this:

```
RULECHECK rule2 ..... TOTAL Result Count = 0 (0) FAILED PATHCHK IN
LAYER DERIVATION
```

This also occurs when the complete layer derivation occurs within the scope of the rule check.

The HIER keyword is often desirable when specifying the ERC Summary Report statement.

Trivial Pin Removal Behavior

Trivial pin removal occurs as a part of circuit extraction and can affect ERC path checking.

Circuit extraction removes trivial pins as defined under “[Netlisting of Ports and Pins](#)” on page 282. Trivial pin removal is based, in part, on the tool determining which nets are trivial.

Trivial pins (but not nets) are always removed by default. However, Calibre nmLVS-H considers a pin that connects a device in a cell to a deviceless net (that is, the net is just metal) to be trivial. ERC path checking (as with Pathchk and ERC Pathchk) does not. In SPICE such a pin would appear like this:

```
.subckt cell 1
m1 1 2 3 p
.ends

.subckt top
x1 1 cell
.ends
```

but the ERC path checking view is more like this:

```
.subckt metal 1 $ Fake device representing piece of metal
.ends

.subckt cell 1
m1 1 2 3 p
.ends

.subckt top
x1 1 cell
x2 1 metal
.ends
```

ERC path checking output is polygon-based; hence, the Pathchk and ERC Pathchk operations do not ignore floating (or trivial, when viewed hierarchically) nets by default. Any net that connects to shapes in other cells is not considered floating. However, Pathchk and ERC Pathchk have a NOFLOAT option, which allows them to ignore nets that are not connected to any devices. The ignoring of such floating nets is more consistent with what Calibre nmLVS-H comparison does.

Calibre nmLVS-H comparison is device-based, so any shape that is not on a path connected to any device is not visible to LVS comparison. Such shapes would be part of floating nets, and Calibre nmLVS-H removes most floating nets. There is no option to override this behavior globally. (Note that named, floating nets are not removed if they form initial correspondence points. [LVS Preserve Floating Top Nets YES](#) preserves floating nets at the top level. The Query Server can be used to identify all floating nets after comparison.)

Even when the NOFLOAT option is used for path checking operations, hierarchical LVS comparison and ERC will not always agree on which nets are floating. Because trivial pin removal is based upon such pins being connected to floating nets, there can be differences between ERC and Calibre nmLVS-H results regarding which pins are removed.

A more likely cause for trivial pin removal differences between ERC and LVS applications is unused device filtering. This is discussed under the description of [LVS Filter Unused Option](#) in the *SVRF Manual*.

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 11-2. Basic ERC Path Checks

This 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:

```
LVS POWER NAME vdd

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. Not executed.
```

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 layer 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 LVS circuit extraction does not perform any rule checks or the Pathchk operation.

Assume the same rule file, but the following invocation command instead:

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

Calibre executes rule check E3, but not rule check E1 or E2 because they appear in the ERC Select Check statement.

Output Auxiliary Files with Results Database — This example demonstrates generating auxiliary files and ERC results data with the Pathchk operation.

Assume the following rule file statements exist:

```
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 ercldb.

Output Auxiliary Files Only — This example generates auxiliary files with the Pathchk operation, but no ERC results data.

Assume the following rule file statements exist and that layer 999 is empty:

```
LAYER empty 999 // no geometry on this layer

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 ercldb because the results of rule check X are empty.

See “[ERC Auxiliary Files](#)” on page 359 for details on auxiliary ERC files.

Using ERC Checks to Debug Soft Connection Errors — See “[LVS Softchk Debug Method](#)” on page 290 for information on how to write ERC checks that can be used to debug Softchk errors.

ERC Operations in Calibre nmDRC

ERC-specific operations executed in a DRC run produce empty result layers and a warning is generated. PRINT options are not executed. ERC Select Check statements have no effect in a DRC run.

The [LVS Execute ERC](#) specification statement has no effect in Calibre nmDRC.

Chapter 12

LVS Circuit Comparison

LVS compares a layout design’s topology to a source netlist. This comparison checks that the functional intent of the source schematic is represented in the layout. Calibre nmLVS and Calibre nmLVS-H use SPICE netlists for the comparison. Discrepancies are listed in a text report. A results database called a Mask SVDB Directory may be used together with Calibre RVE and a layout viewer for graphical debugging.

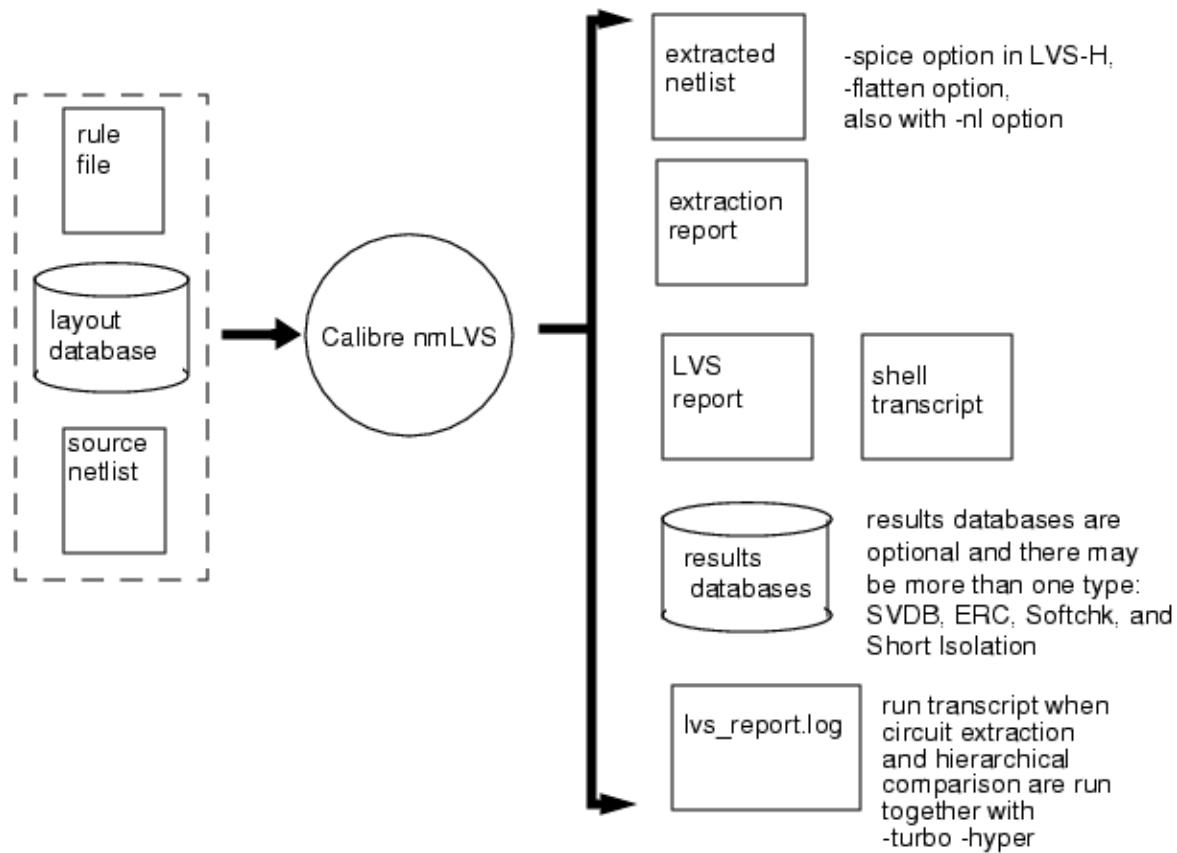
Data Flow in LVS.....	365
LVS Comparison	367
Initial Correspondence Points	369
Component Types	370
Naming Conventions	372
Built-In Device Types	379
Methods for Matching Circuit Elements.....	393
Circuit Comparison Results	394
Device Reduction	396
Device Filtering	428
Nets	430
Trivial Ports, Pins, and Nets.....	438
Logic Gate Recognition.....	439
Pin Swapping	462
Device Property Tracing.....	467

Data Flow in LVS

The inputs to Calibre nmLVS are a rule file, a layout netlist or geometric database, and a source netlist. In LVS-H, an hcell correspondence list is also used and can be specified in its own file, in the rule file, or generated automatically when cell names match between layout and source.

Figure 12-1 shows the LVS data flow for LVS when circuit extraction and netlist comparison are performed in a single step.

Figure 12-1. Single-Step LVS Data Flow



If you run Calibre nmLVS-H with the -spice option, a SPICE layout netlist having a user-specified name is output along with a circuit extraction report. If you run with the -hier option (but not the -spice option) and the Layout System is geometric, then circuit extraction is performed before LVS comparison, and the extracted layout netlist is named automatically. The -flatten option extracts a flat layout netlist.

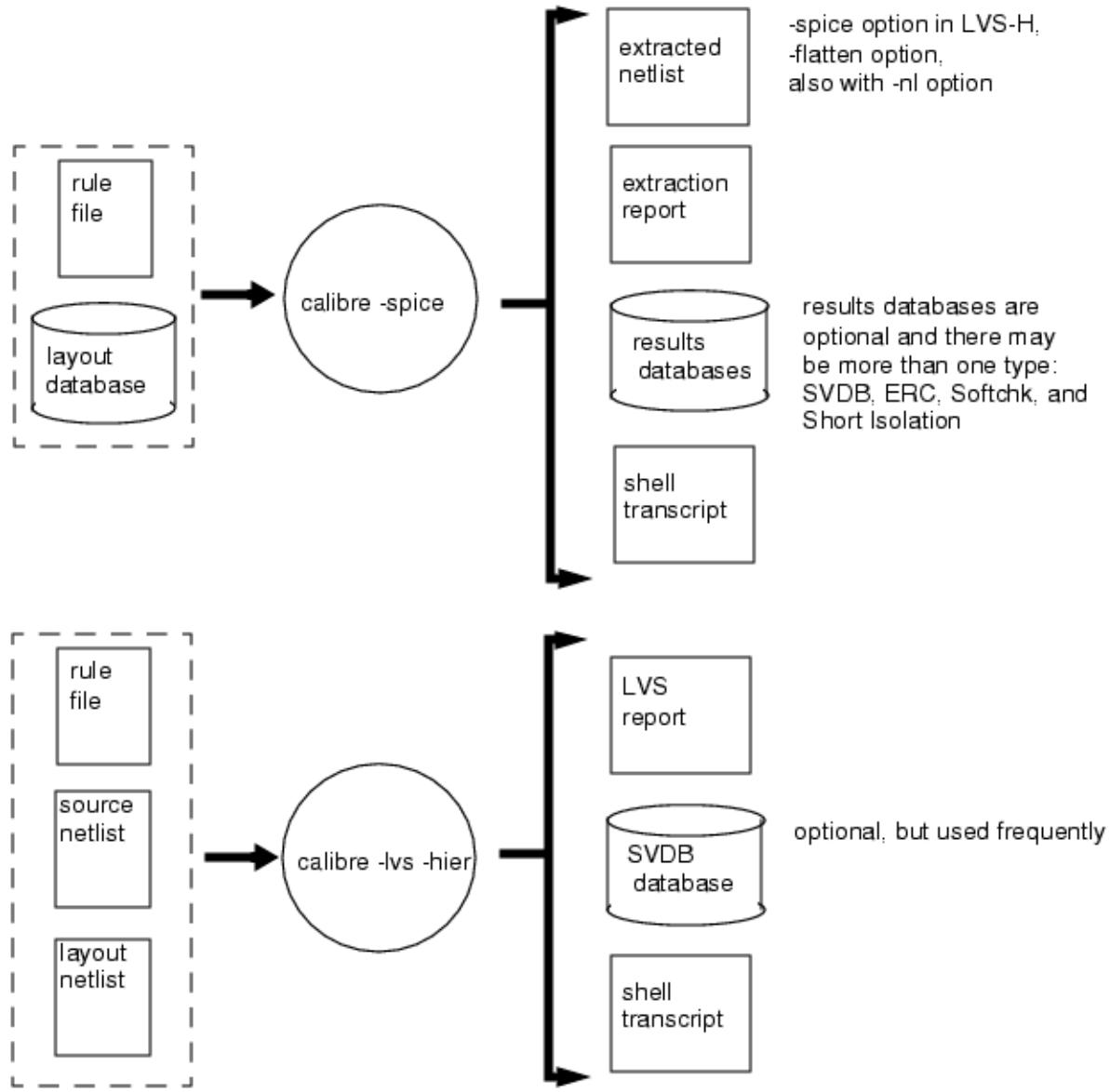
The section “[Setup and Tool Invocation](#)” on page 31 shows the minimum requirements for running the tool set. The section “[Calibre nmLVS and Calibre nmLVS-H Command Line](#)” on page 45 provides examples of command line invocations for the tools. See “[LVS Best Practices](#)” in the *Calibre Solutions for Physical Verification* manual for basic rule file setup instructions.

If you specify ERC checks, short isolation, or soft-connection checking to be run, then LVS produces DRC-style results databases for these applications during circuit extraction. These topics are discussed under “[LVS Circuit Extraction](#)” on page 275.

A netlist comparison run (-lvs option) always produces an [LVS Report](#). “[LVS Results](#)” on page 537 discusses Calibre nmLVS reports and results databases.

Figure 12-2 shows the Calibre nmLVS-H data flow when you perform circuit extraction and comparison as separate steps.

Figure 12-2. Two-Step Calibre nmLVS-H Data Flow



See the chapter “[Hierarchical LVS](#)” on page 473 for information on that topic.

LVS Comparison

LVS applications compare electrical circuits from the specified source netlist and the layout. When the compared circuits are equivalent, LVS establishes 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.

For designs that differ, discrepancies reported in the [LVS Report](#) show the differences between the two circuits. The LVS Report presents discrepancies from the point of view of the source *and* the layout separately. 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 both possible views of the data.

Calibre nmLVS reports discrepancies in terms of incorrect circuit elements, along with additional information that helps classify and locate the errors. To ensure that the matching of different elements does not generate misleading results, internal heuristics are applied. These internal techniques select the best solution among those derived from various methods at different stages of the program.

Calibre nmLVS 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. See “[LVS Results](#)” on page 537 for a description of results files.

You do not need to supply initial correspondence points (see “[Initial Correspondence Points](#)” on page 369 or the [LVS Cpoint](#) statement) between circuit elements, but if you do, the run time usually decreases. Specifying Cpoints is especially useful for net names that do not match between layout and source.

A summary of all rule file specification statements that guide Calibre nmLVS is in the “[LVS Specification Statements](#)” section of the *SVRF Manual*.

Note

 To the extent possible, it is recommended that you run Calibre nmLVS in its default configuration. This means you usually do not need to specify any LVS-specific rule file statements other than [LVS Report](#), [LVS Power Name](#), and [LVS Ground Name](#), unless you have particular needs that can only be met by changing the LVS defaults. A list of LVS specification statement settings appears in the LVS Report, including the defaults if you leave them unchanged.

Hyperscaling with Netlist Comparison

Hyperscaling is enabled in Calibre nmLVS-H through the `-hyper` command line option. The `-hyper` option can only be specified if the `-turbo` option is also used and improves hardware utilization and run times. (See [-hyper](#) in the *Calibre Administrator’s Guide* for additional details.)

When the -hyper cmp option is used, LVS comparison processes are performed simultaneously with circuit extraction, which further improves hardware utilization and run times. To enable the hyperscaling comparison run optimizations, you can use these command lines:

```
# MT mode
calibre -turbo -hyper cmp -lvs -hier -hcell hcells rules

# MTflex mode
calibre -turbo -hyper cmp -remotefile config.txt -lvs -hier -hcell hcells\
rules
```

These trigger circuit extraction when your [Layout System](#) is geometric. Otherwise, they perform just LVS comparison.

The -hyper command line option also supports a remote option, which enables performance enhancements when you are using remote hosts on a server network. When “-hyper cmp remote” is specified, this causes the source netlist to be read from the start of the run. This is earlier than when the cmp option is used alone, and further improves run times.

When hyperscaling is enabled, a transcript of the comparison portion of the run is written to a file of the form *lvs_report_name.log*, where *lvs_report_name* is the name of the [LVS Report](#). Important comparison messages are still echoed to stdout, but the full set of messages are written to the *lvs_report_name.log* file. See “[Hierarchical Comparison with Hyperscaling Transcript](#)” on page 543 for details.

The [LVS Abort On ERC Error](#) YES, [LVS Abort On Supply Error](#) YES, and [LVS Abort On Softchk](#) YES specification statements cause LVS to abort after the source netlist is read but before comparison when hyperscaling is enabled.

Initial Correspondence Points

Calibre nmLVS uses certain 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 (also called anchor points). Initial correspondence points assist in matching layout and source design topologies and make that process more efficient.

LVS does not require that initial correspondence points exist. However, it is recommended that you specify initial correspondence points (using text) on ports of the top-level layout cell. It is also recommended to do this for hcell ports. You can specify correspondence points explicitly in the rule file by using the [LVS Cpoint](#) specification statement. Accurate initial correspondence points can help to improve performance.

LVS matches elements that form initial correspondence points by name. These matches are presumed to be correct, even if the user has mistakenly named an object.

System-generated names do not form initial correspondence points. Neither do names specified with the [LVS Non User Name](#) statement. User-given names that appear on more than one layout

net, or more than one layout instance, or more than one layout port are not used as initial correspondence points (although they are reported). The same is true on the source side.

Initial correspondence points do not prevent logic injection, formation of logic gates, series and parallel device reduction and similar transformations when they appear in lower level cells in hierarchical LVS. Initial correspondence points do prevent such transformations in the top level cell.

Related Topics

[User-Given Names](#)

Component Types

An LVS component type is a name that uniquely identifies the electrical or logical function of a layout or source instance. Calibre nmLVS 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 Calibre nmLVS 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(g) srcdrn(s) srcdrn(d) bulk(b)
```

specifies a MOS transistor device with component type MP.

SPICE netlist — The component types of SPICE elements are described in section “[Element Statements](#)” on page 677.

The [LVS Compare Case](#) statement, in conjunction with the [Layout Case](#) and [Source Case](#) statements, controls case sensitivity of LVS comparison of net, instance, port, component, and model names. LVS is case-insensitive by default.

Component Subtypes

An LVS component subtype (model) is an optional name that classifies, together with the component type, the electrical or logical function of a layout or source instance. Component subtypes are not necessary to match source instances to layout instances. Calibre nmLVS 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 Calibre nmLVS uses to determine component subtypes of instances.

Mask layout — The component subtype of an extracted layout device is equivalent to the value of the optional *model_name* argument in the corresponding [Device](#) operation. For example, this operation:

```
device C(PM) cap poly(pos) metal(neg)
```

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 “[Element Statements](#)” on page 677.

The [LVS Compare Case](#) statement, in conjunction with the [Layout Case](#) and [Source Case](#) statements, controls case sensitivity of LVS comparison of net, instance, port, component, and model names. LVS is case-insensitive by default.

Required Device Model Names

Device element names MP, MN, ME, MD, Q, and D must be accompanied by a model name in the rule file Device statement that is identical to the model name used in the source netlist. This is required for the netlist-to-netlist comparison flow.

There are two exceptions to this rule:

- 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.

Naming Conventions

LVS uses a number of naming conventions that appear in the LVS Report, run transcript, and Mask SVDB Directory results. It is helpful to be familiar with these conventions when debugging results.

Instance Pins and Pin Names	372
Net and Instance Names	373
Ports and Port Names	374
Power and Ground Nets	374
User-Given Names	374
Case-Sensitive Handling of Names.....	375

Instance Pins and Pin Names

Calibre nmLVS 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 — Calibre nmLVS specifies the pin names of extracted layout devices in the rule file, or by default convention.

- **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 device recognition and LVS comparison. Each 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 for LVS comparison if you conform to the pin names specified in “[Built-In Device Types](#)” on page 379.

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.

- **User-defined devices** — [Device](#) operations in the rule file can define user devices. Here, user-defined means for the purpose of device recognition, and the device is not of a built-in type. Such devices have user-defined pin names. 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 379.

SPICE netlist — Pin names of SPICE elements are described in the section “[Element Statements](#)” on page 677.

Pin Filtering

Calibre nmLVS 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.

Calibre nmLVS filters out missing pins to allow higher-level layout components to have pins, such as power and ground pins, 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 matched to each other correctly. 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.

Calibre nmLVS classifies the names of power or ground pins if they are specified in the [LVS Power Name](#) or [LVS Ground Name](#) specification statements, respectively.

Net and Instance Names

Net and instance names are specified in geometric layout and SPICE netlists. They are obtained differently in the two design types.

Mask layout — Calibre nmLVS obtains layout net names from the text object values on layout shapes and paths in the top-level cell (by default). These values are assigned as names to nets in the circuit extraction process performed by LVS. Layout instances cannot be named in Mask LVS because GDS, OASIS, and similar layout databases do not support such naming.

SPICE netlist — Calibre nmLVS obtains net names from the node names in the netlist. Instance names are the element names in the netlist.

Related Topics

[User-Given Names](#)

[Power and Ground Nets](#)

Ports and Port Names

Design ports and port names are specified in geometric layout and SPICE netlists.

Mask layout — Calibre nmLVS specifies a geometric layout port with the [Port Layer Text](#), [Port Layer Polygon](#), and [Port Layer Merged Polygon](#) specification statements in the rule file.

SPICE netlist — Calibre nmLVS specifies design ports 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

Calibre nmLVS 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.
- 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.

User-Given Names

The nets, instances, and ports of layout and source databases can have user-given names, system-generated names, or both. Calibre nmLVS reports differences between user-given names of layout and source elements.

Matching user-given names establish [Initial Correspondence Points](#) in LVS.

Calibre nmLVS 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\$. Names with slash (/) characters qualify as user-given, but node numbers replace them for nets and ports in extracted netlists.
- **SPICE netlist** — A *node* name qualifies as user-given if it contains at least one non-numeric character (letter) and does not contain more than one slash (/) character (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 that is not a digit or the equals (=) sign. (The first character in SPICE element names is always the SPICE element type.) Also, the name must not contain any / characters. 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 Siemens EDA 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.

Case-Sensitive Handling of Names

Layout and source names are handled in a case-insensitive manner by default. Various aspects of case-sensitive name handling can be controlled from the rule file.

SPICE netlist handling has two distinct phases: netlist parsing on read in and netlist comparison in LVS. Netlist parsing affects both LVS and other tools that take SPICE netlists as input, such as Calibre PERC. These phases are controlled separately by distinct specification statements.

You can control the case sensitivity of the parsing of the input netlists through the [Layout Case](#) and [Source Case](#) specification statements. By default, they are both set to NO. When NO is specified, the SPICE parser internally manipulates the case of a netlist so the case is uniform throughout. If you set either of them to YES, then the SPICE netlist parser treats the associated netlist in a case-sensitive manner. Netlist elements whose names differ only in case are treated as distinct elements.

You may want to treat an input netlist as case-sensitive when you have net names that differ only by case that are supposed to be treated as distinct nets. Suppose you have the following:

```
$$ SOURCE
.SUBCKT BLOCK SIG sig Y      $$ SIG and sig are intentionally distinct nets
...
$$ LAYOUT
.SUBCKT BLOCK 1 2 3
...
```

Because SIG and sig are supposed to be different nets, you would want to specify Source Case YES in the rule file to ensure these nets compare properly.

If you want to perform LVS comparison in a case-sensitive manner, you need to specify [LVS Compare Case](#) with an option other than NO. You should also typically set Layout Case and Source Case to YES, otherwise the text cases of the input netlists are not preserved, and the case comparison would not have the intended results. Neither the Layout Case nor Source Case statements cause case-sensitive comparison by themselves.

The following table shows five potentially useful combinations for the case handling statements. The three combinations not shown are not generally useful in most design flows.

Table 12-1. Case-Sensitive Comparison Settings

Layout Case	Source Case	LVS Compare Case	Possible Use Scenarios and Comments
NO	NO	NO	This is the default. Netlists are parsed and compared in a case-insensitive manner.
YES	YES	YES	<p>Compares net names, port names, device component types and subtypes, string properties, and instance names in a case-sensitive manner. This can affect things like hcell selection, property tracing, unused device filtering, logic gate recognition, and property mapping.</p> <p>The LVS Compare Case NAMES, TYPES, SUBTYPES, and VALUES keywords control subsets of elements governed by the YES keyword.</p> <p>Calibre PERC effects: This configuration causes certain settings in the Calibre PERC rule file to become case-sensitive. See “Case Sensitivity for Names” in the <i>Calibre PERC User’s Manual</i>.</p> <p>You can use LVS Compare Case Strict YES to force this rule file configuration when LVS Compare Case YES is specified.</p>

Table 12-1. Case-Sensitive Comparison Settings (cont.)

Layout Case	Source Case	LVS Compare Case	Possible Use Scenarios and Comments
YES	NO	NO	<p>May be useful when there are net names in the layout netlist that differ only by case that need to be treated as separate nets. This might apply in parasitic extraction or third-party flows.</p> <p>If circuit extraction is used to provide the layout netlist, then Layout Preserve Case YES must be used.</p>
NO	YES	NO	<p>May be useful when there are net names in the source netlist that differ only by case that need to be treated as separate nets. This might apply in parasitic extraction or third-party flows.</p>
NO	NO	YES	<p>Netlist effects: This combination has no effect on netlist names because both netlists are manipulated internally to have uniform and identical case. The same is true for the NAMES, TYPES, SUBTYPES, and VALUES keywords.</p> <p> Caution: If device properties are traced, do not use this combination of keywords as this may cause improperly valid property comparison when Trace Property is used with such values.</p> <p>Hcell effects: This combination causes the hcell list to become case-sensitive for both circuit extraction and LVS comparison modules. Cell names that do not match the hcell list by case are not treated as hcells.</p> <p>Circuit extraction effects: A possible use for this combination is to cause ERC net name tracing to become case-sensitive. See “Case Sensitivity of Supply Net Names” on page 356 for more information. This combination affects selection of LVS Power Name and LVS Ground Name nets during circuit extraction. This can affect unused device filtering for Pathchk, ERC Pathchk, and Device Layer operations.</p> <p>See “Case Sensitivity for Names” in the <i>Calibre PERC User’s Manual</i> for details related to Calibre PERC.</p>

For details regarding hierarchical netlist naming conventions, see “[Selection of Extracted Netlist Names](#)” on page 278.

Undesirable Effects of Case-Sensitive Statements

Setting one of Layout Case or Source Case to YES can cause undesirable side-effects.

One undesirable side-effect of enabling case sensitivity is the potential loss of initial correspondence points (see “[Initial Correspondence Points](#)” on page 369 for details). Suppose you have nets “a” and “A” in your source and layout respectively. If these nets form initial correspondence points for case-insensitive comparison, they do not form initial correspondence points when case sensitivity is turned on. This results in a loss of performance.

For netlists where text case is not uniform for given elements, turning on case sensitivity can cause errors as the netlists are parsed. For example, suppose you have the following in your source netlist:

```
.SUBCKT inst pos neg
.ENDS
...
X3 vss vdd INST
```

The subcircuit names `inst` and `INST` parse correctly if Source Case NO is set. However, if Source Case YES is set, this causes an error if there is no source subcircuit called `INST`. Such case discrepancies in netlists can cause difficulties when case sensitivity is turned on.

Be aware that your [Layout Primary](#) and [Source Primary](#) cell names are treated as case-sensitive when you set Layout Case or Source Case to YES. If there is a case mismatch between the rule file and netlist primary cell names, this causes an error.

Note

 Using Layout Case NO and Source Case NO with LVS Compare Case YES is problematic in Calibre PERC. For this reason, it is a good practice to specify LVS Compare Case Strict YES when LVS Compare Case YES is used in such flows.

The previous paragraph’s guidance is especially important when tracing device properties in LVS. Using LVS Compare Case with settings other than NO or NAMES without also specifying case-sensitive handling of the netlists can result in improper device property comparison results.

Built-In Device Types

Certain SPICE device component types are recognized as built-in to the LVS system. These component types receive special processing that other components do not.

[Table 12-2](#) lists the built-in device types and their corresponding component type values. The section “[Component Types](#)” on page 370 shows how component types are determined in the source circuit and in the layout.

Table 12-2. Built-in Device Types for LVS Comparison

Device	Component Type
CMOS N transistor	MN ¹
CMOS P transistor	MP ¹
NMOS enhancement transistor	ME ¹
NMOS depletion transistor	MD ¹
MOS generic transistor	M ²
CMOS LDD N transistor	LDDN ²
CMOS LDD P transistor	LDDP ²
NMOS LDD enhancement transistor	LDDE ²
NMOS LDD depletion transistor	LDDD ²
MOS LDD generic transistor	LDD ²
Resistor	R ¹
Capacitor	C ¹
Diode	D ¹
Bipolar transistor	Q ¹
Jfet transistor	J ²
Inductor	L ²
Voltage source	V ²

1. built-in devices for both device recognition and LVS comparison, and having hard-coded pin names.
2. built-in devices for LVS comparison when recognized pin names are used.

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 module governed by the [Device](#) operation.

The devices that are built-in for device recognition and LVS comparison are indicated with a superscript ¹ in [Table 12-2](#). Such devices have hard-coded pin names that cannot be overridden. Devices that are not built-in for device recognition do not have hard-coded pin names. Devices that have the superscript ² behave like built-in devices for LVS comparison if you use recognized pin names. However, if specified with user-defined pin names, then these devices are user-defined. This behavior is described in detail later.

To perform the special processing described in the following sections, Calibre nmLVS 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 372 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.

Capacitors

Calibre nmLVS performs the following for built-in capacitors: series capacitor reduction, parallel capacitor reduction, pin swapping if requested, processing of soft substrate pins, and pin names are always case-insensitive.

In order to behave as built-in devices, capacitors (component type C and equivalent types specified in an [LVS Device Type](#) specification statement) must have at least two pins conforming to the pin names in [Table 12-3](#). In addition, they can have any number of additional pins with arbitrary names. The third pin typically represents a bulk connection and is called SUB by the device extractor by default, but any name can be used in the Device statement. The optional pins may represent one or more bulk connections or they may be used for any other purpose.

Table 12-3. Capacitor Required Pin Names

Pin	Pin Name
capacitor positive pin	POS or P
capacitor negative pin	NEG or N
optional pins	any names

If you specify capacitor devices in your Device statements differently from what [Table 12-3](#) 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.

The following specification statements control special processing functions used by Calibre nmLVS for capacitors:

- [LVS Annotate Devices](#)
- [LVS Apply Unused Filter](#)
- [LVS Builtin Device Pin Swap](#)
- [LVS Device Type](#)
- [LVS Discard Pins By Device](#)
- [LVS Exact Subtypes](#)
- [LVS Filter](#)
- [LVS Filter Unused Capacitors](#)
- [LVS Filter Unused Option](#)
- [LVS Ignore Device Pin](#)
- [LVS Map Device](#)
- [LVS Netlist Property](#)
- [LVS Property Map](#)
- [LVS Property Resolution Maximum](#)
- [LVS Push Devices](#)
- [LVS Reduce](#)
- [LVS Reduce Parallel Capacitors](#)
- [LVS Reduce Series Capacitors](#)
- [LVS Reduction Priority](#)
- [LVS Report Pinswapped](#)
- [LVS Select Device By Seed Layer](#)
- [LVS Show Seed Promotions](#)
- [LVS Soft Substrate Pins](#)
- [LVS Spice Rename Parameter](#)
- [LVS Spice Replicate Devices](#)

- [LVS Strict Subtypes](#)
- [LVS Unselect Device By Seed Layer](#)

Diodes

Calibre nmLVS performs the following for diodes: parallel diode reduction, processing of soft substrate pins, and pin names are always case-insensitive.

In order to behave like built-in devices, diodes (component type D and equivalent types specified in an [LVS Device Type](#) specification statement) must have at least two pins that conform to the conventions in [Table 12-4](#). In addition, they can have any number of additional pins with arbitrary names. The third pin typically represents a bulk connection and is called SUB by the device extractor by default, but any name can be used in the Device statement. The optional pins may represent one or more bulk connections or they may be used for any other purpose.

Table 12-4. Diode Required Pin Names

Pin	Pin Name
diode positive pin	POS or P
diode negative pin	NEG or N
optional pins	any names

If you specify diode devices in your Device statements differently from what [Table 12-4](#) 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.

The following specification statements control special processing functions used by Calibre nmLVS for diodes:

- [LVS Annotate Devices](#)
- [LVS Apply Unused Filter](#)
- [LVS Device Type](#)
- [LVS Discard Pins By Device](#)
- [LVS Exact Subtypes](#)
- [LVS Filter](#)
- [LVS Filter Unused Diodes](#)
- [LVS Filter Unused Option](#)
- [LVS Ignore Device Pin](#)

- [LVS Map Device](#)
- [LVS Netlist Property](#)
- [LVS Property Map](#)
- [LVS Property Resolution Maximum](#)
- [LVS Push Devices](#)
- [LVS Reduce](#)
- [LVS Reduce Parallel Diodes](#)
- [LVS Select Device By Seed Layer](#)
- [LVS Show Seed Promotions](#)
- [LVS Soft Substrate Pins](#)
- [LVS Spice Rename Parameter](#)
- [LVS Spice Replicate Devices](#)
- [LVS Strict Subtypes](#)
- [LVS Unselect Device By Seed Layer](#)

MOS Transistors

MOS regular transistors recognized as built-in devices receive special default processing by Calibre nmLVS as follows: logic gate recognition, logic injection, parallel transistor reduction, split-gate reduction, shorting of equivalent nodes in split gates, filtering of unused transistors, source/drain pin swapping, processing of soft substrate pins, and pin names are always treated as case-insensitive. Series MOS reduction and processing of soft substrate pins can also be enabled.

In order to behave like built-in devices, MOS transistors (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—as shown in [Table 12-5](#). In addition, they may have *any number* of additional pins with arbitrary names when appropriately specified with LVS Device Type. However, MOS devices with more than four pins are not subject to logic injection and are not processed by [LVS Short Equivalent Nodes MATRIX](#).

The fourth pin typically represents bulk connection and by convention is called B. The rule file Device statement syntax enforces this convention, but the LVS circuit comparison module does not. The optional pins may represent one or more bulk connections or they may be used for any other purpose.

Table 12-5. MOS Transistor Required Pin Names

Pin	Pin Name
MOS transistor gate	G or GATE
MOS transistor source	S or SOURCE
MOS transistor drain	D or DRAIN
optional pins	any names (fourth pin must be B to pass Device M* or LDD* syntax check)

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 LDD transistors are processed by default as follows, if they conform to [Table 12-5](#) pin conventions: logic gate recognition, parallel transistor reduction, split-gate reduction, filtering of unused transistors, and pin names are always treated as case-insensitive. MOS devices M, LDDN, LDDP, LDDE, LDDD, and LDD that do not have the required pin names 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 nmLVS for MOS transistors:

- [LVS Annotate Devices](#)
- [LVS Apply Unused Filter](#)
- [LVS Builtin Device Pin Swap](#)
- [LVS Builtin MOS NRD_NRS](#)
- [LVS Device Type](#)
- [LVS Discard Pins By Device](#)
- [LVS Exact Subtypes](#)
- [LVS Filter](#)
- [LVS Filter Unused MOS](#)
- [LVS Filter Unused Option](#)
- [LVS Ignore Device Pin](#)
- [LVS Inject Logic](#)

- [LVS Map Device](#)
- [LVS MOS Swappable Properties](#)
- [LVS Netlist Property](#)
- [LVS Property Map](#)
- [LVS Property Resolution Maximum](#)
- [LVS Push Devices](#)
- [LVS Recognize Gates](#)
- [LVS Recognize Gates Tolerance](#)
- [LVS Reduce](#)
- [LVS Reduce Parallel MOS](#)
- [LVS Reduce Semi Series MOS](#)
- [LVS Reduce Series MOS](#)
- [LVS Reduce Split Gates](#)
- [LVS Reduction Priority](#)
- [LVS Report Pinswapped](#)
- [LVS Reverse WL](#)
- [LVS Select Device By Seed Layer](#)
- [LVS Short Equivalent Nodes](#)
- [LVS Show Seed Promotions](#)
- [LVS Soft Substrate Pins](#)
- [LVS Spice Conditional LDD](#)
- [LVS Spice Implied MOS Area](#)
- [LVS Spice Rename Parameter](#)
- [LVS Spice Replicate Devices](#)
- [LVS Spice Strict WL](#)
- [LVS Split Gate Ratio](#)
- [LVS Strict Subtypes](#)
- [LVS Unselect Device By Seed Layer](#)

Resistors

Calibre nmLVS 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.

In order to behave like built-in devices, resistors (component type R and equivalent types specified in an [LVS Device Type](#) specification statement) must have at least two pins conforming to the pin names in [Table 12-6](#). In addition, they can have any number of additional pins with arbitrary names. The third pin typically represents a bulk connection and is called SUB by the device extractor by default, but any name can be used in the Device statement. The optional pins may represent one or more bulk connections or they may be used for any other purpose.

Table 12-6. Resistor Required Pin Names

Pin	Pin Name
resistor positive pin	POS or P
resistor negative pin	NEG or N
optional pins	any names

If you specify resistor devices in your Device statements differently from what [Table 12-6](#) 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.

The following specification statements control special processing functions used by Calibre nmLVS for resistors:

- [LVS Annotate Devices](#)
- [LVS Apply Unused Filter](#)
- [LVS Builtin Device Pin Swap](#)
- [LVS Device Type](#)
- [LVS Discard Pins By Device](#)
- [LVS Exact Subtypes](#)
- [LVS Filter](#)
- [LVS Filter Unused Resistors](#)
- [LVS Filter Unused Option](#)
- [LVS Ignore Device Pin](#)
- [LVS Map Device](#)

- [LVS Netlist Property](#)
- [LVS Property Map](#)
- [LVS Property Resolution Maximum](#)
- [LVS Push Devices](#)
- [LVS Reduce](#)
- [LVS Reduce Parallel Resistors](#)
- [LVS Reduce Series Resistors](#)
- [LVS Reduction Priority](#)
- [LVS Report Pinswapped](#)
- [LVS Select Device By Seed Layer](#)
- [LVS Show Seed Promotions](#)
- [LVS Soft Substrate Pins](#)
- [LVS Spice Rename Parameter](#)
- [LVS Spice Replicate Devices](#)
- [LVS Strict Subtypes](#)
- [LVS Unselect Device By Seed Layer](#)

Bipolar Transistors

Calibre nmLVS 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.

In order to behave like built-in devices, 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, but the LVS circuit comparison module does not. The optional pins can represent one or more substrate connections or they can be used for any other purpose. [Table 12-7](#) lists the pin names:

Table 12-7. Bipolar Transistor Required Pin Names

Pin	Pin Name
Q transistor collector	C
Q transistor base	B

Table 12-7. Bipolar Transistor Required Pin Names (cont.)

Pin	Pin Name
Q transistor emitter	E
other pins	any names (fourth pin must be S to pass Device Q statement syntax check)

If you specify bipolar devices in your Device statements differently from what [Table 12-7](#) 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.

The following specification statements control special processing functions used by Calibre nmLVS for bipolar transistors:

- [LVS Annotate Devices](#)
- [LVS Apply Unused Filter](#)
- [LVS Builtin Device Pin Swap](#)
- [LVS Device Type](#)
- [LVS Discard Pins By Device](#)
- [LVS Exact Subtypes](#)
- [LVS Filter](#)
- [LVS Filter Unused Bipolar](#)
- [LVS Filter Unused Option](#)
- [LVS Ignore Device Pin](#)
- [LVS Map Device](#)
- [LVS Netlist Property](#)
- [LVS Property Map](#)
- [LVS Property Resolution Maximum](#)
- [LVS Push Devices](#)
- [LVS Reduce](#)
- [LVS Reduce Parallel Bipolar](#)
- [LVS Report Pinswapped](#)

- [LVS Select Device By Seed Layer](#)
- [LVS Show Seed Promotions](#)
- [LVS Soft Substrate Pins](#)
- [LVS Spice Rename Parameter](#)
- [LVS Spice Replicate Devices](#)
- [LVS Strict Subtypes](#)
- [LVS Unselect Device By Seed Layer](#)

JFET Transistors

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.

Table 12-8. JFET Transistor Recognized Pin Names

Pin	Pin Name
J transistor gate	G or GATE
J transistor source	S or SOURCE
J transistor drain	D or DRAIN
other pins	any names

Calibre nmLVS performs the following for JFETs that conform to [Table 12-8](#): processes soft substrate pins; pin names are always case-insensitive.

If you specify J devices in your Device statements differently from what [Table 12-8](#) 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.

The following specification statements control special processing functions used by Calibre nmLVS for Jfet transistors:

- [LVS Annotate Devices](#)
- [LVS Discard Pins By Device](#)
- [LVS Exact Subtypes](#)
- [LVS Filter](#)

- [LVS Ignore Device Pin](#)
- [LVS Map Device](#)
- [LVS Property Map](#)
- [LVS Reduce](#)
- [LVS Soft Substrate Pins](#)
- [LVS Spice Rename Parameter](#)
- [LVS Strict Subtypes](#)

Inductors

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.

Table 12-9. Inductor Recognized Pin Names

Pin	Pin Name
inductor positive pin	POS or P
inductor negative pin	NEG or N
optional pins	any names

Calibre nmLVS performs the following for inductors that conform to [Table 12-9](#): processes soft substrate pins; pin names are always case-insensitive.

If you specify L devices in your Device statements differently from what [Table 12-9](#) 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.

The following specification statements control special processing functions used by Calibre nmLVS for inductors:

- [LVS Annotate Devices](#)
- [LVS Discard Pins By Device](#)
- [LVS Exact Subtypes](#)
- [LVS Filter](#)
- [LVS Ignore Device Pin](#)
- [LVS Map Device](#)

- [LVS Property Map](#)
- [LVS Reduce](#)
- [LVS Soft Substrate Pins](#)
- [LVS Spice Rename Parameter](#)
- [LVS Strict Subtypes](#)

Voltage Sources

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.

Table 12-10. Voltage Source Recognized Pin Names

Pin	Pin Name
voltage source positive pin	POS or P
voltage source negative pin	NEG or N
optional pins	any names

Calibre nmLVS performs the following for voltage sources that conform to [Figure 12-10](#): processes soft substrate pins; pin names are always case-insensitive.

If you specify V devices in your Device statements differently from what [Table 12-10](#) 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.

The following specification statements control special processing functions used by Calibre nmLVS for voltage sources:

- [LVS Annotate Devices](#)
- [LVS Exact Subtypes](#)
- [LVS Filter](#)
- [LVS Ignore Device Pin](#)
- [LVS Map Device](#)
- [LVS Property Map](#)
- [LVS Reduce](#)

- [LVS Report Pinswapped](#)
- [LVS Soft Substrate Pins](#)
- [LVS Spice Rename Parameter](#)
- [LVS Strict Subtypes](#)

X+ Devices

X+ devices are MOS transistors having a subtype beginning with the letter X and at least one more character.

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.
```

X+ devices prevent the formation of logic gates, and they have a limiting effect on split-gate reduction (see “[Split Gate Reduction](#)” on page 405). 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.

Related Topics

[X+ Transistors](#)

MS and MF Schematic Devices

MS and MF are special device types 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.

Calibre nmLVS 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.

Calibre nmLVS 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 379. 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 370 for more details on specifying component types in the schematic.

Methods for Matching Circuit Elements

Calibre nmLVS iterates between two methods of matching elements: signature-based hashing and tracing.

- **Signature-based method** — Calibre nmLVS assigns signatures to nets and instances in both circuits according to their type and connections. Calibre nmLVS then hashes circuit elements according to the following considerations:
 - Signature of the element
 - Signatures of elements in nearby environments
 - Presence of previously matched elements in their environmentsThe 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** — Calibre nmLVS uses previously matched elements as initial correspondence points. Starting from these correspondence points, Calibre nmLVS traces both circuits one step at a time. Calibre nmLVS establishes a correspondence between elements that can be uniquely matched at each step. The tracing continues until Calibre nmLVS does the following:
 - Matches all elements
 - Detects discrepancies that prevent further tracing
 - Detects interchangeable parts of the circuit that prevent further tracing

Calibre nmLVS repeats both methods until all elements are matched or further elements can be matched. In the latter case, Calibre nmLVS internally tries to correct some of the errors. If errors can be corrected, then more elements are matched and the process is repeated.

Related Topics

[LVS Comparison](#)

Circuit Comparison Results

Calibre nmLVS classifies elements of both compared circuits (nets, instances, and ports) into one of 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. 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 be classified as incorrect when viewed from the layout perspective.
- **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 nmLVS 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. See “[LVS Report](#)” on page 549 for more information.

Ambiguity Resolution [394](#)

Ambiguity Resolution

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.

Calibre nmLVS uses named nets, instances, and ports as initial correspondence points to resolve ambiguous situations. In addition, Calibre nmLVS 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 32.

Calibre nmLVS 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, Calibre nmLVS arbitrarily matches some elements that cannot be resolved otherwise. Calibre nmLVS 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, Calibre nmLVS produces discrepancies at

a later stage. In this case, you should assign layout text 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

Calibre nmLVS 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 behavior 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 things to be aware of:

Initial correspondence points — Initial correspondence points prevent series and parallel reduction in the top-level cell. Initial correspondence points do not interfere with device reduction in lower-level cells for LVS-H.

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 occurs with trivial ports (those not connected to any devices or to nets higher up in the design).

Filtering — Device filtering occurs after reduction.

Device reduction occurs after circuit transformation but before the circuit comparison phase.

Parallel and Series Device Reduction	397
Generic Device Reduction	398
Parallel MOS Transistor Reduction	398
Series MOS Transistor Reduction	400
Semi-Series MOS Transistor Reduction	402
Split Gate Reduction	405
Parallel Capacitor Reduction	415
Series Capacitor Reduction	416
Parallel Resistor Reduction	417
Series Resistor Reduction	419
Parallel Diode Reduction	420
Parallel Bipolar Transistor Reduction	421
Missing and Unknown Property Values	423
Case Comparison of String Properties	423
Device Reduction Programs	424
Tolerance in Device Reduction	424

Parallel and Series Device Reduction

When reducing devices, Calibre nmLVS reduces series structures first, then parallel, then series, then parallel, and so forth until all devices are reduced.

For device configurations that could allow either series or 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 when either form of reduction is possible for a given device configuration.

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 12-3](#), or as series with an internal net 2.

Figure 12-3. Parallel and Series Device Reduction



In general, two parallel devices can also be reduced in series when the following conditions are met:

- The two pins that form the parallel connection are swappable with each other.
- The two pins that 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.

If the LVS Reduction Priority algorithm cannot reduce a set of devices by the specified method, then the devices are treated as separate and are not reduced.

Generic Device Reduction

Calibre nmLVS can reduce built-in or generic, user-defined devices through the LVS Reduce statement. Other statements exist for reduction of specific types of devices in certain connection configurations (parallel, series, and split gate).

Generic device reduction can be made cell-specific by using the [LVS Reduce CELL LIST](#) option. When this is used, reduction of specified devices occurs only in specified cells.

Parallel MOS Transistor Reduction

Parallel MOS transistors are reduced by default during comparison of built-in MOS devices.

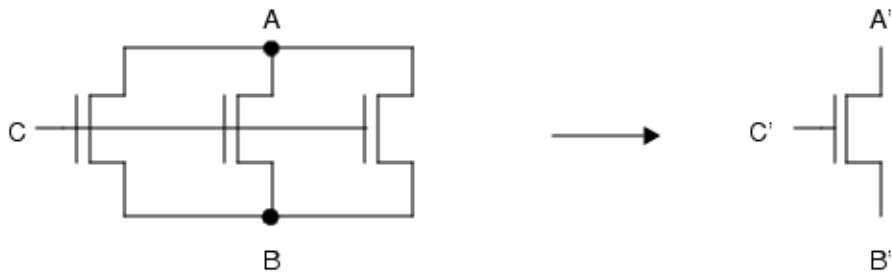
The [LVS Reduce Parallel MOS](#) specification statement controls parallel MOS transistor reduction.

Built-in 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. They conform to pin naming conventions as shown in “[MOS Transistor Required Pin Names](#).” All transistors in the reduction 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 (that is, in parallel).

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 “[Pin Swapping](#)” describes how to specify logical equivalence.

Figure 12-4 shows an example of parallel MOS transistor reduction.

Figure 12-4. Parallel MOS Transistor Reduction



By default, the effective width and length values of the reduced transistor are computed as follows:

$$W = \sqrt{P * Q}$$

$$L = \sqrt{P / Q}$$

where sqrt is the square root function and

$$P = W_1 * L_1 + W_2 * L_2 + \dots + W_n * L_n$$

$$Q = W_1 / L_1 + W_2 / L_2 + \dots + W_n / L_n$$

where W_i and L_i are the width and length of the i th transistor, respectively.

Parallel MOS device reduction also handles effective area of source (AS), area of drain (AD), perimeter of source (PS), and perimeter of drain (PD) using default effective property computations. 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:

$$AS = AS_1 + AS_2 + \dots + AS_n$$

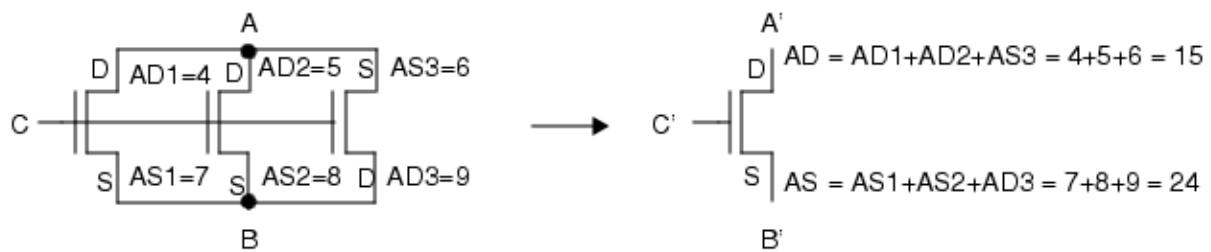
$$AD = AD_1 + AD_2 + \dots + AD_n$$

$$PS = PS_1 + PS_2 + \dots + PS_n$$

$$PD = PD_1 + PD_2 + \dots + PD_n$$

The values of AS and AD, and PS and PD, are interchangeable if some of the transistors have source and drain pins swapped. Calibre nmLVS 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 12-5](#) shows an example.

Figure 12-5. Effective AS/AD Computation With Pin Swapping



Effective property values are computed in both layout and source. If you use default effective property computation (that is, you do not provide your own effective property computations in an [LVS Reduce Parallel MOS](#) statement), then to compute effective 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” in your netlists, respectively, except when otherwise specified in Trace Property statements as discussed under “[Built-In Property Classification](#)” on page 467.

You can override the default effective property computation and specify other formulas as described in the section “[Device Reduction Programs](#)” on page 424.

For geometric layouts, only W and L are calculated by default for MOS devices; however, you can calculate any device properties by specifying a user-defined property computation in your MOS [Device](#) statements. Your [LVS Reduce Parallel MOS](#) statement should then calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

If you calculate NRS and NRD properties in an effective property computation program, you can control whether these properties are swappable through the [LVS Builtin MOS NRD_NRS](#) specification statement.

[LVS Short Equivalent Nodes](#) PARALLEL reduction is prioritized ahead of all other MOS reductions. This allows you to control further MOS reductions through use of the LVS Reduce Parallel MOS YES or the [LVS Reduce](#) equivalent specification statements.

See “[Built-In W/L Partner Properties](#)” on page 468.

Unequally Reduced Devices

Calibre nmLVS 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 12-5](#).

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](#).

Series MOS Transistor Reduction

Series MOS transistors are not reduced by default during comparison of built-in MOS devices.

The [LVS Reduce Series MOS](#) specification statement controls series MOS reduction.

Built-in 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. They conform to pin naming conventions as shown in “[MOS Transistor Required Pin](#)

Names .” All transistors in the reduction 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. Gate, bulk, and optional pins must be connected to the same nets, respectively (that is, parallel).

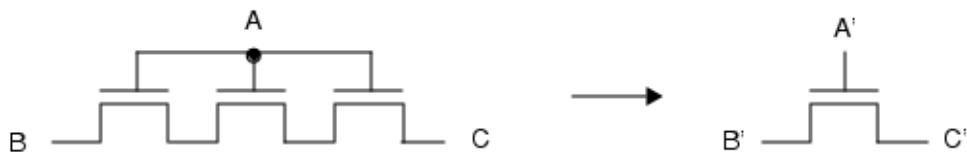
In MN, MP, ME, MD, M, and equivalent devices, source and drain are equivalent and their polarity is immaterial.

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 “[Pin Swapping](#)” on page 462 describes how to specify logical equivalence.

[Figure 12-6](#) shows an example of series MOS transistor reduction.

Figure 12-6. Series MOS Transistor Reduction



By default, the effective width and length values of the reduced transistor are computed as follows:

$$W = \sqrt{P / Q}$$

$$L = \sqrt{P * Q}$$

where $\sqrt{\cdot}$ is the square root function and

$$P = W_1 * L_1 + W_2 * L_2 + \dots + W_n * L_n$$

$$Q = L_1 / W_1 + L_2 / W_2 + \dots + L_n / W_n$$

where W_i, L_i are the width and length of the i th transistor, respectively.

By default, series MOS device reduction does not compute the effective values for area of source (AS), area of drain (AD), perimeter of source (PS), and perimeter of drain (PD) in series MOS transistors.

Effective property values are computed in both layout and source. If you use default effective property computation (that is, you do not provide your own effective property computations in an [LVS Reduce Series MOS](#) statement), then to compute effective width and length values, you

must use the built-in property names “w” and “l” in your netlists, respectively, except when otherwise specified in Trace Property statements as discussed under “[Built-In Property Classification](#)” on page 467.

You can override the default effective property computation and specify other formulas as described in the section “[Device Reduction Programs](#)” on page 424.

For geometric layouts, only W and L are calculated by default for MOS devices; however, you can calculate any device properties by specifying a user-defined property computation in your [MOS Device](#) statements. Your [LVS Reduce Series MOS](#) statement should then calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

See “[Built-In W/L Partner Properties](#)” on page 468.

Semi-Series MOS Transistor Reduction

Semi-series MOS transistor reduction is not performed by default.

The [LVS Reduce Semi Series MOS](#) specification statement controls semi-series MOS transistor reduction.

Built-in 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. They conform to pin name conventions as shown in the table “[MOS Transistor Required Pin Names](#)” on page 384. All transistors in the reduction 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 12-7](#). Gate, bulk, and optional pins must be connected to the same nets, respectively (that is, parallel).

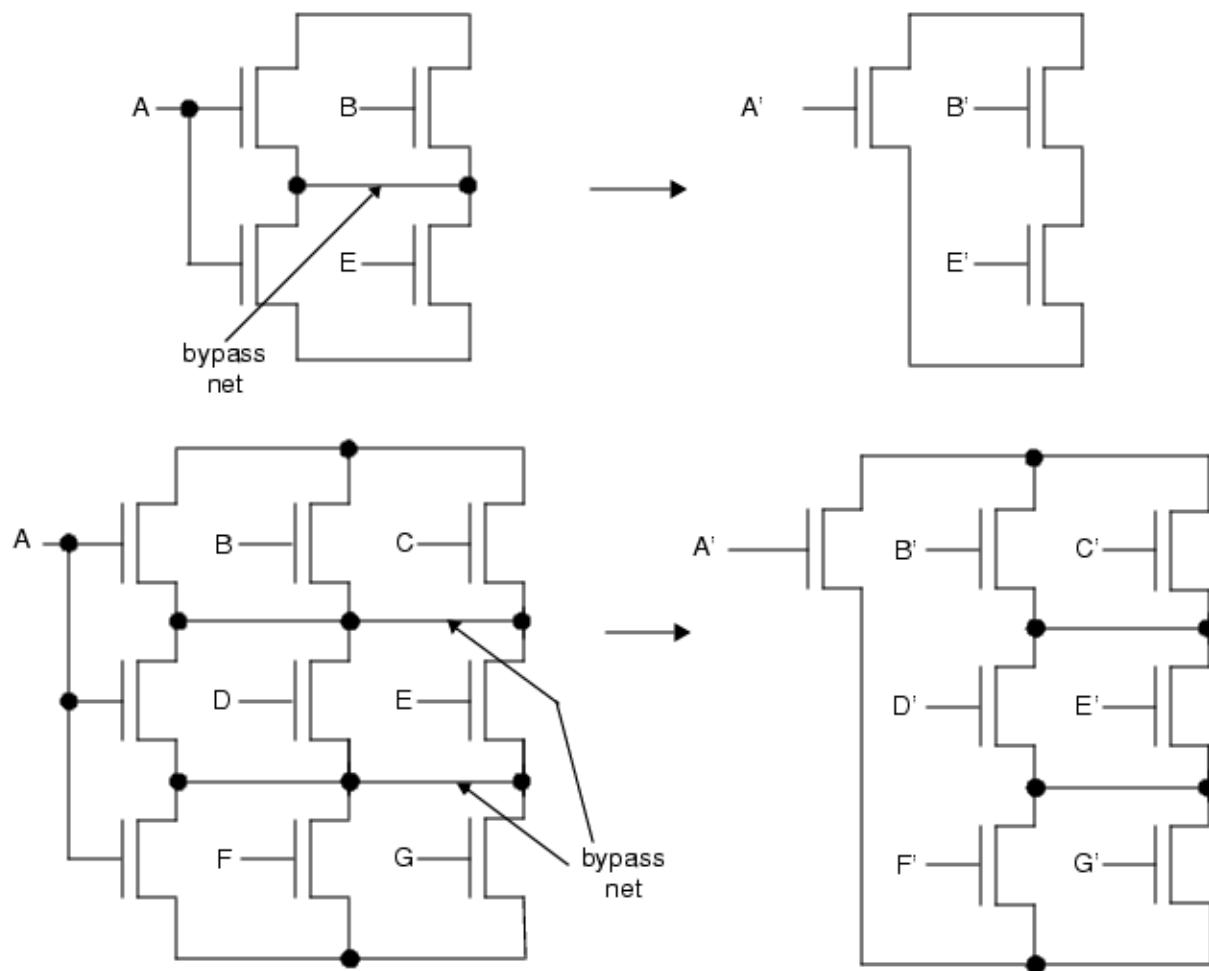
In MN, MP, ME, MD, M, and equivalent devices, source and drain are equivalent and their polarity is immaterial.

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 “[Pin Swapping](#)” on page 462 describes how to specify logical equivalence.

[Figure 12-7](#) illustrates that the bypass net must not be connected to any devices outside of the respective “row” of the series-parallel structure.

Figure 12-7. Reduce Semi-Series MOS



Semi-series MOS reduction is independent of regular series MOS reduction; either or both may be performed.

By default, the width and length values of the reduced transistor are computed as follows:

$$W = \sqrt{P / Q}$$

$$L = \sqrt{P * Q}$$

Where $\sqrt{\cdot}$ is the square root function and

$$P = W_1 * L_1 + W_2 * L_2 + \dots + W_n * L_n$$

$$Q = L_1 / W_1 + L_2 / W_2 + \dots + L_n / W_n$$

where W_i, L_i are the width and length of the i th transistor, respectively.

By default, semi-series MOS device reduction does not compute the effective values for area of source (AS), area of drain (AD), perimeter of source (PS), and perimeter of drain (PD) in semi-series MOS transistors.

Effective property values are computed in both layout and source. If you use default effective property computation (that is, you do not provide an effective property computation in your [LVS Reduce Semi Series MOS](#) statement), then to compute effective width and length values, you must use the built-in property names “w” and “l” in your netlists, respectively, except when otherwise specified in Trace Property statements as discussed under [“Built-In Property Classification”](#) on page 467.

You can override the default effective property computation and specify other formulas as described in [“Device Reduction Programs”](#) on page 424.

For geometric layouts, only W and L are calculated by default for MOS devices; however, you can calculate any device properties by specifying a user-defined property computation in your [MOS Device](#) statements. Your [LVS Reduce Semi Series MOS](#) statement should then calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

Split Gate Reduction

Split-gate MOS structures are reduced by default during comparison of built-in MOS devices.

Split-gate reduction is controlled by the [LVS Reduce Split Gates](#), [LVS Short Equivalent Nodes](#), and [LVS Reduce ... SPLIT GATES](#) specification statements.

In many cases, LVS Short Equivalent Nodes SPLIT (which requires LVS Reduce Split Gates NO) is the preferred configuration in the rule file. This is because it allows device-model-specific reduction, while split-gate reduction does not facilitate this.

Note

 Split-gate reduction implies parallel MOS transistor reduction, even when [Parallel MOS Transistor Reduction](#) is not requested.

If you specify 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). They conform to pin naming conventions as shown in “[MOS Transistor Required Pin Names](#)” on page 384.”

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 12-8](#). 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 as on “[Input Order Considerations](#)” on page 410). When logic gate recognition is disabled, then the order must be the same in all strings. If logic gate tolerance checking is enabled using [LVS Recognize Gates](#) WITHIN TOLERANCE option, and a split gate structure violates the tolerance in any pertinent [LVS Recognize Gates Tolerance](#) statement, then the LVS Reduce Split Gates SAME ORDER option is forced on the entire split gate structure, even when that option is not explicitly specified.

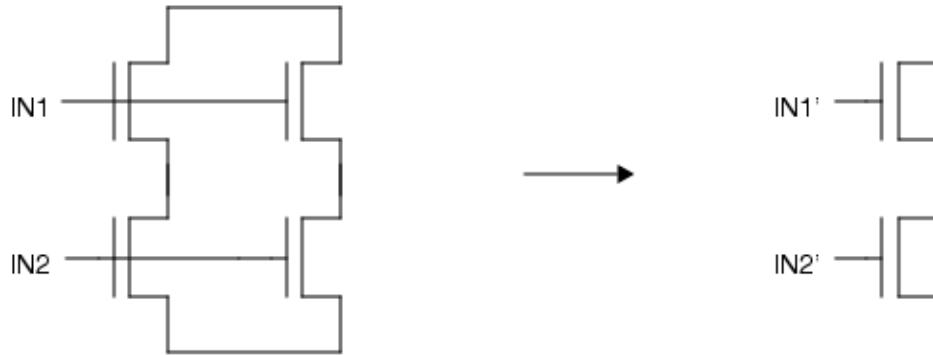
Normally, all the transistors in the split-gate structure must have the same component type, the same number of pins, and the same pin names, and the same pin swap conditions. This can be changed with the LVS Reduce Split Gates MIX TYPES option (see “[Mixed Component Types](#)” on page 410). Subtypes must be the same in each group of transistors that are collapsed together.

For example, in [Figure 12-8](#), 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.

Figure 12-8. Split Gate Reduction



Optional pins may be swapped if they are specified as logically equivalent. See the section “[Pin Swapping](#)” on page 462.

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.

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 in one design 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 in both designs, then a representative net is chosen from each group of nets that were collapsed together in one design and respective nets in the other design are matched to that representative.

By default, for each group of transistors that is reduced to a single transistor, the width and length values of the reduced transistor are computed as follows:

$$L = \sqrt{P / Q}$$

$$W = \sqrt{P * Q}$$

where $\sqrt{\cdot}$ is the square root function and the following relations hold:

$$P = W_1 * L_1 + W_2 * L_2 + \dots + W_n * L_n$$

$$Q = W_1 / L_1 + W_2 / L_2 + \dots + W_n / L_n$$

W_i and 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 (that is, you do not provide an effective property computation in an LVS Reduce Split Gates or LVS Reduce ... SPLIT GATES statement), then to compute effective width and length values, you must use the built-in property names “w” and “l” in your netlists, respectively. This is true except when the properties are otherwise specified in Trace Property statements as discussed under “[Built-In Property Classification](#)” on page 467.

You can override the default effective property computation and specify other formulas as described in the section “[Device Reduction Programs](#)” on page 424.

For geometric layouts, only W and L are calculated by default for MOS devices; however, you can calculate any device properties by specifying a user-defined property computation in your MOS Device statements. Your [LVS Reduce Semi Series MOS](#) statement should then calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

Series-Parallel Split Gate Reduction	407
Semi-Split Gate Reduction	409
Input Order Considerations	410
Error Tolerances	410
Mixed Component Types	410
Ambiguous Topologies and Split Gate Reduction	411

Series-Parallel Split Gate Reduction

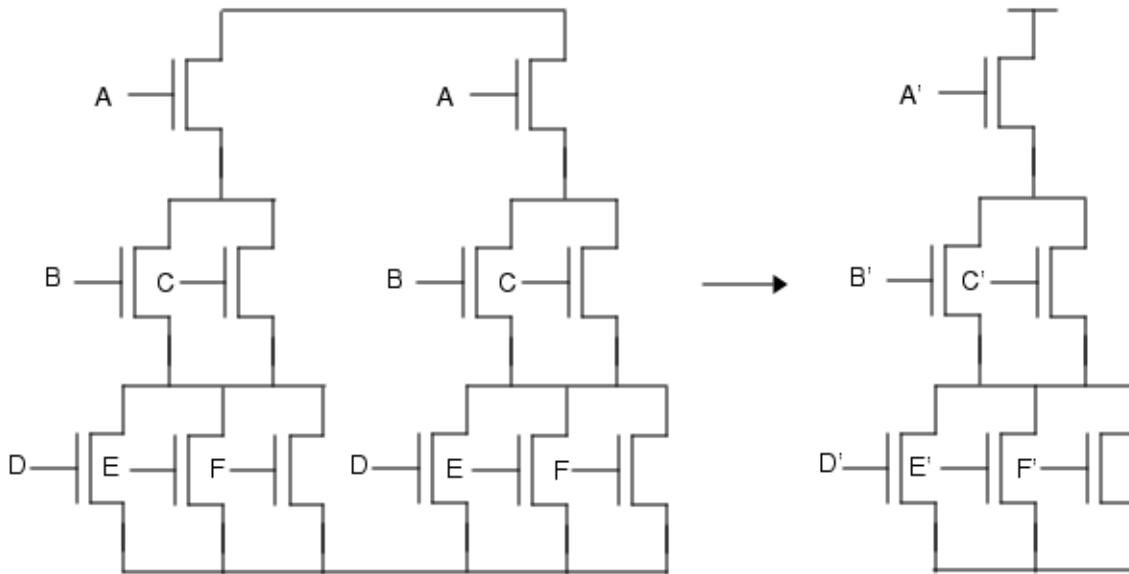
Series-parallel device reduction is a generalization of regular split-gate reduction. It differs from the regular split-gate reduction in that the parallel structures contained in transistor strings are also collapsed.

[Figure 12-9](#) illustrates the series-parallel split-gate reduction. The reduction applies to the series-parallel structures of MOS devices where all series structures have the same number of parallel device structures. In order for the parallel structures to be collapsed, they must have the following characteristics:

- Gate pins are shared.
- Have the same number of transistors with the following attributes being identical:
 - component types and subtypes
 - numbers of pins
 - pin names
 - pin swappability

If there are more than three pins, then all of the optional pins must be connected in parallel across the row. In [Figure 12-9](#), letters A-F indicate nets connected to the gate pins; all gate pins marked with the same letter are assumed to be connected to the same net.

Figure 12-9. Series-Parallel Split Gate Reduction



For comparison, the regular split-gate reduction operates on similar structures but requires that all gates in each parallel group to be shorted together, that is pins B and C must be the same net (same for D, E, and F).

If exactly two parallel device groups are reduced, as drawn in the figure, then some special conditions apply. In order for the reduction to proceed, then at least one of the following must be true for the uppermost net in the diagram:

- The net must be connected to at least one more device.
- The net must have a user-given name and be an initial correspondence point.
- The net must connect to a pin of the cell.

Otherwise, the selection of the uppermost net itself is ambiguous and the reduction is not done.

The series-parallel split gate reduction is enabled with the SP ALSO option of the [LVS Reduce Split Gates](#) specification statement. The other options of this statement apply to the series-parallel split gate reduction and have the same effect as they do for regular split gate reduction. The same is true for property calculations, whether built-in or user-defined, and the LVS Split Gate Ratio specification statement.

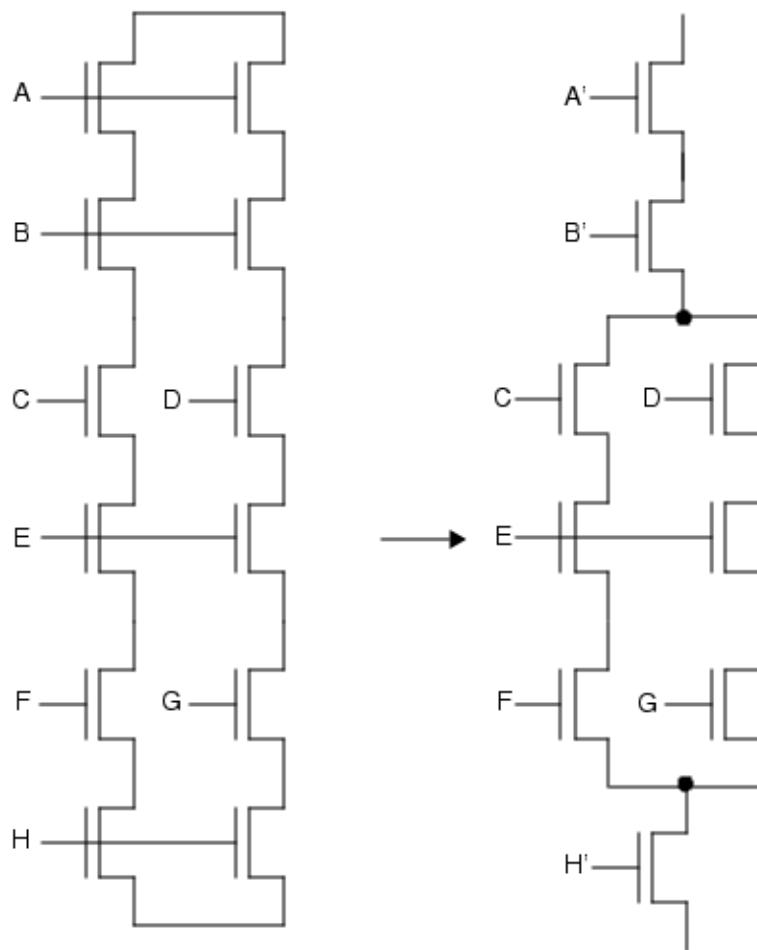
Semi-Split Gate Reduction

Semi-split gate reduction is enabled by the SEMI ALSO option of the LVS Reduce Split Gates statement.

Calibre nmLVS 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 12-10 shows that transistors in row E are not reduced. Unlike fully split gates, the order of transistors in semi-split gates must be the same for all strings of MOS transistors, regardless of whether logic gate recognition is enabled. All strings must consist of the same number of transistors.

Figure 12-10. Reduce Split Gates Example

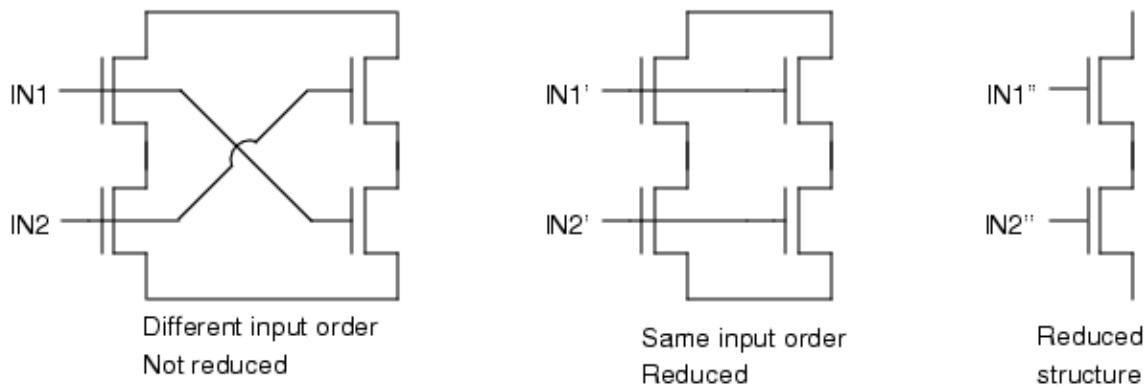


Input Order Considerations

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 12-11 shows the effect of the SAME ORDER option.

Figure 12-11. 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 an LVS Split Gate Ratio discrepancy.

Mixed 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 392) 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.

Ambiguous Topologies and Split Gate Reduction

Certain arrangements of devices are ambiguous for split-gate reduction.

When the order of the layout input data differs from that of the source input data, as shown by the ordering differences in [Figure 12-12](#) and [Figure 12-13](#), the topologies are ambiguous with respect to each other. Hence, the resulting reduced property values can differ because instances with different pre-reduction property values may be associated to one another in different orders.

In the split gate structures shown in the figures, the gate pins of all transistors are shorted to a common net IN. Also, the top and bottom nets (the outputs of a split gate structure) are shorted to a common net OUT.

Assume that in the rule file, the following specification statements exist:

```
LVS REDUCE SPLIT GATES YES
TRACE PROPERTY MP L L
TRACE PROPERTY MP W W
```

With these conditions, it is not possible to reduce the following structures without ambiguity.

Figure 12-12. Layout Schematic

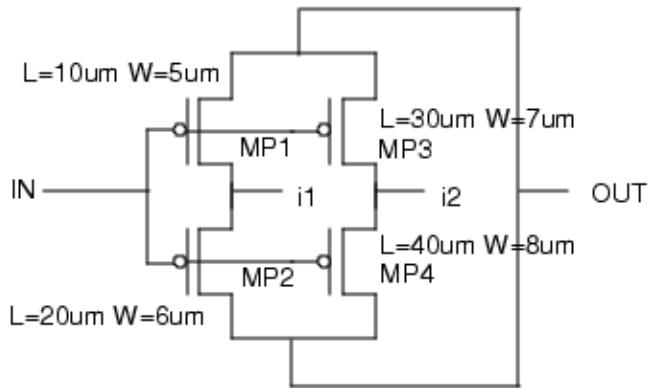
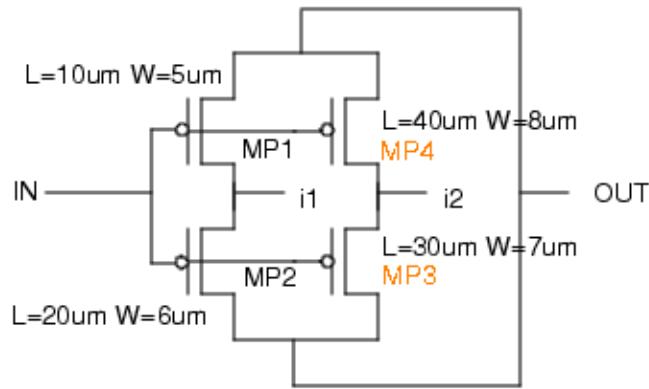
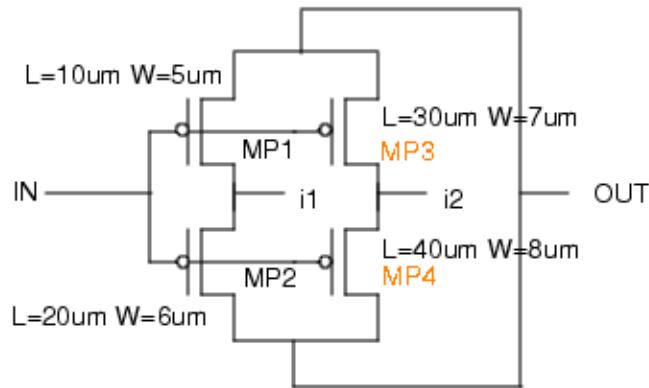


Figure 12-13. Source Schematic



If the source input data from [Figure 12-14](#) is compared to layout input data from [Figure 12-12](#) after reduction, the comparison has the same reduced property values in both layout and source.

Figure 12-14. Alternate Source Schematic



The ambiguity of the preceding topology shown may cause LVS to report an incorrect result when comparing [Figure 12-12](#) to [Figure 12-13](#), while LVS reports a correct result if [Figure 12-12](#) is compared to [Figure 12-14](#).

LVS Reduce Split Gates reduces all split gates, but sets the resulting reduced property values to UNKNOWN. Calibre nmLVS then reports a consistent result each time, regardless of input ordering in the presence of ambiguity.

[Figure 12-15](#) and [Figure 12-16](#) represent an exception that can be handled by split-gate reduction and still produce consistent reduced property values, even though both structures are topologically ambiguous. Regular property-value reduction formulas are applied instead of assigning UNKNOWN property values to the reduced instances.

The right-hand column in [Figure 12-16](#) is inverted vertically with respect to the same column in [Figure 12-15](#). Yet this has no impact on the property values that are used when associating the right-hand column devices with MP1 and MP2 in the left-hand column.

Figure 12-15. Layout with Property-Value Symmetry

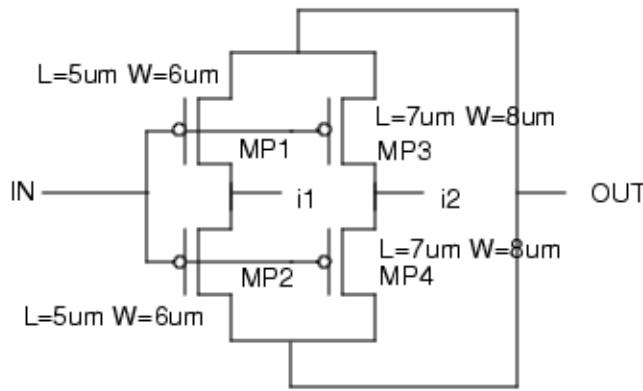
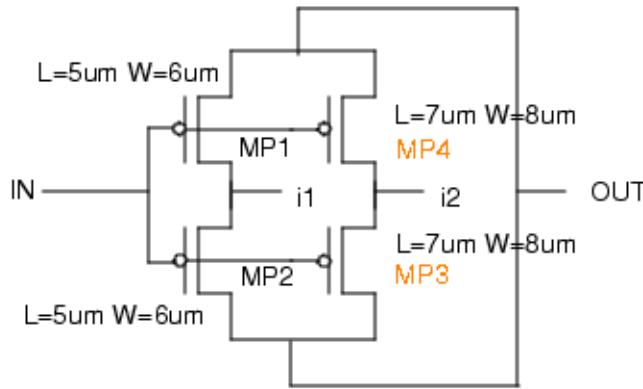


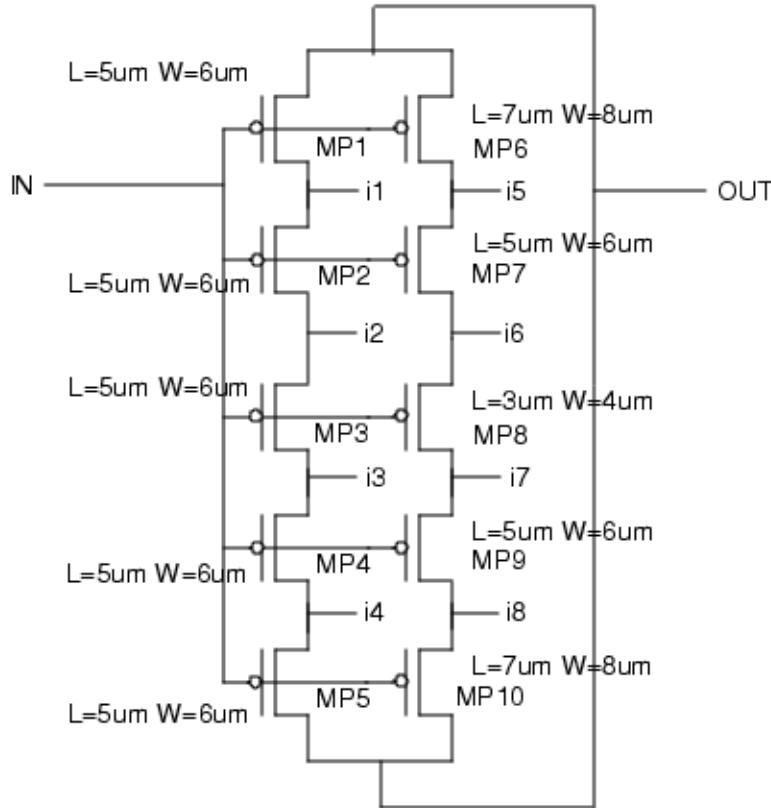
Figure 12-16. Source with Property-Value Symmetry



If you reverse its ordering vertically, the property values read out in the same order. Both [Figure 12-15](#) and [Figure 12-16](#) reduce the same way and produce the same post-reduction property values regardless of the order of input data.

Figure 12-17 shows a larger example of five devices with the same property value symmetry. It is reduced with real property values written to the reduced devices.

Figure 12-17. Input Data with Property-Value Symmetry



The left-hand column can be ignored because it has all same properties and has the same real values after device reduction. Consider the right-hand column. Note that whether you scan top to bottom (MP6, MP7, MP8, MP9, MP10) or bottom to top (MP10, MP9, MP8, MP7, MP6) the list of property values is equivalent.

Table 12-11. List of Property Values Based on the Scan Direction

Scan: Top to Bottom	Property List 1	Scan: Bottom to Top	Property List 2
MP6	L=7um W=8um	MP10	L=7um W=8um
MP7	L=5um W=6um	MP9	L=5um W=6um
MP8	L=3um W=4um	MP8	L=3um W=4um
MP9	L=5um W=6um	MP7	L=5um W=6um
MP10	L=7um W=8um	MP6	L=7um W=8um

Note that property list 1 and property list 2 are the same. As long as each device has the same adjacent neighbors, then the property values produce consistent reduction results. For instance,

MP7 should have MP6 and MP8 as neighbors. Cases like these are detected even if there are an even or odd number of devices in the column.

Each column in a split gate structure is checked for this condition. If all columns in the split gate structure comply, it is reduced normally. Valid property values (not the UNKNOWN value) are written to the resulting reduced devices.

Property values for reduced devices are calculated with the normal split gate reduction formulas. See “[Split Gate Reduction](#)” on page 405 for more information on the property value calculation formulas used when split gate reduction occurs.

Parallel Capacitor Reduction

Parallel capacitor reduction is performed by default during comparison of built-in capacitors.

The [LVS Reduce Parallel Capacitors](#) specification statement controls parallel capacitor reduction. Calibre nmLVS 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.

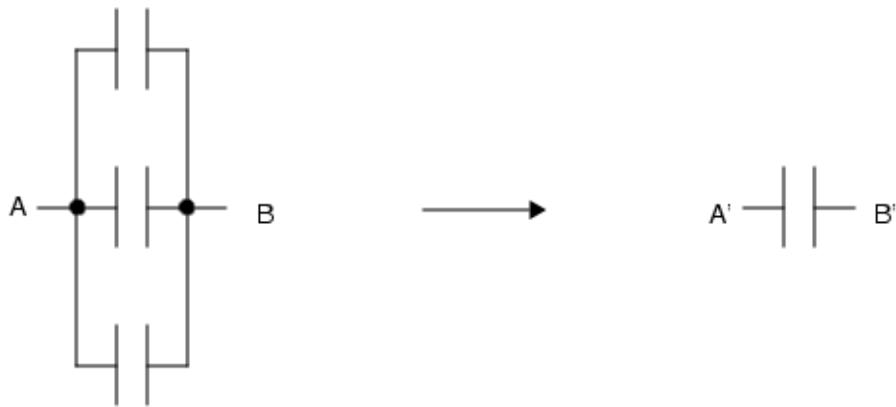
To be reduced, capacitors must have at least two pins with the standard pin names shown in “[Capacitor Required Pin Names](#).[“](#) All capacitors in the reduction group must have the same optional component subtype, number of pins, and pin names. All positive, negative, and optional pins (if any) must be connected to the same nets.

The positive, negative, and optional pins can be swapped if they are specified as logically equivalent. The section “[Pin Swapping](#)” on page 462 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.

[Figure 12-18](#) shows an example of parallel capacitor reduction.

Figure 12-18. Parallel Capacitor Reduction



By default, the effective capacitance, area, and perimeter values of the resulting device are computed as follows:

$$C = C_1 + C_2 + \dots + C_n$$

$$A = A_1 + A_2 + \dots + A_n$$

$$P = P_1 + P_2 + \dots + P_n$$

where C_i , A_i , and P_i are the capacitance, area, and perimeter of the i th capacitor, respectively.

Effective capacitance, area, and perimeter values are computed in both layout and source. If you use default effective property computation (that is, you do not specify a user-defined effective property computation in an [LVS Reduce Parallel Capacitors](#) statement), then to compute the effective capacitance, area, and perimeter values, you must use the built-in property names “c”, “a”, and “p” in your netlists, respectively, except when otherwise specified in Trace Property statements as discussed under “[Built-In Property Classification](#)” on page 467.

You can override the default effective property computation and specify other formulas as described in the section “[Device Reduction Programs](#)” on page 424.

For geometric layouts, only property C is calculated by default for type C devices; however, you can calculate any device properties by specifying a user-defined property computation in your capacitor [Device](#) statements. Your [LVS Reduce Parallel Capacitors](#) statement should then calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

Series Capacitor Reduction

Series capacitor reduction is performed by default during comparison of built-in capacitors.

The [LVS Reduce Series Capacitors](#) specification statement controls series capacitor reduction. Calibre nmLVS 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.

To be reduced, capacitors must have at least two pins with the standard pin names shown in “[Capacitor Required Pin Names](#).“ All capacitors in the reduction 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. All optional pins must be connected to the same nets, respectively (that is, parallel). The optional pins can be swapped if they are specified as logically equivalent. The section “[Pin Swapping](#)” on page 462 describes how to specify logical equivalence.

Figure 12-19 shows an example of series capacitor reduction.

Figure 12-19. Series Capacitor Reduction



By default, the effective capacitance of the resulting device is computed as follows:

$$C = 1 / (1/C_1 + 1/C_2 + \dots + 1/C_n)$$

where C_i is the capacitance of the i th capacitor, respectively.

Effective capacitance values are computed in both layout and source. If you use default effective property computation (that is, you do not specify a user-defined effective property computation in an [LVS Reduce Series Capacitors](#) statement), then to compute the effective capacitance value, you must use the built-in property name “*c*” in your netlists. This is true except when the property is otherwise specified in Trace Property statements as discussed under “[Built-In Property Classification](#)” on page 467.

You can override the default effective property computation and specify other formulas as described in the section “[Device Reduction Programs](#)” on page 424.

For geometric layouts, only property *C* is calculated by default for type *C* devices; however, you can calculate any device properties by specifying a user-defined property computation in your capacitor [Device](#) statements. Your [LVS Reduce Series Capacitors](#) statement should then calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

Parallel Resistor Reduction

Parallel resistor reduction is performed by default during comparison of built-in resistors.

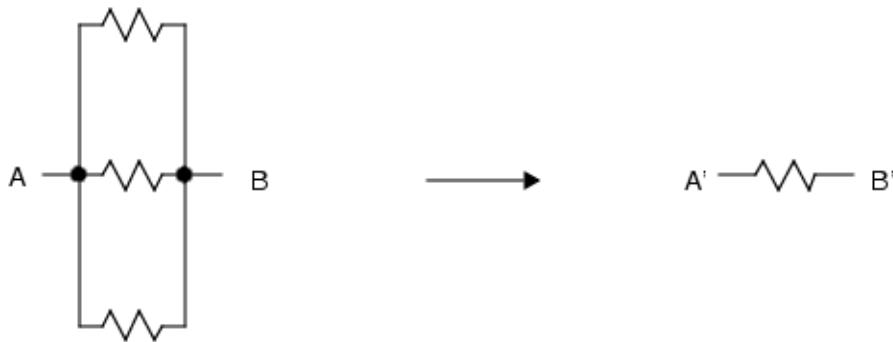
The [LVS Reduce Parallel Resistors](#) specification statement controls parallel resistor reduction. Calibre nmLVS can reduce a group of parallel resistors (device type *R* and any equivalent types indicated with the [LVS Device Type](#) specification statement) to a single resistor.

To be reduced, resistors must have at least two pins with the standard pin names as specified in “[Resistor Required Pin Names](#).[“](#) All resistors in the reduction 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, respectively.

The positive, negative, and optional pins can be swapped if they are specified as logically equivalent. 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. The section “[Pin Swapping](#)” on page 462 describes how to specify logical equivalence.

Figure 12-20 shows an example of parallel resistor reduction.

Figure 12-20. Parallel Resistor Reduction



By default, the resistance value of the reduced transistor is computed as follows:

$$R = 1 / (1/R_1 + 1/R_2 + \dots + 1/R_n)$$

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:

$$W = \sqrt{P * Q}$$

$$L = \sqrt{P / Q}$$

where $\sqrt{\cdot}$ is the square root function and

$$P = W_1 * L_1 + W_2 * L_2 + \dots + W_n * L_n$$

$$Q = W_1 / L_1 + W_2 / L_2 + \dots + W_n / L_n$$

where W_i and L_i are the width and length of the i th resistor, respectively.

Effective resistance, width, and length values are computed in both layout and source. If you use default effective property computation (that is, you do not specify a user-defined effective property computation in an [LVS Reduce Parallel Resistors](#) statement), then to compute the resistance, width, and length values, you must use the built-in property names “ r ”, “ w ”, and “ l ” in your netlists, respectively. This is true except when the properties are otherwise specified in Trace Property statements as discussed under [“Built-In Property Classification”](#) on page 467.

You can override the default effective property computation and specify other formulas as described in the section [“Device Reduction Programs”](#) on page 424.

For geometric layouts, only property R is calculated by default for type R devices; however, you can calculate any device properties by specifying a user-defined property computation in your resistor [Device](#) statements. Your [LVS Reduce Parallel Resistors](#) statement should then

calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

Series Resistor Reduction

Series resistor reduction is performed by default during comparison of built-in resistors.

The [LVS Reduce Series Resistors](#) specification statement controls series resistor reduction. Calibre nmLVS 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.

To be reduced, resistors must have at least two pins with the standard pin names as specified in “[Resistor Required Pin Names](#).” All resistors in the reduction 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. 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. The section “[Pin Swapping](#)” on page 462 describes how to specify logical equivalence.

[Figure 12-21](#) shows an example of series resistor reduction.

Figure 12-21. Series Resistor Reduction



By default, the resistance value of the reduced transistor is computed as follows:

$$R = R_1 + R_2 + \dots + R_n$$

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:

$$W = \sqrt{P / Q}$$

$$L = \sqrt{P * Q}$$

where $\sqrt{}$ is the square root function and the following

$$P = W_1 * L_1 + W_2 * L_2 + \dots + W_n * L_n$$

$$Q = L_1 / W_1 + L_2 / W_2 + \dots + L_n / W_n$$

where W_i, L_i are the width and length of the i th resistor respectively.

Effective resistance, width, and length values are computed in both layout and source. If you use default effective property computation (that is, you do not specify a user-defined effective property computation in an [LVS Reduce Series Resistors](#) statement), then to compute the resistance, width, and length values, you must use the built-in property names “r”, “w”, and “l” in your netlists, respectively, except when otherwise specified in Trace Property statements as discussed under “[Built-In Property Classification](#)” on page 467.

You can override the default effective property computation and specify other formulas as described in the section “[Device Reduction Programs](#)” on page 424.

For geometric layouts, only property R is calculated by default for type R devices; however, you can calculate any device properties by specifying a user-defined property computation in your resistor [Device](#) statements. Your [LVS Reduce Series Resistors](#) statement should then calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

Parallel Diode Reduction

Parallel diode reduction is performed by default during comparison of built-in diodes.

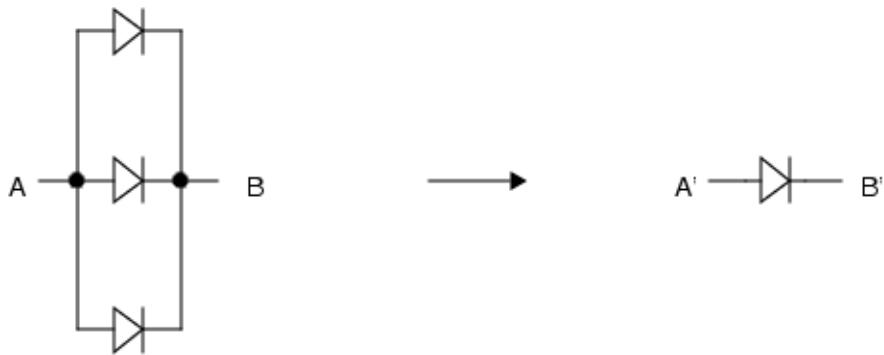
The [LVS Reduce Parallel Diodes](#) specification statement controls parallel diode reduction. Calibre nmLVS can reduce a group of parallel diodes (device type D and any equivalent types indicated with the [LVS Device Type](#) specification statement) to a single diode.

To be reduced, diodes must have at least two pins with the standard pin names as specified in “[Diode Required Pin Names](#).[“](#) 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 “[Pin Swapping](#)” on page 462 describes how to specify logical equivalence.

[Figure 12-22](#) shows an example of parallel diode reduction.

Figure 12-22. Parallel Diode Reduction



By default, the area and perimeter values of the reduced diode are computed as follows:

$$A = A_1 + A_2 + \dots + A_n$$

$$P = P_1 + P_2 + \dots + P_n$$

where A_i and P_i are the area and perimeter of the i th diode respectively.

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” in your netlists, respectively, except when otherwise specified in Trace Property statements as discussed under “[Built-In Property Classification](#)” on page 467.

The default effective property computation can be overridden, and other formulas can be specified as described in the section “[Device Reduction Programs](#)” on page 424.

For geometric layouts, properties A and P are calculated by default for type D devices; however, you can calculate any device properties by specifying a user-defined property computation in your diode [Device](#) statements. Your [LVS Reduce Parallel Diodes](#) statement should then calculate the corresponding effective properties for reduced devices where needed.

Parallel Bipolar Transistor Reduction

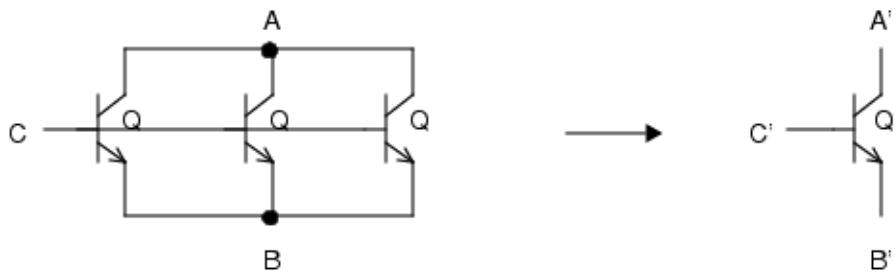
Parallel bipolar transistor reduction is performed by default during comparison of built-in transistors.

Calibre nmLVS 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. The [LVS Reduce Parallel Bipolar](#) specification statement controls parallel bipolar transistor reduction.

To be reduced, bipolar transistors must have at least three pins with the standard pin names as specified in “[Bipolar Transistor Required Pin Names](#).[“](#) All transistors in the reduction 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 “[Pin Swapping](#)” on page 462 describes how to specify logical equivalence.

[Figure 12-23](#) shows an example of parallel bipolar transistor reduction.

Figure 12-23. Parallel Bipolar Transistor Reduction



By default, the effective area of the reduced transistor is computed as follows:

$$A = A_1 + A_2 + \dots + A_n$$

where A_i is the area of the i th transistor.

By default, the effective width and length values of the reduced transistor are computed as follows:

$$W = \sqrt{P * Q}$$

$$L = \sqrt{P / Q}$$

where $\sqrt{\cdot}$ is the square root function and

$$P = W_1 * L_1 + W_2 * L_2 + \dots + W_n * L_n$$

$$Q = W_1 / L_1 + W_2 / L_2 + \dots + W_n / L_n$$

where W_i, L_i are the width and length of the i th transistor, respectively.

Effective property values are computed in both layout and source. If you use default effective property computation (that is, you do not specify a user-defined effective property computation in an [LVS Reduce Parallel Bipolar](#) statement), then you must use the built-in property names “a”, “w”, and “l” in your netlists, respectively. This is true except when the properties are otherwise specified in Trace Property statements as discussed under “[Built-In Property Classification](#)” on page 467.

The default effective property computation can be overridden, and other formulas can be specified as described in the section “[Device Reduction Programs](#)” on page 424.

For geometric layouts, no properties are calculated by default for type Q devices; however, you can calculate any device properties by specifying a user-defined property computation in your bipolar [Device](#) statements. Your [LVS Reduce Parallel Bipolar](#) statement should then calculate, either by default or in a user-defined effective property computation, the corresponding effective properties for reduced devices where needed.

Missing and Unknown Property Values

Under certain conditions, Calibre nmLVS 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:

- If the original input devices lack property X, then the effective property for the reduced input group is “missing”.
- If all input values are present and have valid values, then the effective value for X is calculated as specified by the built-in or user-specified formulas, whichever apply.

Input values are defined as the set of all property values in the input group that participate in the calculation of the 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.”

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. This can also occur 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.

- If using the default property computations, devices in the input group that have identical string properties are reduced with an effective string value equal to the string property. If any of the input devices lack the string property, then the effective value is “unknown.”

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.

Case Comparison of String Properties

By default, comparison of string properties is case-insensitive.

You can make it case-sensitive by specifying [Layout Case YES](#), [Source Case YES](#), and [LVS Compare Case YES](#) or [VALUES](#).

Device Reduction Programs

Device reduction programs may appear in the LVS Reduce family of specification statements.

For example, here is 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 possibly an effective property computation. The entire program is enclosed by 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:

```
// reduce parallel resistors with 0 tolerance for the width measurement.  
// the effective property W is the minimum W in the reduction group.  
LVS REDUCE PARALLEL RESISTORS YES [  
    tolerance W 0  
    effective W  
    W = min( W )  
]
```

The device reduction programming language is fully discussed under “[Device Reduction Effective Property Computation Language](#)” in the *SVRF Manual*.

Tolerance in Device Reduction

The device reduction statements listed here allow you to limit or prevent reduction of devices that have different property values.

Table 12-12. Device Reduction Statements

LVS Reduce Parallel Bipolar	LVS Reduce Series Capacitors
LVS Reduce Parallel Capacitors	LVS Reduce Series MOS
LVS Reduce Parallel Diodes	LVS Reduce Series Resistors
LVS Reduce Parallel MOS	LVS Reduce Split Gates
LVS Reduce Parallel Resistors	LVS Reduce

Tolerance specification consists of two optional keywords called TOLerance and TOLERance STRing, one for numeric properties and the other for string properties. The syntax is this:

```
[TOLerance numeric_property_name tolerance_number
    [... numeric_property_name tolerance_number] ]
[TOLERance STRing string_property_name
    [... string_property_name] ]
```

For example:

```
// tolerance for L and W parameters is 0
// string properties must match str1, str2, or str3 to be reduced
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 and 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 where the property has different numeric values, and the difference exceeds *tolerance_number*. The *tolerance_number* is a floating-point number indicating the tolerance as a percent. The formula for calculating difference is this:

$$\text{abs}((v1-v2)/v1)*100$$

where abs is the absolute value function and v1 and v2 are the property values being compared. The v1 value is the value of the source property. 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, Calibre nmLVS does the check for each property separately. Calibre nmLVS 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 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 property value is not equal to a tolerance string value.

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 compiler error:

```
LVS REDUCE ... [
    TOLERANCE x 0
    TOLERANCE STRING x    // Error, duplicate 'x' declaration.
]
```

In the process of device reduction, Calibre nmLVS iterates over series and parallel reduction steps. For example, series and parallel capacitor reduction steps are repeated. At each iteration, Calibre nmLVS 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 423. 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:

- o 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.)

instance	property
M2001 (MN) (reduced instance)	w: ?
M2014 (MN) (reduced instance)	w: ?
M2021 (MN) (reduced instance)	w: ?

Reduction Tolerance Examples

The following statement reduces parallel MOS devices only when the length values are equal:

```
LVS REDUCE PARALLEL MOS YES [TOLERANCE L 0]
```

The following statement reduces parallel MOS devices only when both width and length are equal:

```
LVS REDUCE PARALLEL MOS YES [TOLERANCE L 0 W 0]
```

The following statement reduces series resistors only when the resistance values are within 5% tolerance and the length values are equal:

```
LVS REDUCE SERIES RESISTORS YES [TOLERANCE R 5 L 0]
```

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 nmLVS allows you to filter out devices during circuit comparison. This allows devices in certain configurations to be ignored. Specification statements in the LVS Filter family control this behavior. Device filtering occurs after reduction in LVS comparison.

The [LVS Filter Unused Option](#) statement supports filtering of unused devices based upon various pin connection configurations. This statement allows more refined control of filtering than the device-specific filtering statements like [LVS Filter Unused Capacitors](#).

A useful statement for filtering cells as devices is [LVS Box](#). LVS Box cells can be filtered as though they were devices by specifying the [LVS Filter](#) statement with the LVS Box cell names.

The following sections describe how unused MOS and bipolar transistors are filtered.

Filtering Unused MOS Transistors	428
Filtering Unused Bipolar Transistors	429

Filtering Unused MOS Transistors

Calibre nmLVS can internally filter unused MOS transistors from source and layout circuits. Filtering is performed only for properly formed MOS transistors.

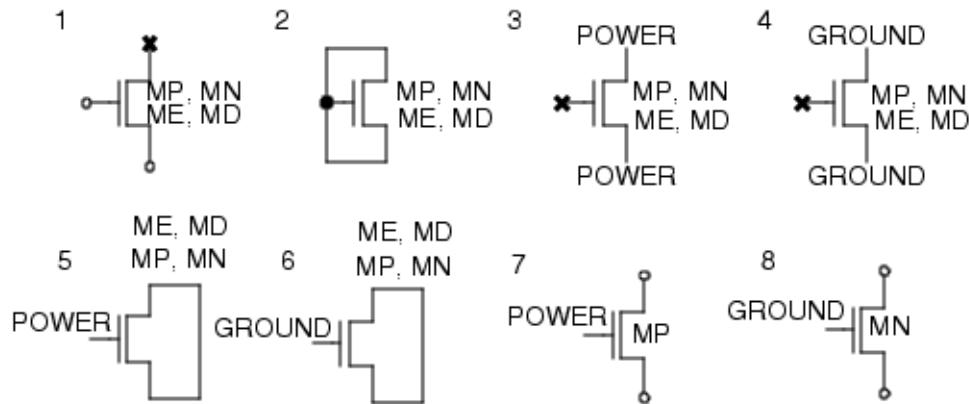
The [LVS Filter Unused MOS](#) specification statement controls unused MOS transistor filtering. The following configurations of transistors are filtered out when YES is specified. Specifically, these are instances with component type MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, or equivalent devices specified in an [LVS Device Type](#) statement that have at least three pins with the standard names.

- MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and equivalent transistors with floating source or drain pin.
- MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and equivalent transistors with source, drain, and gate pins tied together.
- 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.
- 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.
- 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.
- 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.
- MP, LDDP, and equivalent transistors with gate pin connected to a power net.

- MN, LDDN, and equivalent transistors with gate pin connected to a ground net.

[Figure 12-24](#) shows examples of the previous filters, where x denotes a floating pin and o denotes a pin connected to other instances or ports.

Figure 12-24. Unused MOS Transistors

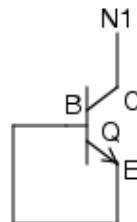


Filtering Unused Bipolar Transistors

Calibre nmLVS can internally filter unused bipolar transistors from the source and layout circuits. Bipolar filtering is useful for verifying gate-array layouts.

The [LVS Filter Unused Bipolar](#) specification statement controls unused bipolar transistor filtering. When YES is specified, filtering is performed only for properly-formed bipolar transistors, specifically, instances with component type Q or equivalent device types specified in an [LVS Device Type](#) statement. The devices must have at least three pins with the standard names as specified in “[Bipolar Transistor Required Pin Names](#).“ Q transistors with base and emitter tied together are filtered out.

Figure 12-25. Unused Bipolar Transistor



Nets

Nets are classified and handled in various contexts in LVS and are reported accordingly.

Usage of Power and Ground Nets	430
Isolated Nets	430
Passthrough Nets	433
Feedthrough Nets	437

Usage of Power and Ground Nets

Supply net names are user-specified in the rule file.

Calibre nmLVS 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 following are true:

- The net name is listed in the [LVS Power Name](#) (or [LVS Ground Name](#)) specification statement.
- The net is connected to a port whose name is listed in the LVS Power Name or LVS Ground Name specification statement and there is no other net in the same design that has this name. If there are several ports with the same power or ground name that are connected to different nets, only one of them is used.

Isolated Nets

Isolated nets are nets that are not connected to any instances or to ports of cells on which LVS operates. Isolated layout and source nets that do not have user-given names in both compared circuits do not cause discrepancies. Named isolated nets are reported throughout the design by default, and, in the top cell, always participate in LVS comparison.

In the layout, isolated nets are essentially bare wires. Flat LVS (calibre -lvs without either -hier or -flatten) reports isolated physical nets because it retains isolated layout shapes in memory and compares them to the source netlist. In a SPICE netlist (used with -hier or -flatten for both layout and source), there is no standard means to represent an extracted bare wire. Therefore, LVS-H uses a control statement to trace isolated physical nets.

In netlist extraction, an isolated net is represented by a *.CALIBRE ISOLATED NETS statement, such as here:

```
.SUBCKT INVX1 A VSS VDD Y
** N=5 EP=4 IP=5 FDC=2
*.CALIBRE ISOLATED NETS: iso
X0 VSS VDD Y A inv_1x $T=0 0 0 0 $X=-1300 $Y=-400
.ENDS
```

Isolated layout nets are reflected in initial object counts and in counts of deleted isolated nets in the LVS Report:

INITIAL NUMBERS OF OBJECTS

	Layout	Source	Component Type
Ports:	4	4	
Nets:	5	5	
Instances:	1	1	MN (4 pins)
	1	1	MP (4 pins)
Total Inst:	2	2	

NUMBERS OF OBJECTS AFTER TRANSFORMATION

	Layout	Source	Component Type
Ports:	4	4	
Nets:	4	5	*
Instances:	1	1	MN (4 pins)
	1	1	MP (4 pins)
Total Inst:	2	2	

* = Number of objects in layout different from number in source.

- o Statistics: **1 isolated layout net was deleted.**

- o **Isolated Layout Nets:**

(Layout nets which are not connected to any instances or ports).

iso

The names of labeled isolated layout nets can result in additional reported layout names not present in the source design:

- o Layout Names That Are Missing In The Source:

Nets: iso

In cells other than the top-level cell, when user-named extracted isolated nets whose names match names in the source design are deleted, they are reported as follows:

- o Matching Deleted Isolated Layout Nets:

(Deleted isolated layout nets from circuit extraction whose names matched names in the source design).

iso

In the top-level cell, named isolated nets that match nets in the source by name are not deleted and participate in LVS comparison.

In LVS-H, when instances of non-hcells or unbalanced instances are expanded into the cells that instantiate them, the node IDs and any names of isolated nets in the expanded cell are not propagated to the parent cells. However, the count of isolated nets in the expanded cell is added to the count of isolated nets in the parent cells.

This behavior applies to the -flatten option because it uses the LVS-H engine. The historical flat engine (-lvs without -hier or -flatten) is able to report net locations because it retains layout information in memory during comparison, while netlists contain net topology only. However, regardless of LVS run mode, Calibre RVE can show isolated nets in the layout.

Again, layout isolated net reporting is performed by default for named nets. To get the same behavior for unnamed nets, see [LVS Netlist Unnamed Isolated Nets](#).

A net that is not isolated in layout or source can become isolated due to the removal of trivial pins, device filtering, and similar circumstances during netlist transformation. These can become trivial nets (see “[Trivial Ports, Pins, and Nets](#)” on page 438), [Passthrough Nets](#), or nets that have their pins removed.

Here is an example of how this latter condition is reported in LVS:

```
2 layout instances were filtered and their pins removed from adjoining
nets.
```

```
1 layout net had its pins removed and was deleted.
```

The net IDs are not reported in this case.

Reporting of isolated nets, except for named isolated nets at the top level, can be disabled using [LVS Report Option I](#). The [Layout Rename Text](#) statement may be used to erase top-level connectivity or port text, thus removing the name of a top-level net.

Floating Named Nets

From the perspective of Calibre nmLVS, *floating nets* are named nets that are not connected to a device pin, port, or initial correspondence point, so they are a subset of isolated nets and are handled in the same way. (This definition may be different from what you are used to. Many users refer to a floating net as one that is connected to a device pin but nothing else. Such a net is not considered floating by the Calibre definition of the term.) Floating nets can be introduced during netlist transformation in the comparison module in the same ways as isolated nets. They can also be introduced in the layout netlist through the [LVS Preserve Floating Top Nets YES](#) statement during the circuit extraction phase.

Passthrough Nets

A passthrough net is a net that is connected to a port and possibly a pin, but nothing else.

Passthrough layout and source nets are ignored during comparison (like [Isolated Nets](#)) by default. This filtering is done because passthrough nets are often present in the layout, but they are rarely present in a schematic design.

Example 12-1. Passthrough Net Processing

Consider these netlists where the layout has passthrough nets but the source does not:

```
$$-- Layout Netlist

.SUBCKT TOP 1 2 3 4      $ net 4 is passthrough
X0 1 2 CELL
X1 2 3 CELL
.ENDS

.SUBCKT CELL 1 2 3      $ net 3 is passthrough
C0 1 2
.ENDS

$$-- Source Netlist

.SUBCKT TOP J K L
X0 J K CELL
X1 K L CELL
.ENDS

.SUBCKT CELL P1 P2
C0 P1 P2
.ENDS
```

These netlists compare as CORRECT, and the passthrough net in the layout top cell is reported like this:

- o Statistics:
1 passthrough layout net was deleted.
- o Passthrough Layout Nets And Their Ports:
(Layout nets which are connected only to ports).
4 (port: 4),

The passthrough in the lower-level layout cell is not reported in this case.

If the source in the preceding example had passthrough nets and ports that corresponded to the layout, but the nets did not match by name, then the reporting for the lower-level cell would be this:

- o Statistics:
1 layout trivial port was deleted.
1 source trivial port was deleted.

Notice the lower-level passthroughs are classified as trivial ports and are deleted.

For the top-level cell, you would see this:

- o Statistics:
1 passthrough layout net was deleted.
1 passthrough source net was deleted.
- o Passthrough Layout Nets And Their Ports:
(Layout nets which are connected only to ports).
4 (port: 4),

If the source and the layout have trivial ports that match by a user-given name in a lower-level cell, then that would be reported as a passthrough. For example:

```
$$-- Layout Netlist

.SUBCKT TOP 1 2 3
X0 1 2 CELL
X1 2 3 CELL
.ENDS

.SUBCKT CELL 1 2 P3 $ lower-level passthrough port matches source name
C0 1 2
.ENDS

$$-- Source Netlist

.SUBCKT TOP J K L
X0 J K CELL
X1 K L CELL
.ENDS

X1 K L CELL
.SUBCKT CELL P1 P2 P3
C0 P1 P2
.ENDS
```

This example compares as correct with the following report for the lower-level cell:

- o Statistics:
1 passthrough layout net was found.
1 passthrough source net was found.
- o Passthrough Layout Nets And Their Ports:
(Layout nets which are connected only to ports).
P3 (port: P3),
- o Initial Correspondence Points:
Ports: P3

Three LVS Report options are available to control handling of passthrough nets and their ports at the top level.

The SP report option is available to report passthrough source nets and their ports:

LVS REPORT OPTION SP

With this option enabled, you would see reports like this for top-level passthroughs:

- o Passthrough Source Nets And Their Ports:
(Source nets which are connected only to ports).
`<net> (port: <port>),`

Two LVS report options allow you to turn unmatched passthrough net warnings into errors.

This statement causes the overall comparison status to be INCORRECT if there is a passthrough net in the source that was not matched to a net in the layout:

`LVS REPORT OPTION SPE`

This option also causes passthrough source nets to be reported like LVS Report Option SP.

This statement causes the overall comparison status to be INCORRECT if there is a passthrough net in the layout that was not matched to a net in the source and subsequently removed:

`LVS REPORT OPTION LPE`

This option has no effect on the reporting of passthrough nets in the source.

Example 1

Consider the following layout and source netlists.

```
* Layout
.subckt top A B C
R0 A B
.ends

* Source
.subckt top A B D
R0 A B
.ends
```

In this case, the layout has a passthrough net C and the source has a passthrough net D. By default, the comparison status is CORRECT, and the following notification appears in the LVS Report:

- o Passthrough Layout Nets And Their Ports:
(Layout nets which are connected only to ports).
`C (port: C),`

Nothing is reported for the source. If LVS Report Option SP is specified, then the comparison status is CORRECT, and the following notification is also reported:

- o Passthrough Source Nets And Their Ports:
(Source nets which are connected only to ports).
`D (port: D),`

If LVS Report Option SPE is in effect, both warnings are reported as shown previously, but the comparison status is INCORRECT, and the following error is reported in the LVS Report:

```
Error: Source passthrough ports missing in layout (see INFORMATION AND
WARNINGS) .
```

Similarly, if LVS Report Option LPE is specified, then the comparison status is INCORRECT. There is no warning for passthrough source nets (unless LVS Report Option SP or LVS Report Option SPE are specified as well) and the following error is reported:

```
Error: Layout passthrough ports missing in source (see INFORMATION AND
WARNINGS) .
```

Example 2

Consider the following layout and source netlists.

```
* Layout
.subckt top A B C
R0 A B
.ends

* Source
.subckt top A B C
R0 A B
.ends
```

In this case, the layout has a passthrough net C and the source has a passthrough net C. The two nets are matched to each other and not removed. The comparison status is always CORRECT, and the following notification is reported:

- o Passthrough Layout Nets And Their Ports:
(Layout nets which are connected only to ports).
C (port: C),

If LVS Report Option SP or LVS Report Option SPE is specified, the comparison status is CORRECT, and the following notification is also reported:

- o Passthrough Source Nets And Their Ports:
(Source nets which are connected only to ports).
C (port: C),

Feedthrough Nets

Feedthrough nets are nets that connect two pins of a cell together, which connect two nets above these pins together. This is sometimes referred to as a *deep short*. In the context of Calibre nmLVS, this term is used in the circuit extraction context rather than the LVS comparison context.

Trivial Ports, Pins, and Nets

A *trivial net* in circuit comparison is a net that is connected only to a port (that is, going up the hierarchy) or only to a pin (that is, going down the hierarchy).

A *trivial port* in circuit comparison is a port that is connected only to a trivial net. The same is true for a *trivial pin*. The difference between a trivial port and a trivial pin is the pin has a connection outside of its parent cell while the port does not.

Unnamed trivial ports and trivial pins are ignored during comparison. For example, the following netlists compare as CORRECT because the trivial elements in the layout netlist are ignored:

Example 12-2. Trivial Net and Trivial Port Processing

```
$$-- Layout Netlist

.SUBCKT TOP 1 2 3
X0 1 2 CELL
X1 2 3 4 CELL      $ third pin is trivial, 4 is a trivial net
.ENDS

.SUBCKT CELL 1 2 3  $ third port is trivial, 3 is a trivial net
C0 1 2
.ENDS

$$-- Source Netlist

.SUBCKT TOP J K L
X0 J K CELL
X1 K L CELL
.ENDS

.SUBCKT CELL P1 P2
C0 P1 P2
.ENDS
```

If a trivial port or a trivial pin is part of a named net that matches in the layout and source, then the port or pin serves as an initial correspondence point for LVS comparison and is not removed.

Trivial nets and ports at the top level, or that match by name in the layout and source at a lower level, are treated in the manner discussed under “[Passthrough Nets](#)” on page 433.

Trivial ports in lower-level cells that match by name in the layout and source are not ignored by default. If you want to ignore trivial named ports in lower-level cells, then specify [LVS Ignore Trivial Named Ports YES](#).

The hierarchical connectivity extractor removes trivial nets and pins from the layout netlist. See “[Netlisting of Ports and Pins](#)” on page 282. You can cause trivial ports in the layout to be reported during circuit extraction with the [LVS Report Trivial Ports YES](#) statement.

Logic Gate Recognition

Calibre nmLVS recognizes logic gates and semi-gates (pullup and pulldown structures and other series-parallel logic gates) in transistor-level circuits.

Calibre nmLVS 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. The tool does the following:

- Forms regular CMOS gates from transistors with component type MP (pullup) and MN (pulldown), or equivalent device types.
- Forms LDD CMOS gates from transistors with component type LDDP (pullup) and LDDN (pulldown), or equivalent device types.
- Forms regular NMOS gates from transistors with component type MD (pullup) and ME (pulldown), or equivalent device types.
- Forms LDD NMOS gates from transistors with component type LDDE (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 370.

The [LVS Recognize Gates](#) specification statement specifies whether logic gate recognition should be performed. The default is all logic gates are recognized.

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, Calibre nmLVS 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 379. 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. Calibre nmLVS 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.

The CELL LIST option allows you to specify a list of cells for which the LVS Recognize Gates statement applies. This enables cell-specific gate recognition. You specify the cell list by using the [LVS Cell List](#) statement.

Tolerance-Based Logic Gate Recognition	440
Effects of Logic Injection	440
Recognition Processes	441
Regular CMOS Gates	442
Regular NMOS Gates	452
LDD Gates	457
X+ Transistors	461
Device and Pin Swapping Checks in Logic Gates	461

Tolerance-Based Logic Gate Recognition

To enable tolerance-based checking, you must specify the LVS Recognize Gates WITHIN TOLERANCE keyword. You must also use either the ALL or SIMPLE keyword. You can also use the CELL LIST option to control logic gate recognition based on cell names.

The [LVS Recognize Gates Tolerance](#) specification statement allows you to limit logic gate recognition based upon device properties matching within specified tolerances or matching as strings. The statement establishes the conditions for recognizing logic gate structures based on tolerance checking of device properties, but it does not enable tolerance checking.

Effects of Logic Injection

Logic injection is enabled by default in Calibre nmLVS-H.

When injected logic structures are introduced, these structures do not participate in gate recognition. For example, injected inverter structures (indicated by _inv elements in the LVS report) do not generally form INV logic gates when [LVS Inject Logic YES](#) is enabled. See “[Logic Injection and Gate Recognition](#)” on page 509 for additional information.

The following logic gate structures participate directly in logic injection when logic injection is enabled:

INV	NAND	NOR
SDW	SUP	SMN
SMP		

If an injected structure is not formed, then that structure can participate in gate recognition.

Other logic gates can be affected by logic injection if a portion of a gate becomes an injected logic component. Such occurrences are context-dependent. SPMN and AOI logic gates are susceptible to this.

Recognition Processes

Calibre nmLVS matches individual transistors in logic gates based on their gate pin connections (transistor pin name G), and all gate types are matched.

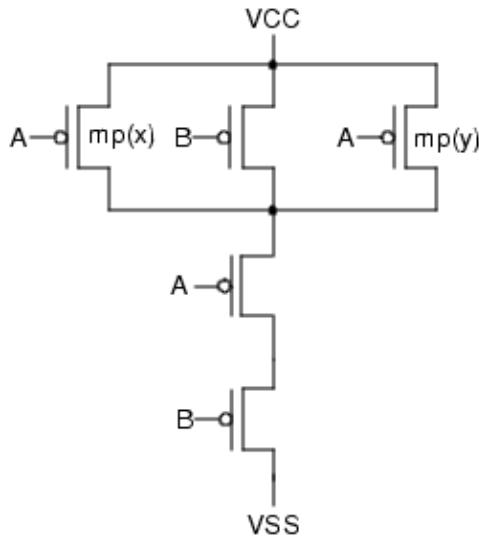
When **LVS Recognize Gates YES** is used, Calibre nmLVS matches internal nets in logic gates based on their relative distance from the output pin or pins of the gate and all gate types, except for complete AOI and OAI gates and higher-level series-parallel structures of type SPxxx(expression) where xxx 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.
- Calibre nmLVS 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. Calibre nmLVS forms only half-gates in these cases to prevent false discrepancies involving subtypes and property values.

In [Figure 12-26](#), Calibre nmLVS must make a choice between the mp(x) and mp(y) transistors. Calibre nmLVS does not form a complete NAND2 gate, but forms an SDW2 structure.

Figure 12-26. LVS Logic Gate Selection



Regular CMOS Gates

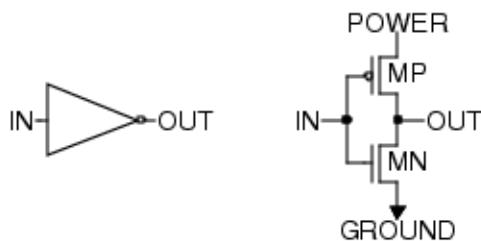
Regular CMOS gates are recognized by default. Specific topologies are shown for each gate type.

For each of the following descriptions, the default gate names appearing in the LVS report are used. If the [LVS Recognize Gates WITH SUBSTRATE](#) keyword is specified, then recognized gate names have suffixes that are either a “v” when the gate does not satisfy the WITH SUBSTRATE criteria, or at least one “b” otherwise.

- **INV**

CMOS inverter.

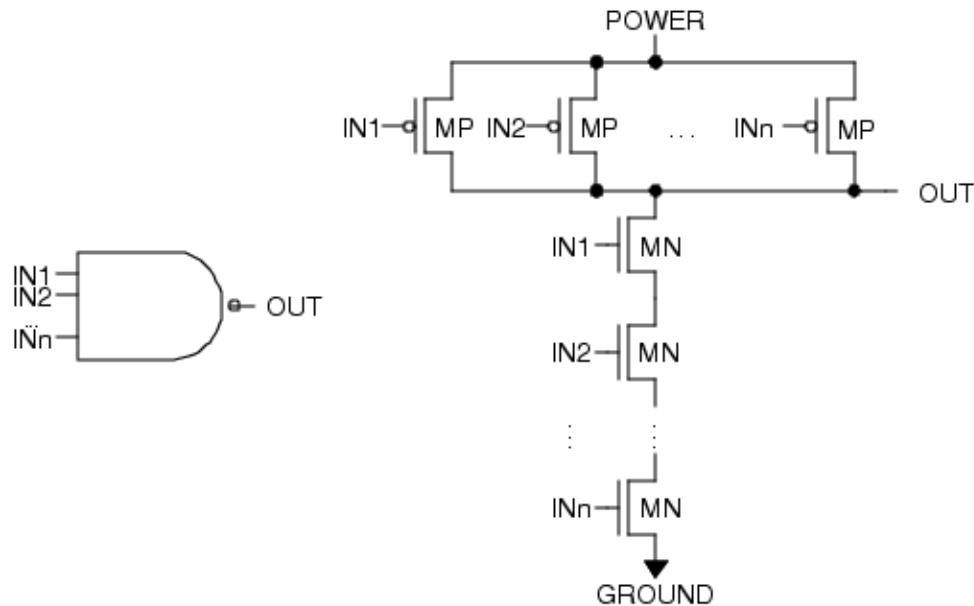
Figure 12-27. INV



- **NAND n**

n -input CMOS NAND. Calibre nmLVS considers all input pins of a NAND n gate logically equivalent. In [Figure 12-28](#), signals IN1, IN2, IN n are all interchangeable.

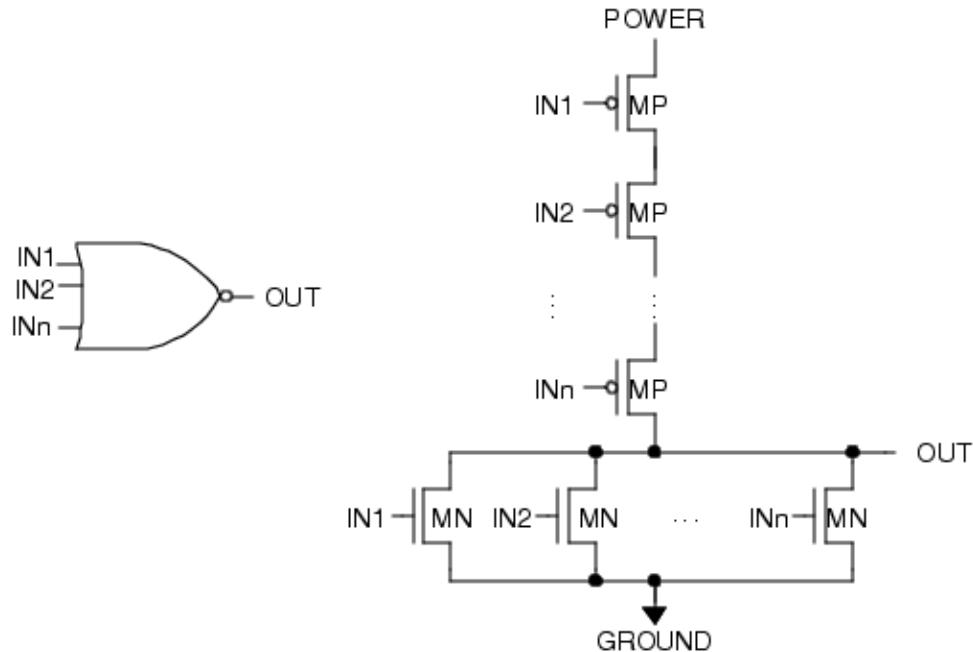
Figure 12-28. NAND n



- **NOR n**

n -input CMOS NOR. Calibre nmLVS considers all input pins of a NOR n gate logically equivalent. In [Figure 12-29](#), signals IN1, IN2, INn are all interchangeable.

Figure 12-29. NOR n

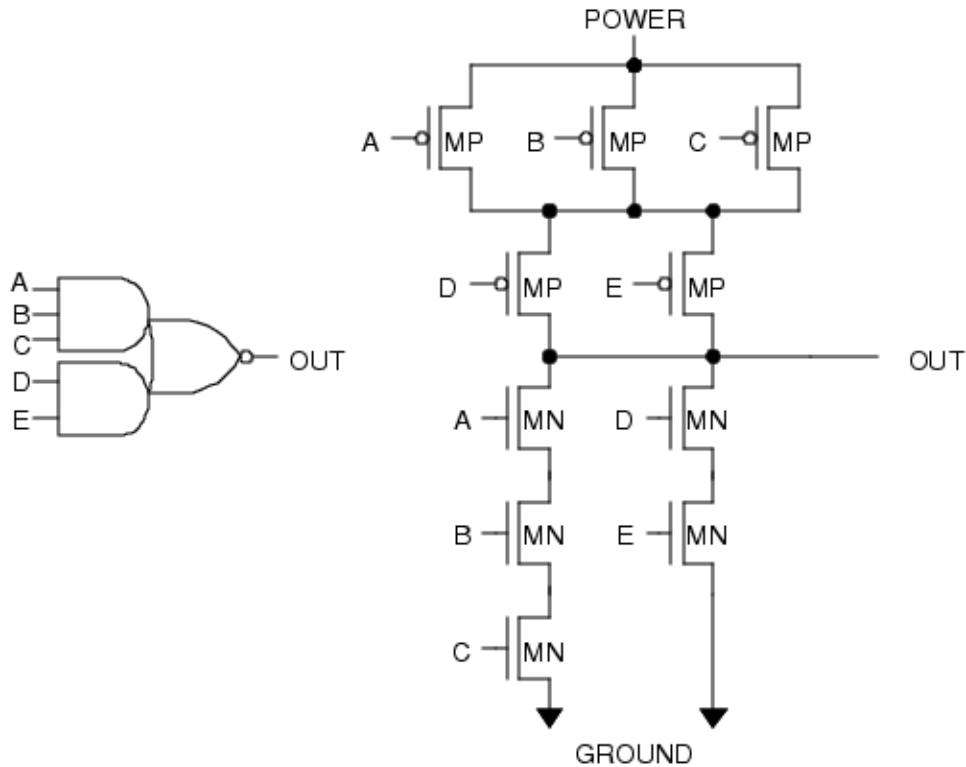


- **AOI_n1_n2_..._nm**

CMOS and-or-invert consisting of m And structures with n_1, n_2, nm inputs each, respectively, leading to an Or-Invert structure.

Calibre nmLVS considers the input pins of each one of the And structures in an AOI gate logically equivalent. In [Figure 12-30](#), signals A, B, and C may be interchanged. Signals D and E are also interchangeable.

Figure 12-30. AOI_3_2



The order of the parallel pullup groups in an AOI gate is interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name `AOI_n1_n2_..._nm` always has $n_1 \geq n_2 \geq \dots \geq nm$. 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.

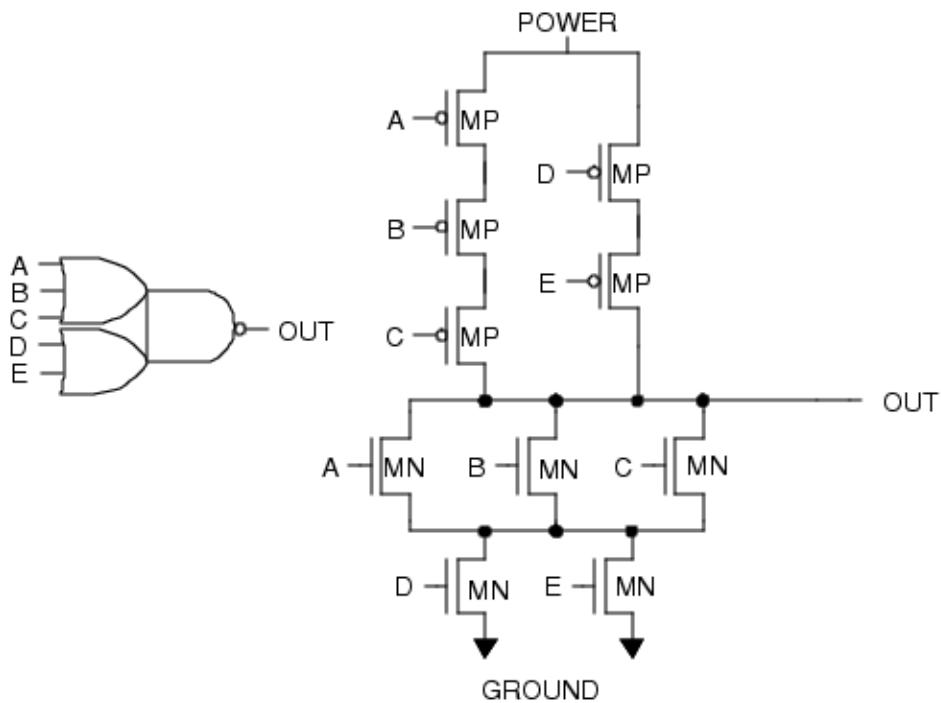
You can prevent the formation of AOI gates by including the [LVS Recognize Gates SIMPLE](#) statement in the rule file. In which case, Calibre nmLVS instead forms separate structures of type SPUP and SDW.

- **OAI_n1_n2_..._nm**

CMOS or-and-invert consisting of m OR structures with n_1, n_2, nm inputs each, respectively, leading to an AND-Invert structure.

Calibre nmLVS considers the input pins of each one of the OR structures in an OAI gate logically equivalent. In [Figure 12-31](#), signals A, B, and C are all interchangeable. Signals D and E are also interchangeable.

Figure 12-31. OAI_3_2



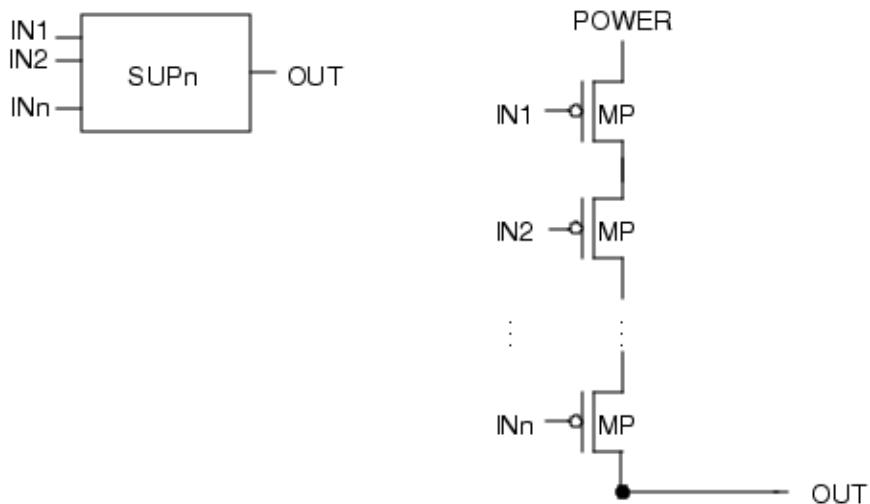
The order of the parallel pulldown groups in an AOI gate is interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name `OAI_n1_n2_ ... _nm` always has $n1 \geq n2 \geq nm$. In the figure, the group of three parallel MP transistors connected to A, 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 [LVS Recognize Gates SIMPLE](#) statement in the rule file. In which case, Calibre nmLVS instead forms separate structures of type SUP and SPDW.

- **SUPn** (serial up)

n -input CMOS serial pullup. Calibre nmLVS considers all input pins of a SUP n gate logically equivalent. In [Figure 12-32](#), signals IN1, IN2, IN n are all interchangeable.

Figure 12-32. SUP n

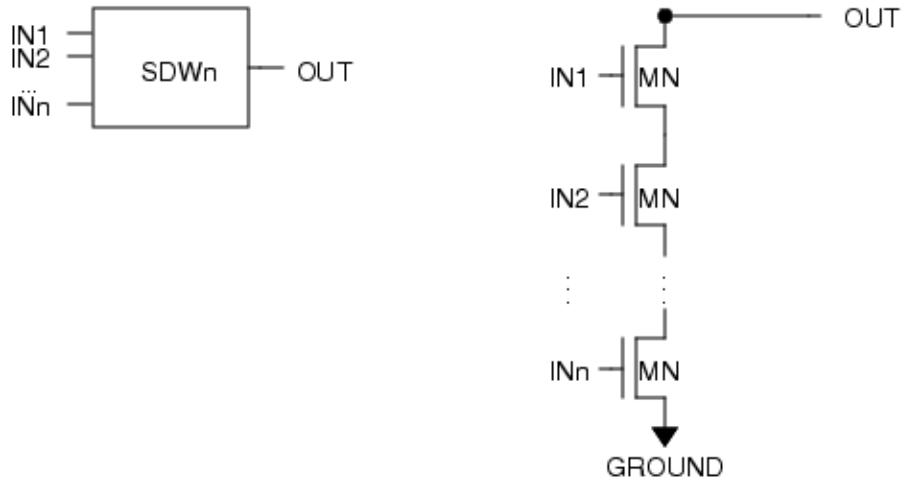


Calibre nmLVS represents pullups as stand-alone gates when they are not contained in complete gates, or when they are contained in OAI gates but complex gate recognition is turned off by including the [LVS Recognize Gates](#) SIMPLE statement in the rule file.

- **SDW n** (serial down)

n -input CMOS serial pulldown. Calibre nmLVS considers all input pins of a SDW n gate logically equivalent. In [Figure 12-33](#), signals IN1, IN2, INn are all interchangeable.

Figure 12-33. SDW n



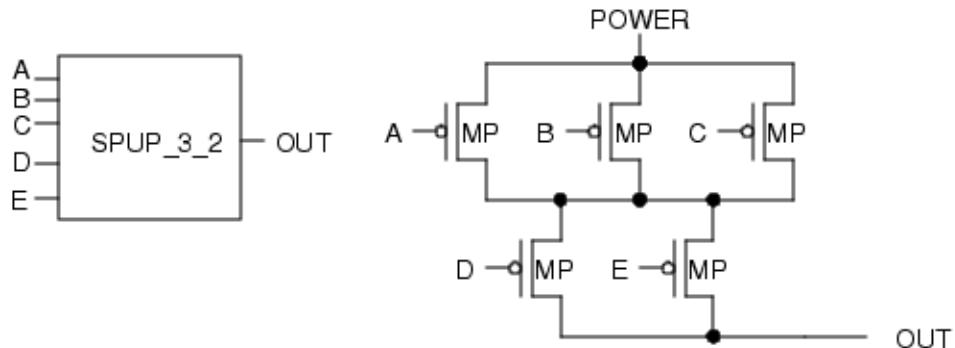
Calibre nmLVS represents serial pulldowns as stand-alone gates when they are not contained in complete gates, or when they are contained in AOI gates but complex gate recognition is turned off by including the [LVS Recognize Gates](#) SIMPLE statement in the rule file.

- **SPUP $_n1_n2_{\dots}nm$** (serial-parallel up)

CMOS serial-parallel pullup. This is a series of m parallel groups consisting of n_1, n_2, \dots, n_m transistors, respectively, leading to a power net.

Calibre nmLVS considers the inputs to the transistors in each parallel group logically equivalent. In [Figure 12-34](#), signals A, B, and C are all interchangeable, and signals D and E are also interchangeable.

Figure 12-34. SPUP_3_2



Calibre nmLVS 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.

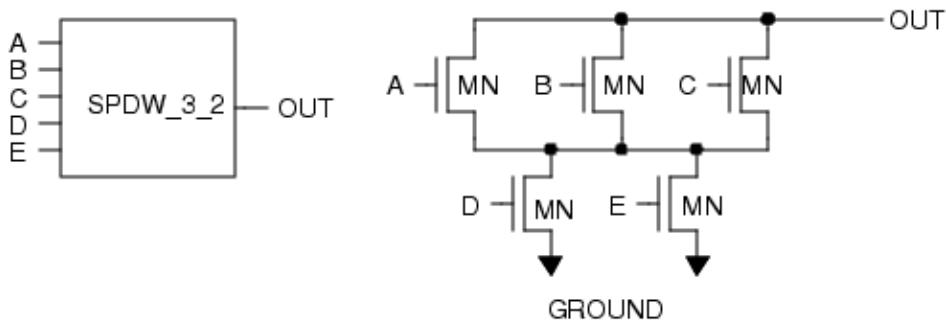
The order of the parallel groups in an SPUP gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name `SPUP_n1_n2_..._nm` always has $n_1 \geq n_2 \geq \dots \geq n_m$. The group of three parallel MP transistors connected to A, B, and C, respectively could have been placed below the group of two parallel MP transistors connected to D and E, respectively.

- **SPDW_n1_n2_..._nm** (serial-parallel down)

CMOS serial-parallel pulldown. This is a series of m parallel groups consisting of n_1, n_2, \dots, n_m transistors, respectively, leading to a ground net.

Calibre nmLVS considers the inputs to the transistors in each parallel group logically equivalent. In [Figure 12-35](#), signals A, B, and C are all interchangeable, and signals D and E are also interchangeable.

Figure 12-35. SPDW_3_2



Calibre nmLVS 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 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 $n1 \geq n2 \geq nm$. In [Figure 12-35](#), they could have been placed below the group of two parallel MP transistors connected to D and E, respectively.

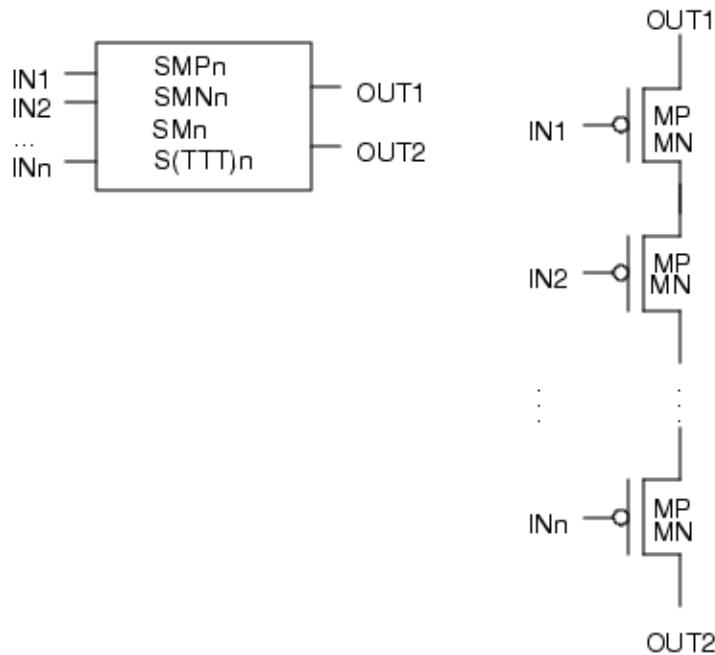
- **SMP n , SMN n , SM n , S(TTT) n**

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.

Calibre nmLVS 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 12-36](#), signals IN1, IN2, IN n are all interchangeable, and signals OUT1 and OUT2 are also interchangeable.

Figure 12-36. SMP n , SMN n , SM n , S(TTT) n Series



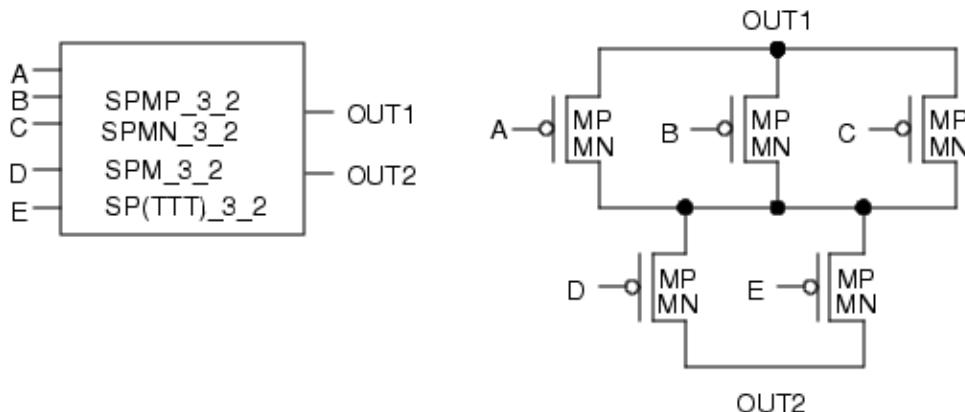
- **SPMP_** $n_1_{n_2_{\dots_{n_m}}}$, **SPMN_** $n_1_{n_2_{\dots_{n_m}}}$, **SPM_** $n_1_{n_2_{\dots_{n_m}}}$, **SP(TTT)_** $n_1_{n_2_{\dots_{n_m}}}$

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 n_1, n_2, n_m transistors, respectively, connected between any two nets (other than power in the case of MP, or ground in the case of MN).

Calibre nmLVS considers the inputs to the transistors in each parallel group logically equivalent and the two outputs logically equivalent. In [Figure 12-37](#), signals A, B, and C are all interchangeable, signals D and E are interchangeable, and signals OUT1 and OUT2 are interchangeable.

Figure 12-37. SPMP_3_2, SPMN_3_2, SPM_3_2, SP(TTT)_3_2



The order of the parallel groups in an 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 $_n1_n2_{\dots}_nm$ always have $n1 \geq n2 \geq \dots \geq nm$. 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 in series.
- (*expression*) denotes a substructure.
- *expression* * *n* denotes a parallel group of *n* transistors, connected in series with the structure described by *expression*.
- *expression* + *n* denotes a series of *n* transistors, connected in parallel with the structure described by *expression*.

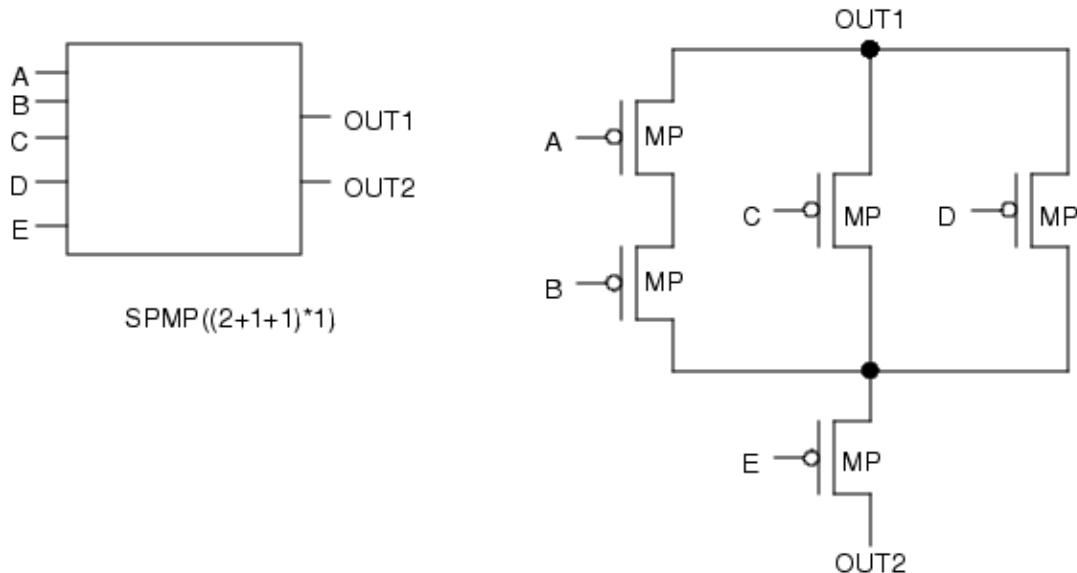
The transistor gate pins form the input pins of the structure. The output pins of the structure are the two nets at the “top” and “bottom” of the structure. There can be any number of input pins but there are always exactly two output pins.

Inputs to transistors connected in parallel are logically equivalent, as are transistors connected in series. In general, the order of any group of substructures connected in parallel is interchangeable, and the order of any group of substructures connected in series is also interchangeable. The two outputs are also logically equivalent.

You can prevent the formation of high level serial-parallel structures by including the [LVS Recognize Gates SIMPLE](#) statement in the rule file.

In [Figure 12-38](#), 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.

Figure 12-38. SPMP((2+1+1)*1) Structure

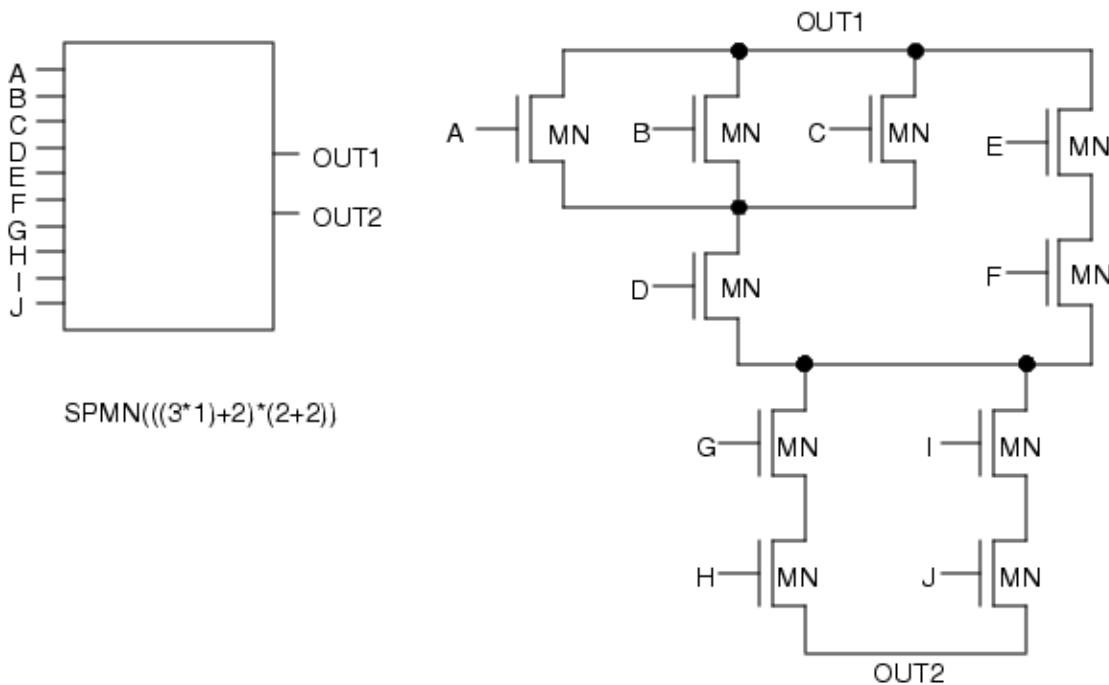


In [Figure 12-39](#), the following signals are interchangeable:

- 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.

Figure 12-39. SPMN(((3*1)+2)*(2+2)) Structure



Related Topics

[Logic Gate Recognition](#)

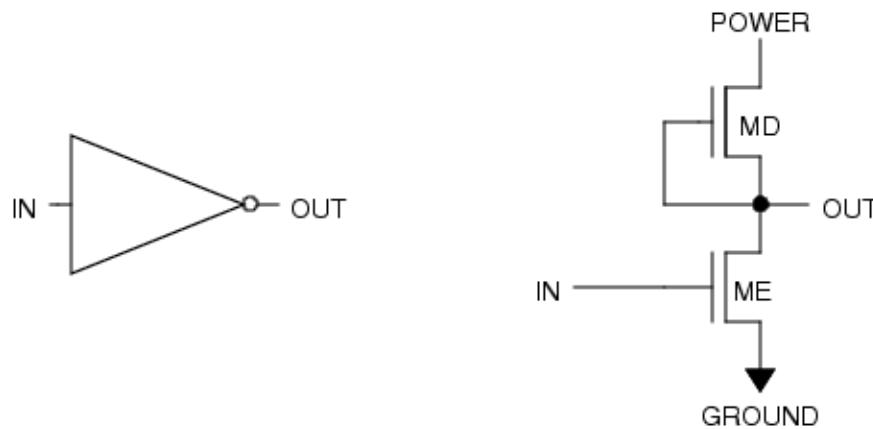
Regular NMOS Gates

Regular NMOS gates are recognized by default. Specific topologies are shown by gate type.

For each of the following descriptions, the default gate names appearing in the LVS report are used. If the [LVS Recognize Gates WITH SUBSTRATE](#) keyword is specified, then recognized gate names have suffixes that are either a “v” when the gate does not satisfy the WITH SUBSTRATE criteria, or at least one “b” otherwise.

- **INV** — NMOS INVerter.

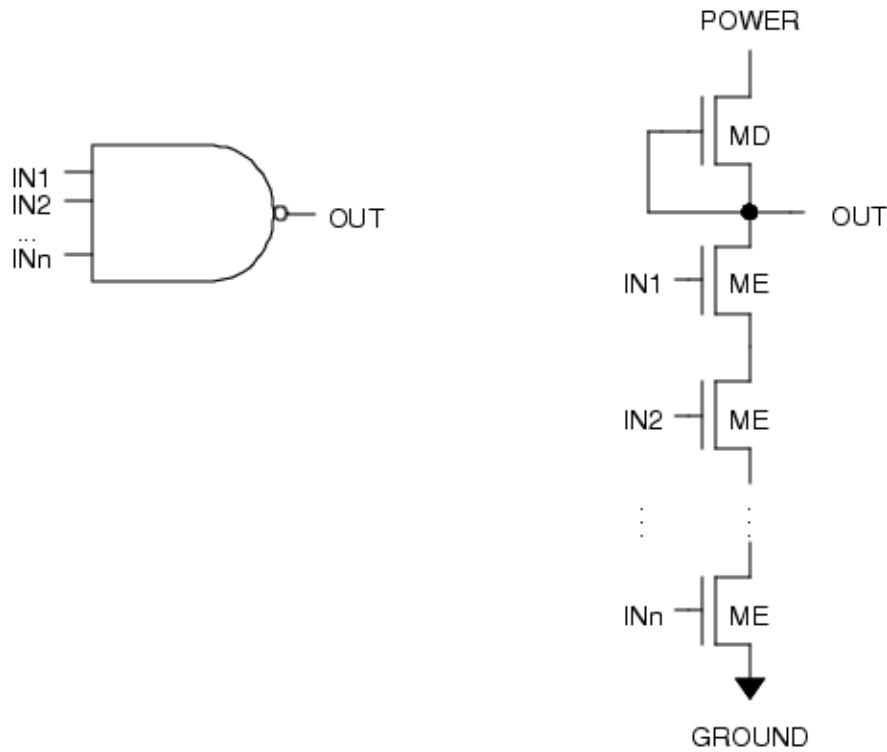
Figure 12-40. INV



- **NAND n** — n -input NMOS NAND.

Calibre nmLVS considers all input pins of a NAND n gate logically equivalent. In the next figure, signals IN1, IN2, IN n are all interchangeable.

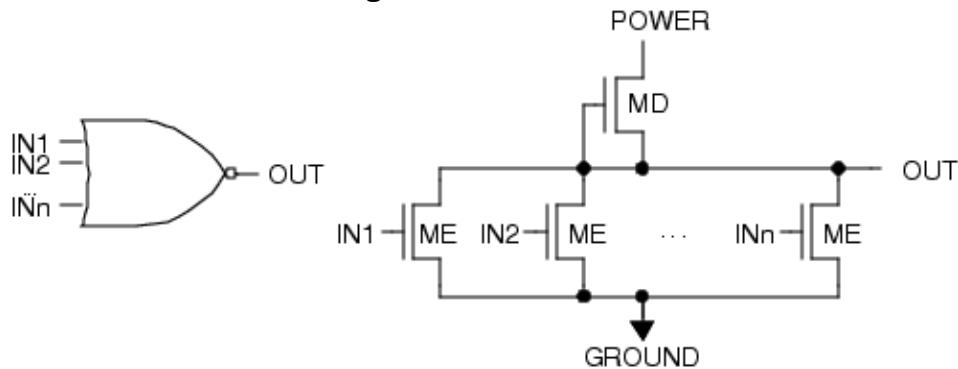
Figure 12-41. NAND n



- **NOR n** — n -input NMOS NOR.

Calibre nmLVS considers all input pins of a NOR n gate logically equivalent. In the next figure, signals IN1, IN2, IN n are all interchangeable.

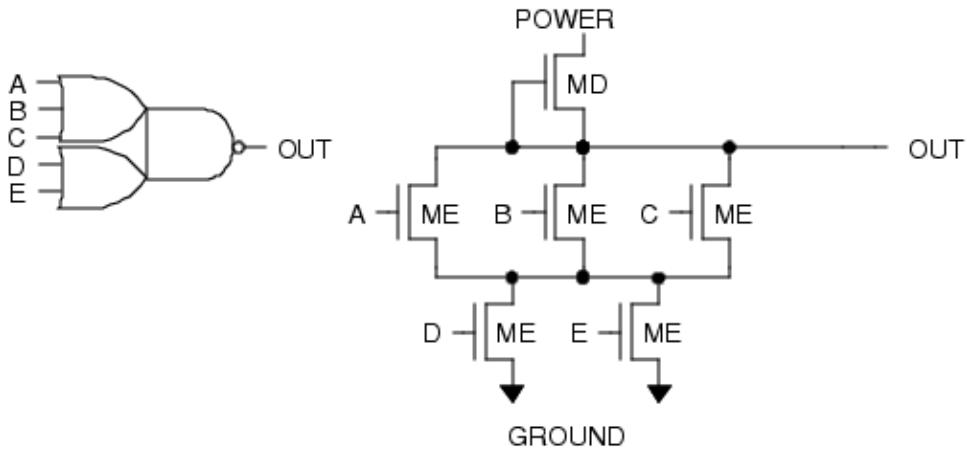
Figure 12-42. NOR n



- **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.

Calibre nmLVS considers the input pins of each one of the Or structures in an OAI gate logically equivalent. In the next figure, signals A, B, and C are all interchangeable, and signals D and E are also interchangeable.

Figure 12-43. OAI_3_2



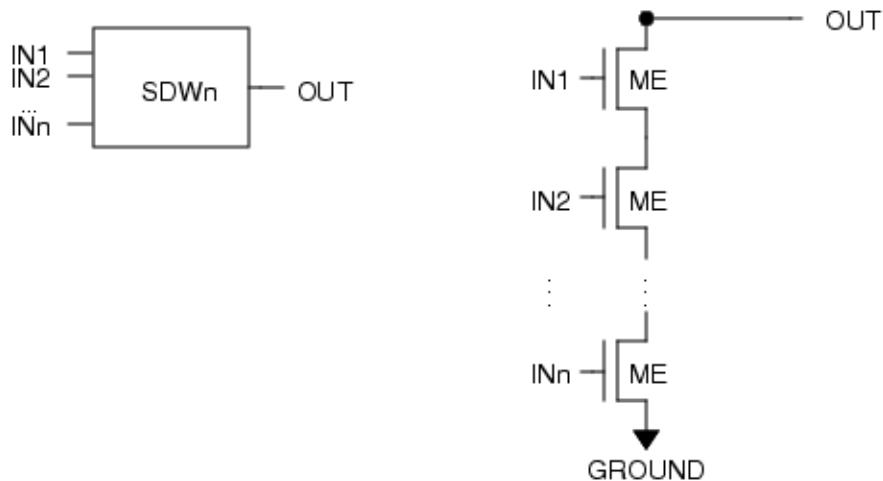
The order of the parallel pulldown groups in an AOI gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name OAI_n1_n2_..._nm always has $n1 \geq n2 \geq nm$. In the next figure, the group of three parallel ME transistors connected 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, Calibre nmLVS instead forms structures of type SPDW from the pulldown part of the gate.

- **SDW n** — n -input NMOS serial pulldown.

Calibre nmLVS considers all input pins of an SDW n gate logically equivalent. In the next figure, signals IN1, IN2, INn are all interchangeable.

Figure 12-44. SDW_n

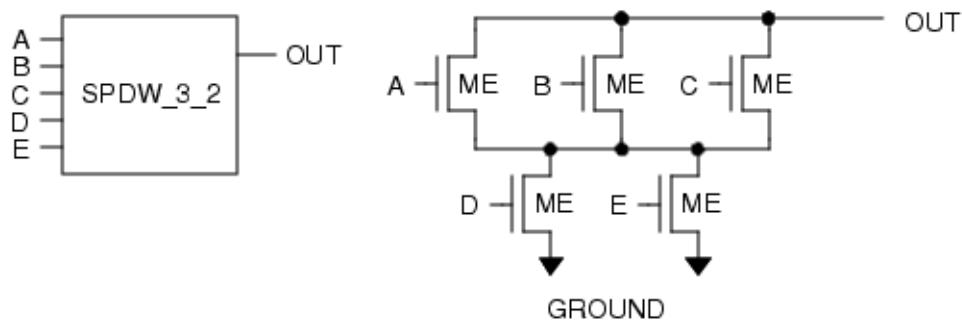


Calibre nmLVS represents NMOS serial pulldowns as standalone gates when they are not contained in complete gates.

- **SPDW_n1_n2_..._nm** — NMOS serial-parallel pulldown structure. This is a series of m parallel groups consisting of n_1, n_2, nm transistors, respectively, leading to a ground net.

Calibre nmLVS considers the inputs to the transistors in each parallel group logically equivalent. In the next figure, signals A, B, and C are all interchangeable, and signals D and E are also interchangeable.

Figure 12-45. SPDW_3_2



Calibre nmLVS 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 $n_1 \geq n_2 \geq nm$. In the preceding figure, the group of three parallel ME transistors connected to A, B, and C, respectively, could have been

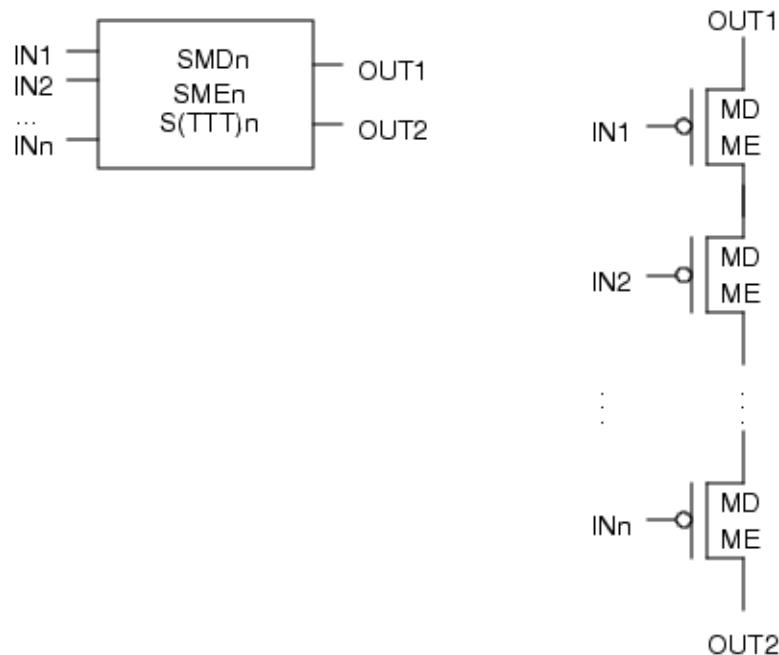
placed below the group of two parallel ME transistors connected to D and E, respectively.

- **SMD n , 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.

Calibre nmLVS 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 the next figure, signals IN1, IN2, IN n are all interchangeable, and signals OUT1 and OUT2 are also interchangeable.

Figure 12-46. SMD n , SME n , S(TTT) n



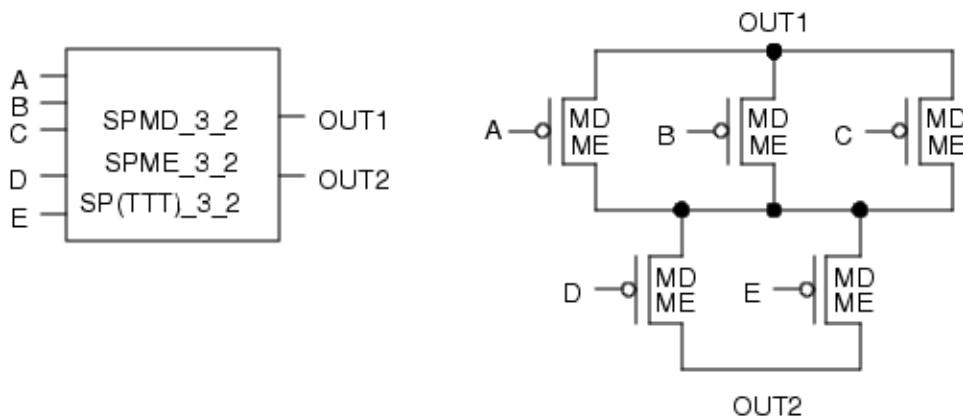
- **SPMD_n1_n2_..._nm, SPME_n1_n2_..._nm, SP(TTT)n_n1_n2_..._nm**

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 n_1, n_2, 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).

Calibre nmLVS considers the inputs to the transistors in each parallel group logically equivalent and the two outputs logically equivalent. In the next figure, signals A, B, and C are all interchangeable, signals D and E are interchangeable, and signals OUT1 and OUT2 are interchangeable.

Figure 12-47. SPMD_3_2, SPME_3_2, SP(TTT)_3_2



The order of the parallel groups in an 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 $\text{SPMD}_n1_n2_{\dots}nm$ and $\text{SPME}_n1_n2_{\dots}nm$ always have $n1 \geq n2 \geq nm$. In the preceding figure, 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 [SP*\(*expression*\) CMOS structures](#).

Related Topics

[Logic Gate Recognition](#)

LDD Gates

LDD devices are MOS transistors with non-swappable source and drain pins. 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 Built-in 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 or ground nets. These gates include:

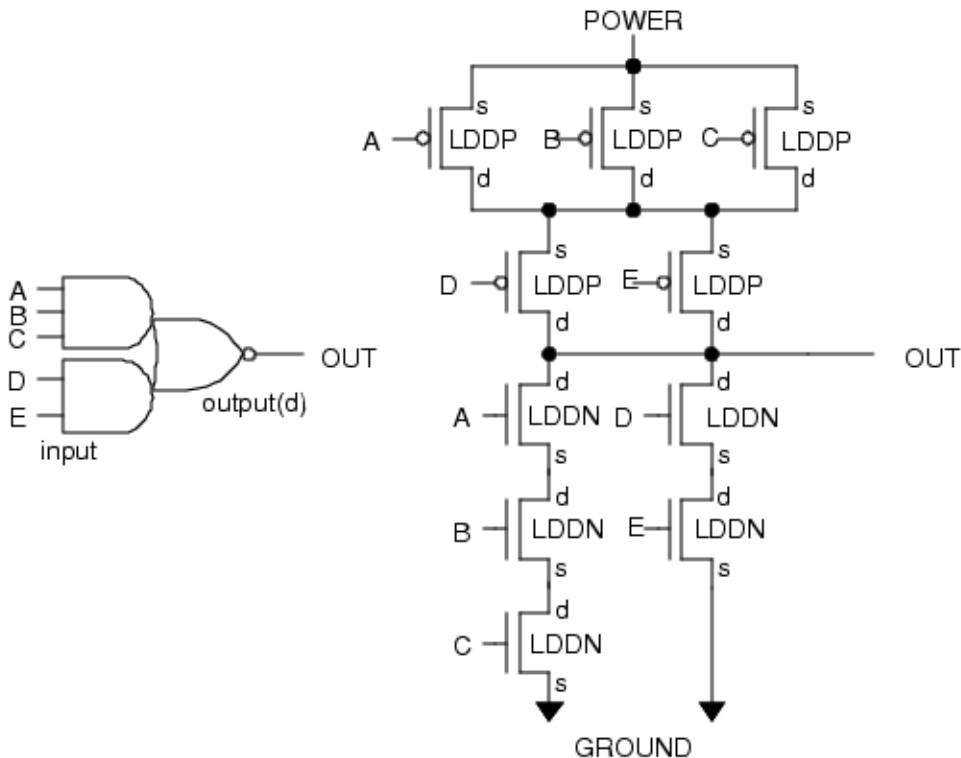
INV	NAND n
NOR n	AOI_ $n1_n2_\dots_{nm}$
OAI_ $n1_n2_\dots_{nm}$	SUP n
SDW n	SPUP_ $n1_n2_\dots_{nm}$
SPDW_ $n1_n2_\dots_{nm}$	

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 *output(d)*, while the output of a regular MOS gate is called *output*. The (d) means that the output net is connected to drain pins of LDD transistors.

[Figure 12-48](#) shows a CMOS-style AOI_3_2 gate made with LDD-type transistors.

Figure 12-48. LDD AOI_3_2



LDD Non-Voltage Gates

Non-voltage gates are gates that do not require power or ground nets. These gates include the following:

SLDDN n

SLDDN

SLDDP n

SLDDD n

SPLDDN $_n1_n2_\dots_{nm}$

SPLDD $_n1_n2_\dots_{nm}$

SPLDDP $_n1_n2_\dots_{nm}$

SPLDDD $_n1_n2_\dots_{nm}$

SPLDDN(*expression*)

SPLDD(*expression*)

SPLDDP(*expression*)

SPLDDD(*expression*)

S(TTT) n

SPLDDE(*expression*)

SP(TTT)(*expression*)

SP(TTT) $_n1_n2_\dots_{nm}$

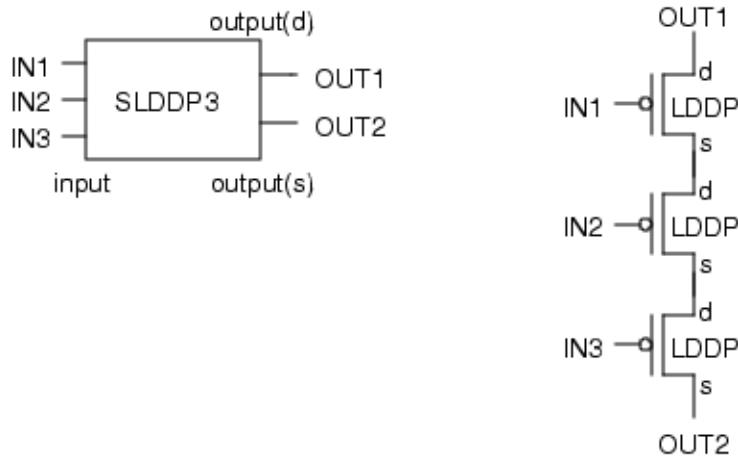
The string TTT stands for any device type classified a LDD-type MOS device with an [LVS Device Type](#) statement. It can also stand for a regular MOS device type when LVS Builtin Device Pin Swap NO is specified. For more information on the (*expression*) notation, refer to [SP*\(*expression*\)](#) CMOS structures.

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(*expression*) corresponds to SPME(*expression*).
4. An LDD-type non-voltage gate has two non-swappable outputs. One is connected only to source pins and is called *output(s)*. The other is connected only to drain pins and is called *output(d)*. Recall that the outputs of a regular MOS gate are both called *output*.

Figure 12-49 shows a SLDDP3 gate.

Figure 12-49. SLDDP3



Mixed Gates

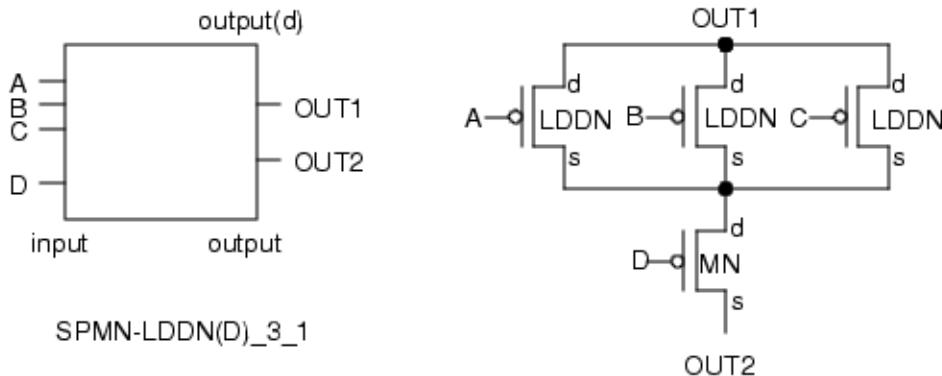
LDD-type and regular MOS transistors form mixed non-voltage gates, such as gates that do not require power or ground nets.

Mixed gates are composed of an S or SP gate of one type in series with a unique single transistor of the other type. The mixed gate is formed only if there is a *unique* available single transistor of the other type. One regular and one LDD-type transistor in series also form a mixed gate. The LDD-type transistors and the regular MOS transistors must be of the same kind (D, E, N, P, or generic) to form a mixed gate.

A mixed gate has two outputs, one connected to regular MOS transistors is called *output*. The other is called *output(d)* if it is connected to drain pins of LDD-type transistors or *output(s)* if it is connected to source pins of LDD-type transistors.

A mixed gate contains the name of both transistor types that compose it, the single transistor first, for example: SPMN-LDDN(D)_3_1. The (D) means that the output net connected to the LDD transistors is connected to drain pins. [Figure 12-50](#) shows this gate.

Figure 12-50. SPMN-LDDN(D)_3_1



Related Topics

[Logic Gate Recognition](#)

X+ Transistors

X+ devices are MOS transistors with component subtypes that begin 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+ transistors normally do not form logic gates by default. 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. See “[X+ Devices](#)” on page 392.

Device and Pin Swapping Checks in Logic Gates

You can check the physical order of connections to logic gate inputs, including the physical order of parallel device groups.

The structures are considered logically equivalent (swappable) in either of the following cases:

- Specify that logic gate recognition is not to be performed by setting [LVS Recognize Gates](#) NO in your rule file.
- Use component subtypes X+ as described in the section “[X+ Transistors](#)” on page 461.

Pin Swapping

LVS comparison 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.

Calibre nmLVS forms equivalence of input pins in logic gates. Refer to section “[Logic Gate Recognition](#)” on page 439 for a special case of this capability. For other component types, there are several methods for designating logically equivalent pins.

Default Pin Swapping for Devices

Calibre nmLVS always considers pins that have identical layer names to be logically equivalent (swappable).

Calibre nmLVS 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.

Calibre nmLVS 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 Built-in Device Pin Swap](#) statement.

Pin swapping for device instances can be reported as errors in the LVS Report through the [LVS Report Pinswapped](#) statement.

Rule File Pin Swap Lists

Calibre nmLVS designates pins of extracted layout devices as logically equivalent by placing the pin names in a single *pin_swap* list in a Device statement. For example:

```
DEV C cap1 poly m1 (POS NEG) // swappable POS, NEG
```

Pins on a single layer in the **Device** statement are always swappable, such as the m1 layer here:

```
DEV F seed m1(p1) m1(p2) poly(p3) // swappable p1, p2
```

Affected Database Systems — Calibre nmLVS 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 Calibre nmLVS execution:

- Calibre geometric database systems (for example, GDSII, OASIS, or ACSII)
- 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. Calibre nmLVS 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 Calibre nmLVS 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, Calibre nmLVS 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 Calibre nmLVS 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 “[Subcircuit Definitions](#)” on page 695 for more information), or if they are instances of [LVS Box](#) cells (although LVS Box does support pin swap lists). Instances of regular (non-empty) hcells in hierarchical circuit comparison are not primitive and thus do not inherit pin swappability from Device operations. 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. Calibre nmLVS 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) cap1 poly m1 (POS NEG) // C with model A
DEVICE C(B) cap2 m1 m2           // C with model B
```

SPICE:

```
C1 1 2 $[A] $$ C with model A
C2 3 4 $[B] $$ C with model B
```

C1 receives pin swappability from the DEVICE C(A) operation. Its pins are swappable. C2 receives pin swappability from the Device C(B) operation. Its pins are not swappable unless indicated as 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. 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. 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. Its pins are not swappable unless indicated as swappable by other means. C2 receives pin swappability from the DEVICE C operation. Its pins are swappable. C3 receives pin swappability from the DEVICE C operation. 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. 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).

Pin Equivalence with SPICE as Layout System

When a SPICE netlist represents the layout, Calibre nmLVS-H utilizes pin swappability information from a Device operation during circuit comparison. This occurs when the specification statement Layout System SPICE is indicated.

When your [Layout System](#) is geometric, the previous behavior applies when you specify these command lines:

```
calibre -spice ... -lvs ...
calibre -lvs -hier ...
calibre -lvs -flatten ...
```

These command lines use SPICE as an intermediate representation for the layout, and the source can be of any allowed type. This is useful when user-defined devices with swappable pins are entered as primitive subcircuits in SPICE netlists.

Generally, there is no logical equivalence between hcell pins (hcells are hierarchically corresponding cells in Calibre nmLVS-H).

There are two exceptions to this rule, namely, trivial pin swappability (see “[Trivial Pin Swappability](#)” on page 496) and pin swapping in memory cells and containing blocks (see “[SRAM Bit-Cell Recognition](#)” on page 501).

Device Property Tracing

Calibre nmLVS 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	467
Built-In W/L Partner Properties	468
Device Count Comparison After Reduction	469

Built-In Property Classification

The Calibre nmLVS circuit comparison module recognizes certain property names as built-in properties for the purpose of device reduction and for other processing.

For example, Calibre nmLVS computes effective values for built-in properties when it reduces devices in series and parallel. See “[Device Reduction](#)” on page 396.

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:

A, AD, AS, C, L, P, PD, PS, R, W

denote area, area of drain, area of source, capacitance, length, perimeter, perimeter of drain, perimeter of source, resistance, and width, respectively. These properties are calculated by default for various built-in devices during parallel and series device reduction. 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 property names to trace. The device for which this is particularly important is the diode (element D). The SPICE netlister shows area as “AREA= n ” and perimeter as “PJ= m ”. 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 contain different property names for source properties than the default property names. (Note this does not apply to the layout.) Doing so overrides the built-in property name for the source netlist. This allows you to take advantage of the default effective property computation for LVS Reduce statements, without having to write

a user-defined effective property computation to accommodate the non-default source property name.

For example, this statement:

```
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 has property name W. The default effective property computations for MOS device reduction recognize that WIDTH corresponds to W in the source, and makes the appropriate calculations internally.

Built-In W/L Partner Properties

Calibre nmLVS automatically reads from the input database L properties *if they are required* for the calculation of effective W values during device reduction. When required, L properties are read even when they are not traced themselves.

W and L are sometimes called “partner” properties because one is often required to calculate effective values for the other during device reduction.

Specifically, Calibre nmLVS 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, Calibre nmLVS automatically reads W properties if they are required for the calculation of effective L values.

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.

By default, Calibre nmLVS reads any SPICE netlist parameter starting with the letter W as the width property. Similarly, any SPICE netlist parameter starting with the letter L is read as the length property. This allows some inherent flexibility in how you construct your Trace Property statements when using such parameters. If you want only the characters “W” and “L” to be read as width and length, then you should specify [LVS Spice Strict WL YES](#).

Related Topics

[Device Property Tracing](#)

Device Count Comparison 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 is 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:

Example 12-3. Comparing Device Counts After Reduction

```
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

M2 1 2 3 4 P

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

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)

The Device operation is not necessary if you compare SPICE to SPICE.

You can perform similar device count comparisons for series reduction by including the following statement in the rule file:

```
LVS REDUCE MP SERIES [  
    EFFECTIVE M  
    M=SUM(M) // Add up M values during series reduction.  
]
```

The comparison results for the corresponding series reduction cases would be identical to the parallel ones shown in the preceding example.

See also [LVS Spice Replicate Devices](#) in the *SVRF Manual*.

Chapter 13

Hierarchical LVS

Calibre nmLVS-H is a hierarchical LVS application. Calibre nmLVS-H maintains design hierarchy and exploits this hierarchy to reduce processing time, memory use, and LVS discrepancy counts.

The command line options for Calibre nmLVS are discussed under “[Calibre nmLVS and Calibre nmLVS-H Command Line](#)” on page 45. The -hier, -hcell, and -spice options are of particular interest. You can also use Calibre Interactive to specify a hierarchical LVS run.

Hcells	474
Hierarchy Preservation by Layer Operations.....	491
Cell Pushdown	495
Hierarchical LVS Comparison.....	496
LVS Box Cells.....	497
High-Short Resolution.....	498
SRAM Bit-Cell Recognition	501
Parameterized Cells.....	503
Logic Injection	505

Hcells

The term *hcell* stands for *hierarchically corresponding cell*. Hcells are cells that exist in both the layout and source designs and that (presumably) perform the same function. Calibre cannot determine if cells are identical in function because mask layout formats or SPICE netlists do not contain such information. Also, simply because two cells may have the same name is not necessarily a guarantee they have the same topology or device properties. So the user must specify cell correspondence in hierarchical LVS flows.

Calibre nmLVS-H operates using something called an hcell list. If you do not specify hcells, then the hcell list is empty. An empty list of hcells causes LVS-H circuit comparison to run flat. See “[Hcells and Circuit Extraction](#)” on page 276 for a discussion of the behavior during circuit extraction.

Hcells may be specified in the rule file with [Hcell](#) specification statements, in an external cell correspondence file referenced by the [-hcell](#) command line option, in an automatically generated hcell list using the [-genhcells](#) option, or implicitly with the [-automatch](#) command line option (-automatch does not apply during netlist extraction). Each of these methods may establish its own list of hcell name pairs. The lists are combined by the tool at runtime and used as a single hcell list. All cell name pairs are treated equally, regardless of how they were established.

The hcell set influences tool performance, so you should select hcells carefully. It is most desirable to select hcells while the design is being constructed. If you do not have an hcell list for an existing design, you can select your hcell list using the Calibre Interactive — LVS interface. See “[Performing Hcell Analysis in Calibre nmLVS](#)” in the *Calibre Interactive User’s Manual*. You can also do this from the Query Server. For those details, see “[Hcell and Hierarchical Analysis](#)” in the *Calibre Query Server Manual*. See “[Hcell Recommendations for Digital Place and Route Flows](#)” in the *Calibre Solutions for Physical Verification* manual for related hcell selection criteria. The [LVS Hcell Report](#) specification statement is also useful in this context. The format details are discussed under “[Calibre LVS Hcell Report Format](#)” on page 483.

In general, an hcell list should reference cells from a standard cell library, large macro blocks, and embedded memory blocks. For large memory blocks, including one or two matching levels of hierarchy within the core array of the memory is often desirable.

LVS-H selectively expands certain cells when it is beneficial for performance. Providing an exhaustive list of all cells in the hierarchy is not needed and can be counter-productive. Specifying certain cells like via cells or other very small cells (typically containing no devices in their hierarchy) could impede the performance-improvement heuristics in hierarchical runs. Omitting a cell from the hcell list does not necessarily mean it is entirely expanded because it may have hcells in its hierarchy that are preserved. If it is the intent to preserve hierarchy regardless of the performance cost, then using a comprehensive cell list serves that purpose.

A line in an external hcell list (referenced with the -hcell option or generated by the -genhcells command line option) appears as follows:

```
layout_cell source_cell
// this is a comment.
```

The *layout_cell* is a layout cell name and *source_cell* is the name of the corresponding source subcircuit. The two arguments are separated by space or tab characters. Lines where // are the first two non-whitespace characters are ignored and are treated as comments. Comments trailing non-whitespace characters are not allowed. A STRICT keyword can appear in an automatically generated hcell list. This keyword is for the internal use of the LVS module.

Case sensitivity of hcell names in 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 nmLVS to circuit extraction as well as circuit comparison. See “[Case-Sensitive Handling of Names](#)” on page 375 for more information.

The [LVS Exclude Hcell](#) specification statement in the rule file prevents specified cells from serving as hcells under any circumstances.

Other Calibre applications may behave differently with respect to hcells than what is described in this section. For information on how Calibre nmDRC-H handles hcells, see “[Calibre nmDRC-H Use of Hcells](#)” on page 151.

Hcell Correspondence

Cell correspondence between layout and source is defined in an hcell list. An hcell list has layout cell names in the left column with corresponding source cell names in the right column.

```
// layout source
ABC      DEF
GHI      JKL
UVW      XYZ
```

In a pair of hcells, the layout cell name and corresponding source cell name may be the same, or they may be different. You can specify a 1-to-many relationship by placing a layout name in several lines with different source names:

```
// layout source
ABC      DEF
ABC      MNO
ABC      PQR
```

Similarly, you can specify a many-to-1 relationship by placing a source name in several lines with different layout names:

```
UVW      XYZ
RST      XYZ
OPQ      XYZ
```

However, m-to-n (many-to-many) relationships are not allowed, for example, this results in an error during circuit comparison:

```
UVW      XYZ
RST      XYZ
OPQ      XYZ
OPQ      GHI // error. OPQ mapped previously to XYZ,
              // which is mapped to three different names.
```

Many-to-many cell correspondence is a global error that causes hierarchical LVS comparison to abort without comparing individual cells. A pair of cell names that leads to the many-many correspondence is indicated in the run transcript. In the previous example, the following message may appear in the transcript:

```
ERROR: Correspondence "OPQ" "GHI" leads to a many-many correspondence.
```

Many-to-many cell correspondence is also indicated with a respective secondary comparison status in the LVS Report. The primary comparison status is INCORRECT because the cells cannot be compared.

Note

-  Hcells can be cloned by the Layout Clone Rotated Placements and Layout Clone Transformed Placements YES statements. Cloning of hcells can cause many-to-many cell correspondence errors to occur. The clone names contain “\$C<n>\$ strings in them, where <n> is an integer. If such errors occur, you need to adjust the hcell list to mitigate them.
-

Many-to-many correspondence is not checked in the circuit extraction stage because only layout cells matter in that context.

When Calibre nmLVS-H identifies corresponding cells during the netlist comparison phase, conflicts can prevent two cells from being matched. A conflict exists between cells A and B if both of the following are true:

Cell A already corresponds to cell C.

Either cell B contains an instance of cell C, or C contains an instance of B. Containment can be direct or indirect.

Under such conditions, cells A and B cannot correspond without creating a cycle in the merged hierarchy. (It is possible to create an hcell correspondence cycle in the design hierarchy when hcell containment topology on the layout side does not match that on the source side.) Thus,

LVS refrains from matching the cells even if they should correspond based on explicit hcells or -automatch. When such a conflict occurs, the following warning is produced:

WARNING: Cell <name> cannot be made to correspond with cell <name> due to a hierarchical conflict.

It is possible this situation can be triggered by the Query Server's hcell and hierarchical analysis interface. The Query Server's hcell analyzer selection processing identifies correspondences in an arbitrary order, which can differ from the order used by LVS. Hence, different conflicting cells may be excluded by the Query Server versus LVS, and consequently the selected hcells from the Query Server may lead to poor LVS performance. Typically, you need to resolve such conflicts by modifying the hcell list. Otherwise, the set of corresponding cells is unpredictable and sensitive to arbitrary ordering. See also “[Hierarchical Cells Forming a Cycle](#)” on page 588.

Using the following list, where TOP is the primary cell name, is essentially equivalent to running flat LVS:

```
TOP TOP
```

However, for SPICE-to-SPICE comparisons, it often runs faster than not using it.

By default, primitive devices correspond by component type. You can override this for user-defined devices by including device element names in the hcell correspondence file. The cell correspondence file then exclusively determines the correspondence of the user-defined primitive devices. This mapping does not work for built-in devices.

Wildcards in Hcell Names

The hcell list names may specify * wildcards, which match zero or more characters. Wildcards can be used in layout or source names. Wildcard patterns in hcell lists behave as follows:

- If wildcards are used for only a layout or the source name (not both) on a single line of an hcell list, then the wildcards match any strings in that design's cell names according to the pattern specified for the name. If more than one cell name matches the pattern, this establishes a many-to-one hcell correspondence. For example, assume this hcell list:

```
inv*_1* inv_1x
```

Any layout cell names starting with the string “inv” followed by any number of characters, followed by “_1”, followed by any number of characters, correspond to the source name `inv_1x`.

- If wildcards are used for both the layout and the source names on a single line of an hcell list, then only the wildcards on the layout side match strings. The wildcards on the source side are placeholders for the strings that the layout wildcards match and are replaced by any matched strings to form candidate source cell names. For any candidate

source cell name that exists in the source design, that cell is used for hcell correspondence. For example, suppose these are layout cell names:

```
blue1
blue2
green
```

and these are source cell names:

```
red0
red1
red10
```

Assume this hcell list line:

```
blue* red*
```

The cell names blue1 and blue2 match the layout pattern. The match of the string “1” causes a match of the source name red1. The match of the string “2” causes no match of a source name.

- If wildcards are used for both the layout and source names on a single line of an hcell list, then each layout pattern wildcard matches only the longest possible string of characters in each layout cell name. The longest possible match for a single wildcard is unambiguous. For multiple wildcards in a pattern, each wildcard, from left to right, matches the longest possible string and leaves the remaining portion of a name to be matched by the next wildcard. So, for this name:

```
buf$$1x$1x
```

and this wildcard pattern:

```
buf*${}
```

The first wildcard matches “\$\$1x” and the second wildcard matches “1x”.

- If wildcards are used for both the layout and source names on a single line of an hcell list, then the number of wildcards used in the layout pattern must match the number of wildcards used in the source pattern. If there is a mismatch, and the layout pattern matches at least one cell name, then a runtime error results, and the LVS comparison overall status is NOT COMPARED. Any strings that match the layout wildcards are substituted for the wildcards in the source pattern, one-for-one, and in the order the wildcards occur. For example, this hcell list pair is disallowed because the wildcard count does not match in the layout and source patterns:

```
blue*_2x red_*2x // bad. wildcard mismatch.
```

The preceding usage results in a runtime error and an overall status of NOT COMPARED in an LVS report. This is OK:

```
blue*_2x red*_2x
```

The wildcards having the same colors are in a one-to-one correspondence for string matching and substitution.

- Unlike the Hcell specification statement, names using wildcards should not usually be quoted in a -hcell list because the quotes are treated as literal.

Care must be taken that wildcards do not lead to many-to-many cell correspondence, which causes an error. For example, specifying this:

```
A* A
*B B
```

could cause a match where layout cells AB and A1B match source cells A and B, which causes an error because of the ambiguity of the match. Changes to layout cell names can lead to ambiguous hcell name matches when wildcards are used.

Related Topics

[Hcells](#)

Hcells and Circuit Comparison

Hcells are compared as hierarchical entities. For each pair of hcells, the layout cell is compared to the paired 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 (flat).

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 the automatic correspondence by including the device names in the hcell list. The hcell list then exclusively determines their correspondence. Warnings are issued for cell names that do not exist in the input data.

The -automatch option causes cells that correspond by name in layout and source to serve as hcells. If using -automatch, you should ensure that subcircuits in the source serve the same function as cells in the layout with the same names. In most cases, a well-chosen hcell list delivers better performance than -automatch.

Indications of a Non-Optimal Hcell List in LVS Comparison

Certain indications are given in the run transcript and the LVS Report if the hcell list is not optimal.

These are indicators that an hcell list is too small:

- The overall memory consumption (LVHEAP) during LVS comparison is unusually high.
- The INITIAL NUMBERS OF OBJECTS section of the LVS Report shows large numbers (1E5 or more) of primitive devices.

Other indicators of a problematic hcell list:

- For certain hcells, if there are more instances of a component in the layout or source at a lower level of the hierarchy, but the reverse is true farther up in the hierarchy, then the hcell pair should probably be removed from the list.
- There are many discrepancies related to pin swapping for a given hcell.

If either of these latter situations occurs, try specifying [LVS Exclude Hcell](#) for the affected hcells and see if that resolves the issue.

- Hcell instances may be expanded by the circuit comparison algorithm if it determines that expansion is necessary in order to match the layout and source hierarchies. This is indicated in the transcript by “EXPANDING unbalanced cells” entries. For more information refer to [LVS Expand Unbalanced Cells](#).

When cells contain large numbers of flattened instances, memory allocations that occur during the stages “FLATTENING non-corresponding cells” and “EXPANDING unbalanced cells” are reported in the run transcript. Performance may be improved by identifying hcells with very large instance counts and finding contained cells that may function as additional hcells. Specifying contained cells identified in this way as hcells helps reduce overall per-cell contained instance counts and can result in improved performance.

To facilitate this identification, the additional transcript memory information includes the following:

- Layout or source design.
- Cell name.
- Instance count.
- Memory in use (LVHEAP and MALLOC).
- Elapsed time.

Example output:

```

Resolving DEEP SHORTS ...
DEEP SHORTS resolved. CPU TIME = 2 REAL TIME = 2 LVHEAP = 1692/1737/1737
MALLOC = 1794/1794/1794 ELAPSED TIME = 32

EXPANDING unbalanced cells ...

Source cell Z1374F now contains 1000071 flattened instances.
LVHEAP = 1880/1882/1882 MALLOC = 1932/1932/1932 ELAPSED TIME = 32

Layout cell Z1374G now contains 1000036 flattened instances. LVHEAP =
2345/2345/2345 MALLOC = 2486/2486/2486 ELAPSED TIME = 34

EXPANDING unbalanced cells in LAYOUT cell Z1374G, SOURCE cell Z1374F
C1802 in Z1374F ( 28 instances, SOURCE )

```

For the preceding report, the cells Z1374G and Z1374F are candidates to search for corresponding child cells to be declared as hcells.

Seed Promotion and Hcells

In certain cases, device seed promotion can occur in the circuit extraction module when devices cannot be completely formed at the level of the seed shape. The device seed can be promoted up the hierarchy and the device is recognized at a higher level. This is indicated by a *.SEEDPROM statement in the extracted SPICE netlist for the subcircuit that had the device seed promoted out of it.

This topic is discussed from the layout perspective under “[Seed Promotion](#)” on page 323. Here is an example:

Example 13-1. Seed Promotion

```

.SUBCKT cell_1 1 2 4
** N=4 EP=3 IP=0 FDC=1
*.SEEDPROM
M0 4 1 2 2 n L=5e-08 W=1.25e-07 $X=1025 $Y=850 $D=0
.ENDS

.SUBCKT cell_2
...
X4 IN1 GND Y cell_1 $T=-875 -650 0 0 $X=25 $Y=0
X5 IN1 GND Y cell_1 $T=-450 -650 0 0 $X=450 $Y=0
.ENDS

```

If seed promotion occurs out of a cell, and that hcell is preserved as an hcell during the comparison phase (including the use of -automatch), it is maintained as an hcell by default during hierarchical comparison even though devices have been promoted from it up the hierarchy. This could result in circuit comparison discrepancies where devices are present in differing subcircuits in the layout versus the source.

The [LVS Push Devices](#) YES specification statement (which is the default) allows for LVS to recover from seed promotion in many cases, and is the default behavior. However, it cannot guarantee recovery from seed promotion in all cases.

In certain cases, it may be advantageous to expand hcells during the comparison phase, where those hcells have experienced seed promotion out of them. This could mitigate discrepancies during the comparison phase. For information on expanding seed promotion cells, see [LVS Expand Seed Promotions](#). Another alternative is to cause the affected hcells not to be used as hcells. The [LVS Exclude Hcell](#) statement can be used in this case.

Layer derivation based layer promotion can cause devices to form higher up in the hierarchy without the usual seed promotion indicators. To mitigate the effects of promotion, it is best to use layer operations that tend to preserve hierarchy when deriving device seed shapes. See “[Hierarchical Degradation Effects in Layer Operations](#)” for information about layer operation effects upon hierarchy.

[LVS Use Careful AND](#) YES, or the AND operation LVSCAREFUL keyword, can be useful when the AND operation is responsible for device layer shapes being promoted. This promotion occurs before device recognition, so it may not lead to seed promotion such as when *.SEEDPROM appears in the extracted netlist.

If you are extracting MOS device properties such as AS, AD, PS, and PD for parasitic extraction flows, use of the [LVS Push Devices](#) SEPARATE PROPERTIES statement is recommended. See the *SVRF Manual* for details.

Connectivity Dependent Transformation

Two or more nets shorted together at a higher level of hierarchy in corresponding hcells 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 that occur during comparison such as unused device filtering.

The [LVS Expand Ambiguously High Shorted Hcells](#) statement controls automatic expansion of hcells with high-shorted pins that are matched in a many-to-one configuration, and that cause false LVS comparison discrepancies.

Related Topics

[Hcells and Circuit Extraction](#)

Calibre LVS Hcell Report Format

Output from LVS Hcell Report specification statement.

This report shows which hcells were matched and how, which hcells were rejected and why, and is useful for hcell list optimization for the input designs in a hierarchical LVS run.

Format

The CALIBRE LVS HCELL REPORT consists of a header followed by named sections. The form of the report is controlled using the [LVS Hcell Report](#) BY REJECTION (default) or BY HCELL keywords. The major sections are identical regardless of which report format is selected.

Header

The report header has the following entries:

REPORT FILE NAME:	<filename>
LAYOUT NAME:	<pathname> ('<primary_cell>')
SOURCE NAME:	<pathname> ('<primary_cell>')
RULE FILE:	<filename>
CREATION TIME:	<timestamp>
CURRENT DIRECTORY:	<working_directory_pathname>
USER NAME:	<username>
CALIBRE VERSION:	<Calibre_version>

The names of the fields that are completed at runtime are as shown.

SETTINGS

This section shows the rule file configuration for the following statements when the report was produced:

- [LVS Auto Expand Hcells](#)
- [LVS Compare Case](#)
- [LVS Exclude Hcell](#)
- [LVS Expand Ambiguously High Shorted Hcells](#)
- [LVS Expand Seed Promotions](#)
- [LVS Expand Unbalanced Cells](#)
- [LVS Flatten Inside Cell](#)
- [LVS Hcell Report](#)

HCELL SPECIFICATIONS

This section summarizes how the aggregate list of hcells used for the run was comprised. Entries are as follows:

```
Hcell file from <-genhcells | -hcell>:  <filename>
Command included -automatch:  <no | yes>
Hcells found in SVRF:        <no | yes>
```

The -hcell or -genhcells entries only appear if those options are used on the command line. If a custom script is specified for -genhcells, the path to the script and its contents are listed. The final line pertains to use of the [Hcell](#) specification statement (or its TVF equivalent) in the rule file.

HCELLS EXCLUDED BY LVS EXCLUDE HCELL

If any hcells are excluded by an LVS Exclude Hcell statement in the rule file, then this section of the report appears with a list of such cells.

FAUX BIN CELLS

If faux bin cells are constructed during a circuit extraction run and the LVS Hcell Report statement was active, then this section of the report appears. A faux bin cell is a pseudocell (ICV_* cell) generated by the hierarchical database constructor that can be traced to a single cell in the original layout design. The pseudocell name and the corresponding original name are listed in this section.

CORRESPONDENCE DETAILS

This is the primary section of the report. The section contains a detailed table with entries related to the handling of hcell correspondences. The meanings of the names of the columns are largely intuitive and are defined in [Table 13-1](#).

For every pair of cells that Calibre considered as a potential correspondence, this section provides information about the pair, whether the correspondence was established, and if not, why it was rejected. If a pair was established, the report shows whether any instances of the cells were expanded due to LVS Expand Unbalanced Cells YES or LVS Flatten Inside Cell. Also, if circuit extraction identified the layout cell as a high-cost hcell or was subject to layer operations that degrade hierarchy, the optimization operation and score are provided whether or not the cell was automatically expanded during comparison.

The default report form (optionally specified with the BY REJECTION keyword) and the form specified using the BY HCELL keyword are similar in many ways, but there are notable differences.

The default report contains a column named “Hcell specification,” which lists corresponding cell pairs as they were originally specified. The BY HCELL form replaces the “Hcell specification” column with a “Rejected” column that lists whether or not a pair was rejected and, if so, the reason why.

A given potential correspondence can be implied by multiple user-specified hcells, or by both hcells and the -automatch option. Such a potential correspondence is only listed once in either report form. In the default report, it is arbitrary which hcell specification and match type are listed with it. In the BY HCELL report, the correspondence appears in only one HCELL SPECIFICATION subheading (described later in this section).

The default report format contains subsections listed in [Table 13-2](#). The CORRESPONDENCES ESTABLISHED subsection always appears, and it is always the final subsection if there is more than one. The other subsections appear in arbitrary order and only when required.

In the default report, if any specified hcells cannot be found in the layout or source, then they are listed at the end of the section in a table like this:

UNMATCHED HCELLS		

Layout name	Source name	From
-----	-----	-----
<cell>	<cell>	<originating source>

The cell names appear as they do in the originating source for the hcell specification indicated in the “From” column.

The BY HCELL format contains additional lines as follows:

```
HCELL SPECIFICATION - LAYOUT NAME: <layout list name>      SOURCE NAME:  
<source list name>    FROM: <list origin>
```

This shows original names (in the original text case) that appear in the hcell lists, along with the originating source of the names: “SVRF” for Hcell statements in the rule file, or a pathname for a -hcell file or the list generated by a -genhcells script. Under each subheading, potential correspondences that were rejected are listed before those that are accepted. These lines can be particularly helpful when wildcard matching of hcell names is used.

If -automatch is specified on the command line, then the BY HCELL report includes an AUTOMATCHING subheading, and all potential correspondences for this matching method are listed afterward. Primary cells are listed here if not matched elsewhere in the report.

When an hcell specification leads to no potential correspondences, it is given a subheading in the BY HCELLS report followed by no line items (an empty subsection). This allows easy identification of those hcell specifications that did not match any actual cell pairs. The same holds for AUTOMATCHING subsections. If no cell pairs were found having matching names, and the top-level cells were explicitly matched, an empty AUTOMATCHING section will be present.

At the end of this section, there is a legend that lists the various mnemonics that actually appear as reason codes, along with their meanings.

LARGEST FLATTENED CELLS

This section lists non-hcells having the greatest internal instance counts prior to flattening. The cells may be non-hcells either because they were in no hcell list (including -automatch) or they were hcells subject to expansion. Up to 20 cells are listed for each design. Cells in this list might make good hcells.

The layout and source cells are listed separately by name along with the numbers of instances contained in these cells. The contained instance counts are broken out into two classes: “Cell” which refers to cell placements, and “Primitive” which refers to primitive device instances.

If there are no flattened cells, then this section of the report is omitted.

UNBALANCED HCELLS EXPANDED

This section is included only when the WITH EXPANDED keyword is specified in the LVS Hcell Report statement, and then only when there is at least one expanded hcell that meets the threshold for inclusion. The data are formatted as a table with the following column headings:

Layout			Source		
Within cell	Expanded cell	Placements expanded	Within cell	Expanded cell	Placements expanded
-----	-----	-----	-----	-----	-----

Each row in the table represents the expansion of one hcell within another. The columns labeled “Within cell” contain the layout and source names for the containing hcell, while those labeled “Expanded cell” contain the names for the hcell that was expanded. The “Placements expanded” columns provide the number of placements of the cell that were expanded into the containing cell in the layout and source circuits. One row is generated for each pair of hcells where both of the following are true:

The expanded cell was expanded into the containing cell due to LVS Expand Unbalanced Cells processing.

The expanded cell's overall placement count in either the layout or the source circuit (or both) is greater than or equal to the threshold given in the WITH EXPANDED specification. The overall placement count here means the sum of the placement counts of the cell across all other hcells after flattening non-hcells.

In scenarios involving one-to-many correspondences, multiple rows may be written to provide all the data relating to the expansion of one correspondence within another.

POTENTIAL HCELL ADDITIONS

This section lists any hcells that potentially could make good hcells, thus providing additional memory and runtime savings. When present, it identifies certain layout or source non-hcells based on various metrics.

In each case, a source cell name or a layout cell name is provided. Since Calibre cannot generally determine how cells in the layout and source should correspond (no such information exists in GDS, OASIS, or SPICE), the user must determine how to match such names in hcell specifications. No distinction is drawn between cells that did not match any hcell specification and those that were rejected for some reason, they are both non-hcells.

Two different metrics identify cells that it might benefit performance to make into hcells: HIC and internal instance count.

The Hierarchical Instance Count (HIC) of a layout or source design is the sum of instance counts across all that design's hcells after expanding all non-hcells. Reductions in the HIC tend to correspond to reduced LVS memory usage and run times. There are two related subsections in the LVS report, one for the layout circuit and one for the source circuit. In each case, potential hcells are chosen by their calculated HIC reduction potential, which is expressed as a percentage rounded to the nearest integer. For each such cell, the cell name and percentage are listed in the appropriate subsection.

The actual HIC reduction that occurs upon making one of these cells an hcell may be different from the calculated potential due to factors like unbalanced hcell expansion, which the simple calculations done for this report do not account for. The Query Server's hcell analyzer accounts for such factors, however. Also note that the HIC reduction potential for each listed cell is calculated assuming that the given cell is added as an hcell and all else remains constant. Therefore, though several cells may be listed, once one of them is converted to an hcell, the calculations for the others are no longer valid. The hcell analyzer solves this by updating the calculations after each hcell is chosen, and before identifying the next candidate. For these reasons, the hcell analyzer is the preferred tool for finding potential hcell additions. A generic message to that effect appears in this subsection of the report.

Up to three source cells and three layout cells may be listed. Each is a non-hcell with a potential to improve the HIC in its circuit by at least five percent. Large cells are defined as those whose internal instance count is at least one percent of the total flat instance count of the circuit. Such cells are often good choices for hcells. Subsections are provided to list the large cells in the source and layout designs. Each such subsection provides a list of cell names with instance counts.

All empty subsections are omitted. For example, if there are no large cells in the source design, there will be no subsection for them. If there are no cells to list in any of the subsections, the entire section is omitted.

Parameters

The entries in the following table are organized by the CORRESPONDENCE DETAILS column headers as they appear from left to right and reading from top to bottom. Header names that appear more than once in a given report format are not repeated.

Table 13-1. Correspondence Table Column Header Definitions

Column	Description
Layout	Lists cells from the layout design.
High-cost hcell	If there is an adverse performance impact for maintaining a cell as an hcell during circuit extraction, or the cell is subject to layer operations that degrade hierarchy, that information appears in this column. The former is related to the LVS Auto Expand Hcells specification statement. This column does not apply in an exclusively SPICE-to-SPICE comparison run.
Source	Lists cells from the source design that correspond to those in the Layout column in the aggregate hcell list.
Hcell specification	Lists hcell names as they were specified and the originating source. Appears only in the default form of the report.
Rejected	Indicates whether an hcell pair was rejected and, if so, why. Appears only in the BY HCELL form of the report.
Expanded	Indicates whether an hcell (for correspondences that were actually established) was expanded due to LVS Expand Unbalanced Cells YES or LVS Flatten Inside Cell statements. If so, the instance count and reason are given.
Cell	Lists the names of cells as returned from the SPICE parser. These names are case-sensitive if the rule file specifies such handling. Names receive a “>” prefix if the cell has an internal instance count representing at least 1% of the overall flat instance count of the circuit containing the cell (source or layout).
Ports	Lists initial cell port counts.
Match type	Indicates how the cell was matched. Types include the following: direct — exact match (case-insensitive by default). wildcard — matched by a wildcard pattern. clone — a layout cell is a clone cell whose original name matches a specified hcell name (not applicable to Source names). auto — matched by the -automatch command line option heuristic (case-insensitive by default). This category always applies to top-level cells.
Optimization	Indicates the layout optimization applied to calculate a high cost hcell during circuit extraction. The optimization is an acronym code. The codes are defined in the report. These codes are not generally useful to an end user except possibly for regression purposes. What is important is the cost associated with an optimization, as cost can be used to control hcell expansion.

Table 13-1. Correspondence Table Column Header Definitions (cont.)

Column	Description
Cost	Indicates the cost a layout cell incurs if it is maintained as an hcell. This is calculated only for high-cost layout cells. The value is a percentage represented as an integer used in the LVS Auto Expand Hcells specification statement.
Layout name	Lists layout cell names as they appear in a user specification. Blank if matched by -automatch heuristic. Appears only in default report form.
Source name	Lists source cell names as they appear in a user specification. Blank if matched by -automatch heuristic. Appears only in default report form.
From	Lists the originating source of an hcell pair specification. The value is either a filename or “SVRF”. SVRF indicates a rule file Hcell statement (or equivalent in TVF). Appears only in default report form. This field is blank if the cell was matched with -auto, and the Match type field will show “auto”.
Y/N	Indicates whether an hcell pair was rejected. N means no; Y means yes. Appears only in BY HCELL report form.
Reason	Indicates the reason why an hcell pair was rejected using a code. Appears only in BY HCELL report form. Codes are described in Table 13-2 .
Instances	Lists the total number of expanded instances in layout and source when an expansion occurs due to LVS Expand Unbalanced Cells YES or LVS Flatten Inside Cell statements. This applies only to valid correspondences. If the correspondence was rejected, or if no instances of either cell were expanded, this field is left blank.
Reason	Indicates the reason why a valid hcell pair was expanded. The reason is given as an acronym code related to rule file statements. The codes are these: LEUC — due to LVS Expand Unbalanced Cells YES. LFIC — due to LVS Flatten Inside Cell. BOTH — due to both specification statements.

The following table lists subsections that can appear in the default report format. The BY HCELL report code is a corresponding Reason code that appears in the Rejected column of that report.

Table 13-2. BY REJECTION Report CORRESPONDENCES Subsections and Related BY HCELL Reason Codes

Subsection Title in Default Report
Description
BY HCELL Reason Code
CORRESPONDENCES ESTABLISHED
Lists valid hcell correspondences. This subsection is always present in the default report.
BY HCELL Reason Code: not applicable
CORRESPONDENCES EXPANDED DUE TO AMBIGUOUS HIGH SHORTING
Lists cells rejected as hcells because their pins are ambiguously high-shorted. See LVS Expand Ambiguously High Shorted Hcells .
BY HCELL Reason code: AHS
CORRESPONDENCES FULLY EXPANDED DUE TO LVS FLATTEN INSIDE CELL
Lists cells rejected as hcells due to LVS Flatten Inside Cell statements. Hcells that are expanded in some instances but not others due to LVS Flatten Inside Cell are listed as established.
BY HCELL Reason code: FEXP LFIC
CORRESPONDENCES FULLY EXPANDED DUE TO UNBALANCED HCELL EXPANSION
Lists cells rejected due to LVS Expand Unbalanced Cells YES (the default). Hcells that are expanded in some instances but not others due to LVS Expand Unbalanced Cells are listed as established.
BY HCELL Reason code: FEXP LEUC
CORRESPONDENCES FULLY EXPANDED DUE TO LVS FLATTEN INSIDE CELL AND UNBALANCED HCELL EXPANSION
Lists cells rejected as hcells for both of the preceding two reasons listed in this table. Hcells that are expanded in some instances but not others due to the two specification statements are listed as established.
BY HCELL Reason code: FEXP BOTH
CORRESPONDENCES IGNORED DUE TO HIERARCHICAL CONFLICTS
Lists cells rejected as hcells because using them would create a cyclical reference in the merged hierarchy. This phenomenon is discussed under “ Hcell Correspondence ” on page 475.
BY HCELL Reason code: HIERCONF
CORRESPONDENCES REJECTED BECAUSE THEY WOULD HAVE RESULTED IN MANY-TO-MANY RELATIONSHIPS
Lists cells rejected as hcells because they form a many-to-many match. This phenomenon is discussed under “ Hcell Correspondence ” on page 475.
BY HCELL Reason code: MANYMANY

Table 13-2. BY REJECTION Report CORRESPONDENCES Subsections and Related BY HCELL Reason Codes (cont.)

Subsection Title in Default Report	Description
BY HCELL Reason Code	
CORRESPONDENCES REJECTED DUE TO AUTOEXPANSION	<p>Lists cells rejected as hcells because they were subject to layer operations that degrade hierarchy.</p> <p>BY HCELL Reason code: AUTO</p>
CORRESPONDENCES REJECTED DUE TO DEFERRED STRICTNESS CONSTRAINTS	<p>Lists cells rejected as hcells due to mismatch in port counts. This can occur when the -genhcells command line option is used. See hcells::automatch in the <i>Calibre Query Server Manual</i> for details.</p> <p>BY HCELL Reason code: STRICT</p>
CORRESPONDENCES REJECTED DUE TO INCOMPATIBLE KINDS	<p>Lists cells rejected as hcells because one of the cells is a primary cell but the other is not.</p> <p>BY HCELL Reason code: DIFFKIND</p>
CORRESPONDENCES REJECTED DUE TO SEED PROMOTION	<p>Lists cells rejected as hcells because seed promotion has occurred in a layout cell. See “NO TITLE.”</p> <p>BY HCELL Reason code: SEEDPROM</p>

Examples

See the example under [LVS Hcell Report](#).

Hierarchy Preservation by Layer Operations

Some hierarchical layer operations preserve the original layout hierarchy (although not necessarily completely). Others can cause hierarchical degradation. The use of layer operations that degrade hierarchy should be avoided in the derivation of device seed layers.

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, pin, or auxiliary shapes in device recognition, or are used to derive such 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 across the boundary of an hcell (as specified with the -automatch or -hcell command-line options), then that cell may no longer match its schematic description. This results in discrepancies in Calibre nmLVS-H.

Normally, the tool can recover from device promotion due to hierarchy degradation in pin layers or auxiliary layers. This is done with the [LVS Push Devices](#) YES specification statement.

Device promotion caused by pin or auxiliary layers can significantly increase circuit extraction run time and raise memory consumption, but the device recognition module can normally recover from this and report those devices at the expected level of hierarchy. Therefore, circuit comparison can normally succeed in such cases.

However, LVS Push Devices YES cannot recover from hierarchy degradation that occurs in the derivation of device seed layers. (A seed layer is the first layer mentioned in a [Device](#) operation). Therefore, it is crucial to avoid hierarchy degradation in the derivation of device seed layers.

For example, suppose you have this Device operation:

```
DEVICE my_dev seed layer1(pin1) layer1(pin2) <aux>
```

Suppose you have this layer derivation:

```
seed = layer2 TOUCH aux
```

When the aux layer is present at a higher level than an hcell (but only above certain placements of the cell), then the seed shape cannot be formed in the cell. It can only be formed at the higher level where the aux layer exists. As a result, the device cannot be pushed down. This can result in LVS comparison discrepancies.

[Table 13-3](#) shows commonly-used layer operations in LVS and their effects on hierarchy.

Table 13-3. Hierarchical Degradation Effects in Layer Operations

Hierarchical Degradation Effect	Layer Operations	Comments
These tend to preserve hierarchy	AND two-layer (the LVSCAREFUL keyword increases hierarchy preservation) Area Copy Cut DFM Copy DFM Stamp Enclose Extent Inside Interact Net Net Area Net Area Ratio NOT OR Outside Size without OVERLAP ONLY Stamp Touch With Text	Some operations have “Not” counterparts (e.g., Area and Not Area). The effect of the negated operation is the same as the non-negated one.

Table 13-3. Hierarchical Degradation Effects in Layer Operations (cont.)

Hierarchical Degradation Effect	Layer Operations	Comments
These tend to degrade hierarchy	AND (single-layer) Angle Coincident Edge Coincident Inside Edge Coincident Outside Edge Density DFM Property Donut Extents Grow Holes Inside Edge Length Magnify Merge Outside Edge Path Length Perimeter Shift Size with OVERLAP ONLY Touch Edge Touch Inside Edge Touch Outside Edge Vertex With Edge XOR	<p>Some operations have “Not” counterparts (e.g., Angle and Not Angle). The effect of the negated operation is the same as the non-negated one.</p> <p>Layer operations that perform measurements tend to degrade hierarchy (Area is an exception). Edge-directed operations also tend to degrade hierarchy.</p>
	DFM Expand Edge Enclosure Expand Edge External Internal	<p>Promotion increases with increasing expansion or measurement constraints.</p> <p>If it makes sense to use OPPOSITE metric, this can reduce promotion.</p>
	Flatten, Rotate, XOR (single layer)	Flattens the input layer.

The [Layout Base Layer](#) specification statement assists in hierarchy optimization. This statement is highly recommended for all hierarchical applications. Care should be taken in specifying base layers correctly. Calibre nmLVS-H imposes no design restrictions concerning geometry overlapping cell placements or overlaps of cell placements.

The LVS Use Careful AND YES statement causes the LVSCAREFUL keyword to be applied globally to AND layer operations. This setting can be useful to avoid device layer shape promotion.

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 nmLVS-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 nmLVS-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 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 nmLVS-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 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 nmLVS-H as well as to certain other hierarchical Calibre applications, including Calibre xRC (but they do not apply to Calibre nmDRC-H).

Hierarchical LVS Comparison

Calibre nmLVS-H uses hierarchically corresponding cells (hcells) in the layout and source to manage the hierarchical structure during the run. Corresponding hcells are compared and discrepancies are reported at the cell level.

You specify hcells in an hcell list with the -hcell command line option or by using the [Hcell](#) specification statement, or both. The -genhcells or -automatch command line options may also be used (of the two, -genhcells is preferable). Calibre nmLVS-H uses this information to do cell-by-cell comparison during the run.

Hierarchical Pins	496
Trivial Pin Swappability	496

Hierarchical Pins

Pins of hierarchically corresponding cells (hcells) do not have to be texted, but it is recommended to do so. 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 can cause specific matching of pins, nets, and instances within cells at lower levels of hierarchy. Pin texting often improves run time in the circuit comparison module, and it improves ease of interpreting discrepancies.

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 an [LVS Recognize Gates](#) statement to enable logic gate recognition, and it is recommended that you use [LVS Power Name](#) and [LVS Ground Name](#) statements to allow formation of complete gates.

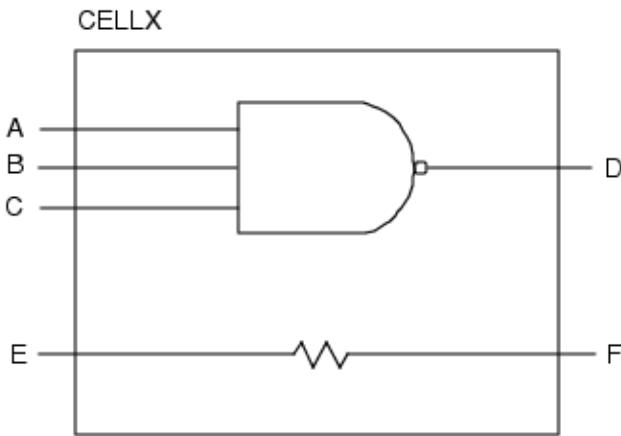
Trivial pin swappability is not carried up from logic gate pins where the properties or subtypes of respective transistors within the logic gate are not symmetric. Only traced properties are considered. Property values must be exactly equal—equality within tolerance values is not sufficient. Usually, 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 13-1](#).

Figure 13-1. Trivial Pin Swappability



In the figure, pins A, B, and C are swappable because they are connected directly to a NAND gate’s input and nothing else. Pins E and F are swappable because they are connected to a resistor.

LVS Box Cells

An LVS box cell is one that is verified only to the level of its ports (or pins, when placed as an instance). The internal structures of a box cell are not verified.

Box cells are enabled using the [LVS Box](#) specification statement. Without the BLACK or GRAY keywords, interactions between geometries inside box cells and outside of them are taken into account. That is, regular box cells are still sensitive to placement context; hence, things like seed promotion out of a box cell can still occur. With the BLACK and GRAY keywords, placement interactions are not taken into account, unless [LVS Box Peek Layer](#) is specified.

The BLACK keyword causes the geometries inside cells to be completely ignored. The GRAY keyword processes box cells in isolation from any placement context, and it allows the

geometries within the box cells to be passed on to downstream tools for applications like parasitic extraction.

See “[LVS Box Statement Usage](#)” in the *Calibre Solutions for Physical Verification* manual for additional details about LVS box flows.

Box Cells in Circuit Extraction

During circuit extraction, all placements of layout LVS Box cells are written as subcircuit calls to the extracted layout netlist. Layout box cells behave much like hcells during circuit extraction.

LVS Box cells are never pushed down into other cells. (Cell pushdown can occur in some Calibre tools for very small cells during the hierarchical database construction phase. LVS Box cells do not participate in this optimization.)

The contents of LVS Box cells are written to the extracted netlist by default when the BLACK and GRAY options are not used. This behavior can be changed through the [LVS Netlist Box Contents NO](#) or [LVS Preserve Box Cells YES](#) statements in the rule file. The latter statement also suppresses seed promotion out of box cells and the netlisting of any devices that originate from within box cells. (Device extraction does not occur when BLACK is specified, or when GRAY is specified without the DEVICES option.)

Unnamed nets (or pins) within LVS Box cells are written to the extracted netlist by default (not when LVS Box BLACK is specified). Such nets might be promoted to ports if they are connected to devices within the box cell and it is reasonable to treat them as ports. You can disable this netlisting behavior through the [LVS Netlist Unnamed Box Pins NO](#) statement in the rule file. Named pins (those that are connected to texted nets) always appear in the extracted netlist, even if they are on nets that have no devices.

Box Cells in Netlist Comparison

During LVS comparison, the contents of the LVS Box cells are ignored when the layout or source netlist is read. These cells are treated as primitive devices with pins. Connections are verified to the pin level and no farther.

The LVS component type of a box cell is the cell name. The cell has no subtype, but subtypes may be specified with the \$[mname] or \$.MODEL = mname constructs in subcircuit calls in SPICE netlists, just like for empty subcircuits. LVS pin names are the cell pin names.

High-Short Resolution

High-short resolution is a process used in Calibre nmLVS-H when processing connectivity information. It is applied to increase performance and capacity and to allow correlation between layout and source designs.

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 circuit extraction and during hierarchical circuit comparison, with some minor differences between the two as described next.

- **Circuit Extraction** — A high-shorted group of pins is a group of pins in a cell that are consistently connected together farther 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 user-given 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 user-given names as well as unnamed pins, 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 user-given name only and other unnamed pins, then all pins in the group are merged together.
 - If a high-shorted group of pins has only unnamed pins, then all pins are merged together.

This processing is identical in all cells, including LVS Box cells (where the pins may be ports). [Table 13-4](#) shows some possibilities of how high-shorted pins are resolved in the circuit extraction module.

Table 13-4. High-Shorted Pin Resolution Examples

Original high-shorted pins	Pins after high-short resolution	Notes
A B	A B	Different names remain separate even if unnamed pins are present.
A B 1	A B 1	
A A A B B	A B	Pins are merged by name.
A A A B B 3 4 5	A B 3	Pins are merged by name, and unnamed pins are merged.
A 1	A	Only one name, therefore all pins are merged.
A A 1 2 3	A	
1 1 2 2	1	Unnamed pins are merged.

Note that it is possible for pins (and respective nets) in a cell to remain separate within the cell even though they are globally connected and 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.

High-short resolution merges cell pins connected by `Sconnect` operations in all instances of a cell. For example, consider the pin configurations in the following table:

Original high-shorted pins	Cell pins stamped due to <code>Sconnect</code> (--) in all cell instances	Pins after high-short resolution
A B 3 4	none	A B 3 4
A B 3 4	A --> 3	A B 4
A B 3 4	A --> 3 B --> 4	A B

The original cell pins A, B, 3, and 4 are all high-shorted. If `Sconnect` is not the cause of the high short, then all these pins remain unmerged. But if `Sconnect` statements cause pins to be stamped in all instances of the cell, then those pins are merged by high-short resolution in accordance with the respective `Sconnect` connections.

- **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 these cases hold:
 - 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 (this differs from the circuit

extraction behavior). These names can be used later, for example, to establish initial correspondence points. The same is true for high-shorted nets. Here is an example:

Original high-shorted pins	Comparison in cell	Comparison in LVS Box Cell
A A A B B 3 4 5	A/B (Multiple pin names are stored for comparison to the other design.)	Name chosen according to rules described in the following paragraphs.

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. Circuit comparison *does not* resolve high shorts in built-in devices such as C, M, R, and so forth, 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 (does not apply when the BLACK keyword is used).

The handling of high-short resolution of LVS Box cell ports is controlled through the **LVS Preserve Box Ports** statement.

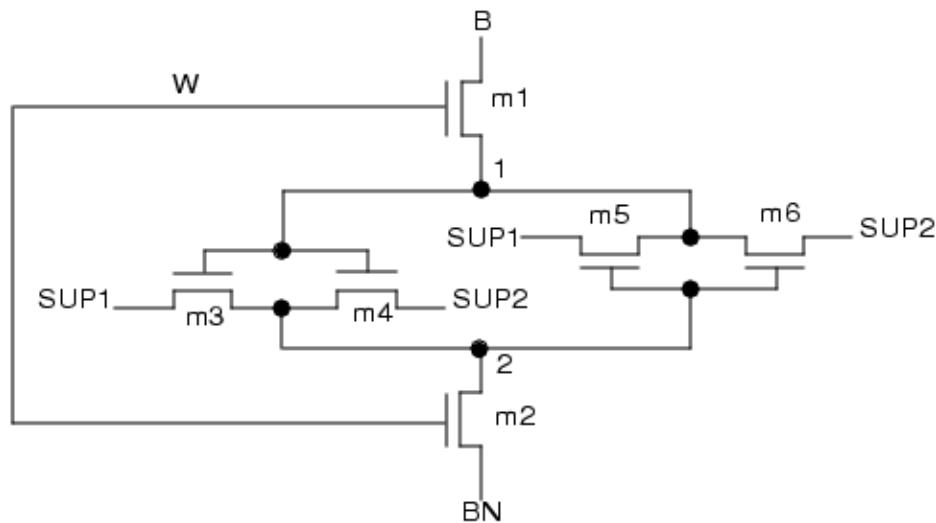
In certain situations (particularly in designs using “colored” metal layers), hcells having a many-to-one correspondence can have high-shorted pins that cause false LVS discrepancies. Due to connection ambiguities in such configurations, the best way to resolve the issue is to allow the affected hcells to be expanded. Specifying **LVS Expand Ambiguously High Shorted Hcells YES** in the rules accomplishes this.

SRAM Bit-Cell Recognition

Calibre nmLVS-H recognizes the common SRAM bit-cell structure. As a result, hierarchical LVS allows you to swap certain hcell pins in memory designs.

In [Figure 13-2](#), the names POWER and GROUND were replaced with SUP1 and SUP2, respectively.

Figure 13-2. 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.

Calibre nmLVS-H checks component types, subtypes, properties, and substrate pin connections to ensure that the structure is symmetric. Specifically, component types, subtypes, properties and substrate pin connections must be equal, respectively, on the following:

- transistors designated m1 and m2
- transistors designated m3 and m5
- transistors designated m4 and m6

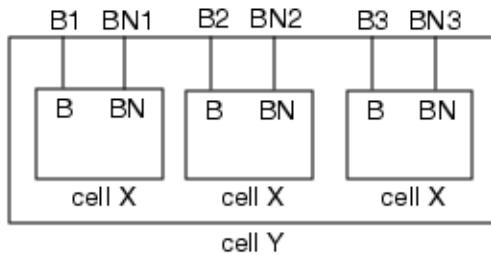
If LDD-type devices are used, then polarity is checked and must observe the symmetry of the structure. Only properties traced with [Trace Property](#) specification statements are checked. Property values must be exactly equal and equality within tolerance values is not sufficient. Note that, usually, only the source is required to have equal values, as described in the next paragraph. Substrate pins are any pins other than D, G, or S.

Given this structure, Calibre nmLVS-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. 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, Calibre nmLVS-H carries swappability information as far as possible up the hierarchy. Specifically, swappability information is carried up along nets that are connected only to a respective placement pin and to a external port of the cell, and are not connected to any other objects in the cell. In [Figure 13-3](#), given that pins B and BN of cell X are swappable, then in cell Y pin B1 is swappable with BN1, B2 is swappable with BN2, and B3 is swappable with BN3.

Figure 13-3. Carrying Pin Swappability Up the Hierarchy



Parameterized Cells

By default, the SPICE netlist reader in Calibre nmLVS-H flattens all parameterized subcircuits. A subcircuit is parameterized if it has at least one subcircuit call that references it and that specifies parameter values. The subcircuit is parameterized regardless of whether those parameters are actually used within the subcircuit.

In the following example, both subcircuits AAA and BBB are parameterized due to the X1 and X2 calls:

```
.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 $$ subcircuit calls passing parameters
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.

Note

 MOS devices having more than four pins do not participate in logic injection.

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 “[Injected Components and the LVS Report](#)” on page 507).

If cross-reference information is requested, corresponding original devices are written out. This can be done using either the `-ixf` command line option or the [Mask SVDB Directory IXF](#) keyword.

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 Calibre nmLVS-H first creates 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 they prevent injection if they appear on internal nets of injected circuits. This makes logic injection behave similarly to gate recognition with respect to preservation of user-given names.

Automatic detection of swappable hcell pins is enabled as appropriate.

The best-case candidates to benefit from logic injection 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).

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× savings (by injecting SRAM bit structures).

Note

 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.

Logic injection operates only in hierarchical LVS. Logic injection is controlled by the [LVS Inject Logic](#) specification statement. This statement may appear at most once. The default value is YES.

When logic injection is enabled, the transcript reflects this. 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 kinds were injected.

Injected Components and the LVS Report	507
Injected Component Identification	507
Injected Component Instance Pin Identification	508
Missing Injected Instance Discrepancy	508
Unmatched Injected Instance	509
Logic Injection and Gate Recognition	509
Internal Net Matching	510
Logic Injection and Pin Swappability	510
Injected Component Naming Conventions	511
Bit Structure Core	512
Bit Structure	513
Bit Rows	515
Inverters	517
Inverter Chains	518
Series Gates	519
Parallel Gates	521
NAND and NOR Gates	523
Multiplexer Structure	525
XOR and XNOR Gates	527
Transmission Gate Multiplexers	532

Register File Bit	533
-------------------------	-----

Injected Components and the LVS Report

From the user's perspective, components created by logic injection behave largely like logic gates. Injected components appear 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 (_).

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 have different injected component names. Detailed descriptions of each type of injected logic contain names and descriptions of all such configurations appear in the [Logic Injection](#) section.

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.

Related Topics

[Logic Injection](#)

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.

Related Topics

[Logic Injection](#)

Logic Injection and Gate Recognition

Logic injection and logic gate recognition are unrelated transformations. They can be enabled and disabled separately, and serve different purposes.

[Logic Gate Recognition](#) allows for swappability of signals and structures that are logically equivalent, but topologically different. Logic gate recognition does not save any memory. Logic injection, however, saves memory but does not allow for any swappability that would not exist otherwise.

Logic injection interacts with logic 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 are not 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 are swappable. If logic injection is enabled but gate recognition is disabled, pins are not 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.

Related Topics

[Logic Injection](#)

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, Calibre nmLVS-H 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, Calibre nmLVS-H 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.

If logic gate tolerance checking is enabled (using [LVS Recognize Gates](#) WITHIN TOLERANCE option), and an injection structure violates the tolerance in any pertinent [LVS Recognize Gates Tolerance](#) statement, then the input pins of the injected circuit are not swappable. Note that circuits with topological swappability, such as SRAM bit cells or parallel device groups, are not affected by this tolerance check since their pins are always swappable.

Detailed descriptions of every type of injected logic appear in the [Logic Injection](#) section and 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.

Injected Component Naming Conventions

In injected component descriptions that follow, 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.

Injected component names follow this general naming convention:

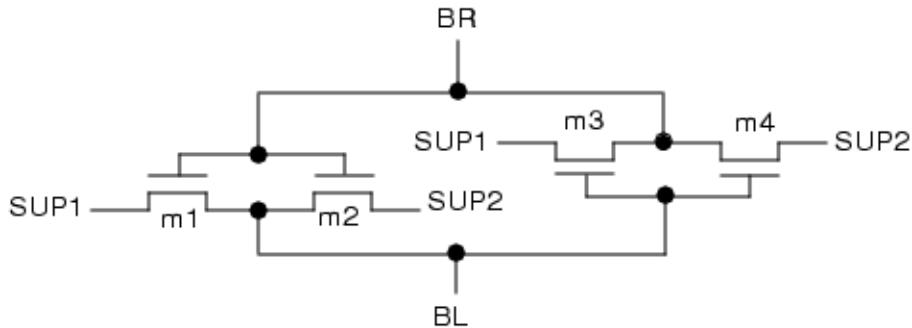
_xxxv — Voltage. Injected component of type xxx consisting of three-pin or four-pin MOS devices with substrate pins connected to the main supply nets in the component.

_xxxb — Bulk. Injected component of type xxx consisting of four-pin MOS devices with substrate pins not connected to the main supply nets of the component.

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 13-4. _bitcorev Structure



Requirements for successful logic injection:

- In Figure 13-4, 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 — Four-pin bit core (BL, BR, SUP1, SUP2), three-pin or four-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 13-4](#).)

_bitcoreb — Six-pin bit core (BL, BR, SUP1, SUP2, SUB1, SUB2), four-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 $2.5\times$. For comparison, if the bit core could be made into an hcell, memory consumption is reduced by $2.8\times$.

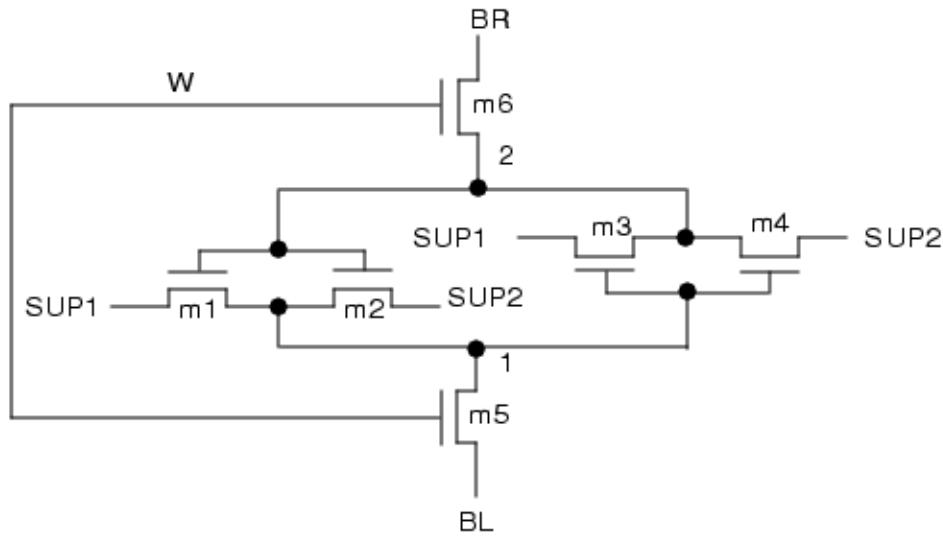
Related Topics

[Logic Injection](#)

Bit Structure

A standard SRAM bit structure is recognized and injected.

Figure 13-5. _bitv Structure



Requirements for successful logic injection:

- In [Figure 13-5](#), 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 — Five-pin bit structure (BL, BR, W, SUP1, SUP2), three-pin or four-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 13-5](#).)

_bitb — Seven-pin bit structure (BL, BR, W, SUP1, SUP2, SUB1, SUB2), four-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 $6\times$.

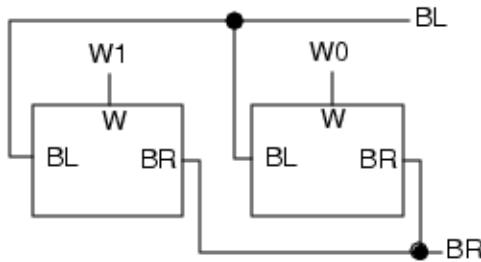
Related Topics

[Logic Injection](#)

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 13-6. _bitrow2v Structure



Requirements for successful logic injection:

- In Figure 13-6, 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 are 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 examines 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), three-pin MOS devices or four-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), four-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×, depending on the length of the row.

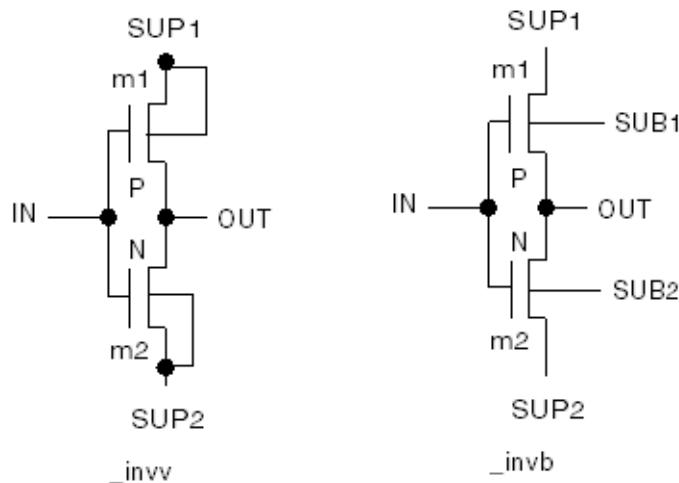
Related Topics

[Logic Injection](#)

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 13-7. _inv Structures



Requirements for successful logic injection:

- In the figure, the names IN, OUT, SUP1, SUP2, SUB1, and SUB2 are for reference only; no text is needed within the cell itself. These nets may be connected to other devices not shown in the diagram.
- 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 — Four-pin inverter (IN, OUT, SUP1, SUP2), three-pin or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_invb — Six-pin inverter (IN, OUT, SUP1, SUP2, SUB1, SUB2), three-pin or four-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×. For comparison, if the inverter structure could be made into an hcell, memory consumption is reduced by 1.6×.

Related Topics

[Logic Injection](#)

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 13-8. _invx3v Structure



Requirements for successful logic injection:

- In Figure 13-8, the names IN and OUT are for reference only; no text is required in the cells. 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 — Four-pin chain of N inverters (IN, OUT, SUP1, SUP2), three-pin or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_invxNb — Six-pin chain of N inverters (IN, OUT, SUP1, SUP2, SUB1, SUB2), four-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× for chains of 32 inverters.

Related Topics

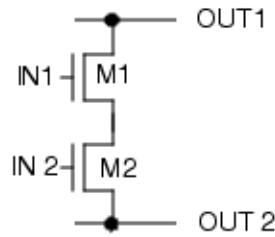
[Logic Injection](#)

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 [Figure 13-9](#), there is a 2-input series gate with no substrate pins.

Figure 13-9. _smn2v, _smp2v, _sup2v, _sdw2v Gates



Requirements for successful logic injection:

- In Figure 13-9, the names IN1, IN2, OUT1, and OUT2 are for reference only; no text is required in the cell. 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 IN_n, IN₂ with IN_{n-1}, and so forth.
- If gate recognition is enabled, all input pins IN₁ ... IN_n 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:

_smpnv — Series gate with n input pins IN₁ ... IN_n, and two output pins OUT1 and OUT2 ($n+2$ pins altogether) formed from three-pin and four-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.

_smpnb — Series gate with n input pins $\text{IN}_1 \dots \text{IN}_n$, two output pins OUT1 and OUT2 , and a substrate pin SUB ($n+3$ pins altogether) formed from four-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 $1.6\times$ for 2-transistor series gates, and $2.1\times$ for 4-transistor series gates. For comparison, if the series gates could be made into hcells, memory consumption would be reduced by $1.7\times$ for 2-transistor series gates, and $2.2\times$ for 4-transistor series gates.

Related Topics

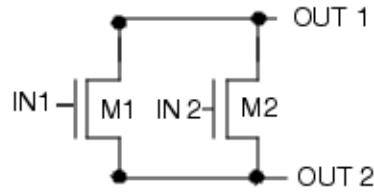
[Logic Injection](#)

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 [Figure 13-10](#), there is a 2-input parallel gate with no substrate pins.

Figure 13-10. `_pmn2v`, `_pmp2v`, `_pup2v`, `_pdw2v` Gates



Requirements for successful logic injection:

- In Figure 13-10, the names IN1, IN2, OUT1, and OUT2 are for reference only; no text is required in the cell. 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.
- Calibre nmLVS-H 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, Calibre nmLVS-H may decide to avoid injecting those gates in that hcell.

Pin swappability:

- All input pins IN1 ... IN_n 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 ... IN _{n} , and two output pins OUT1 and OUT2 ($n+2$ pins altogether) formed from three-pin and four-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 ... IN_n, two output pins OUT1 and OUT2, and a substrate pin SUB ($n+3$ pins altogether) formed from four-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 two-transistor parallel gates, and 1.7 \times for four-transistor parallel gates. For comparison, if the parallel gates could be made into hcells, memory consumption would be reduced by 1.5 \times for two-transistor parallel gates, and 1.9 \times for four-transistor parallel gates.

Related Topics

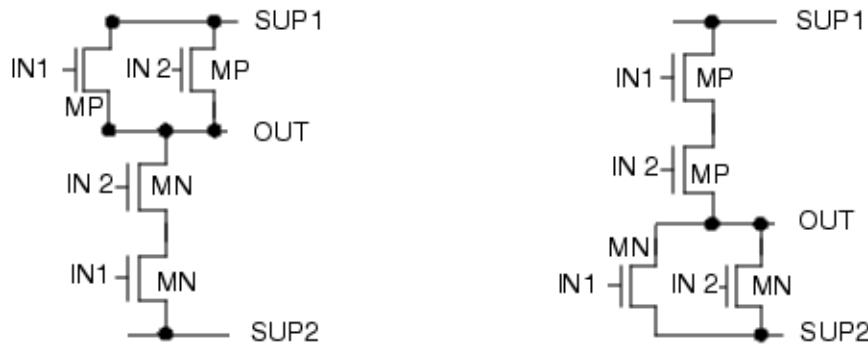
[Logic Injection](#)

NAND and NOR 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 an arbitrary number of input pins are recognized and injected. In [Figure 13-11](#), there is a two-input NAND gate and a two-input NOR gate, both with no substrate pins.

Figure 13-11. _nand2v and _nor2v Gates



Requirements for successful logic injection:

- In Figure 13-11, the names IN1, IN2, OUT, SUP1, and SUP2 are for reference only; no text is required in the cell. 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.
- 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 $IN_1 \dots IN_n$ 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 $\text{IN}_1 \dots \text{IN}_n$, and pins OUT, SUP1, and SUP2 ($n+3$ pins altogether) formed from three-pin and four-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 $\text{IN}_1 \dots \text{IN}_n$, and pins OUT, SUP1, SUP2, B1, and B2 ($n+5$ pins altogether) formed from four-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.

_nornv — NOR gate with n input pins $\text{IN}_1 \dots \text{IN}_n$, and pins OUT, SUP1, and SUP2 ($n+3$ pins altogether) formed from three-pin and four-pin MOS devices with substrate pins of P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_nornb — NOR gate with n input pins $\text{IN}_1 \dots \text{IN}_n$, and pins OUT, SUP1, SUP2, B1, and B2 ($n+5$ pins altogether) formed from four-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.

Related Topics

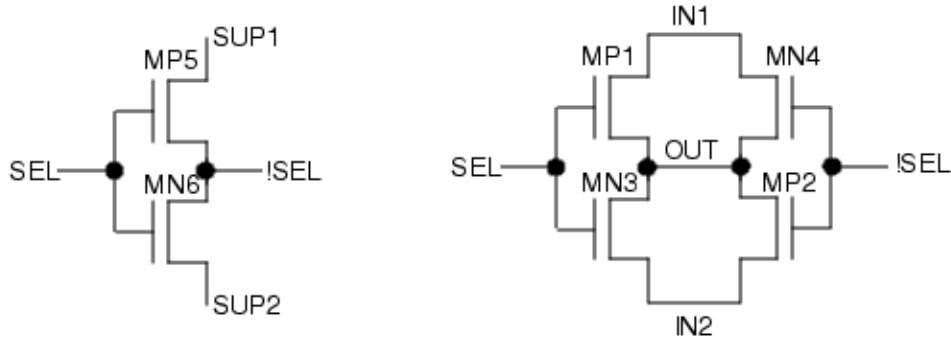
[Logic Injection](#)

Multiplexer Structure

Multiplexers based on a transmission gate multiplexer (TGM) structure are recognized and injected.

Figure 13-12 shows the multiplexer structure:

Figure 13-12. _mx2v Gate Multiplexer



All nets labeled 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 13-12, the names IN1, IN2, SEL, !SEL, SUP1, and SUP2 are used for reference only; no text is required in the cell. 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.

Possible configurations:

_mx2v — Six-pin multiplexer structure (IN1, IN2, SEL, OUT, SUP1, SUP2), three-pin MOS devices or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_mx2b — Eight-pin multiplexer structure (IN1, IN2, SEL, OUT, SUP1, SUP2, SUB1, SUB2), four-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 $4.9\times$.

Related Topics

[Logic Injection](#)

XOR and XNOR Gates

XOR gates based on a TGM structure are recognized and injected.

[Figure 13-13](#) and [Figure 13-14](#) show two XOR gates based on a multiplexer mx2 structure:

Figure 13-13. _xra2v Multiplexer

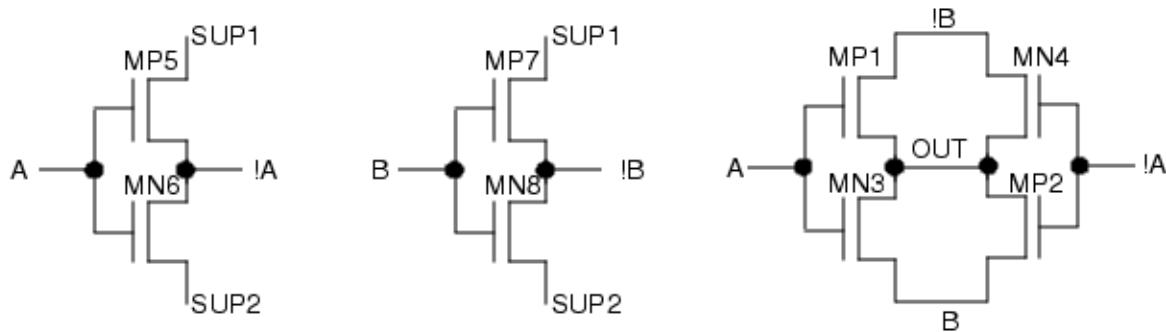
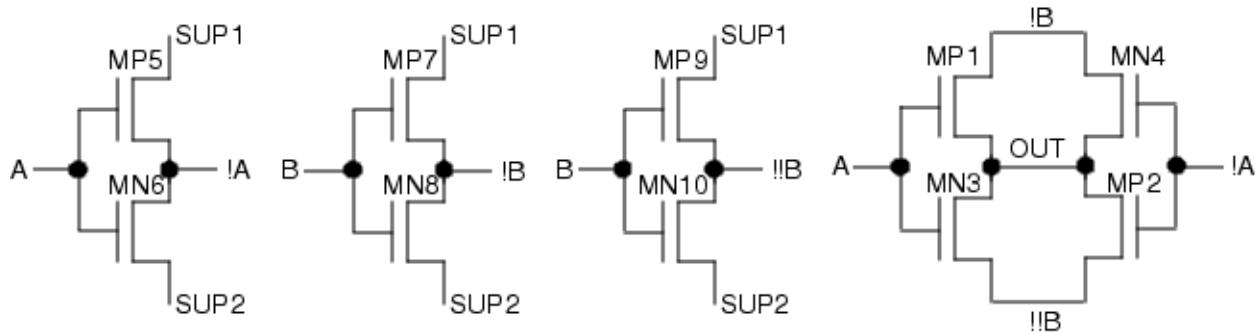


Figure 13-14. _xr2v Multiplexer



All nets labeled 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 13-13](#) and [Figure 13-14](#), the names A, B, OUT, SUP1, and SUP2 are used for reference only. No text is required in the cell. 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 does not examine devices inside the injected gate.

Possible configurations:

_xr2v — Five-pin XOR gate (A, B, OUT, SUP1, SUP2), three-pin MOS devices or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xr2b — Seven-pin XOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), four-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 — Five-pin XOR gate (A, B, OUT, SUP1, SUP2), three-pin MOS devices or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xra2b — Seven-pin XOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), four-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

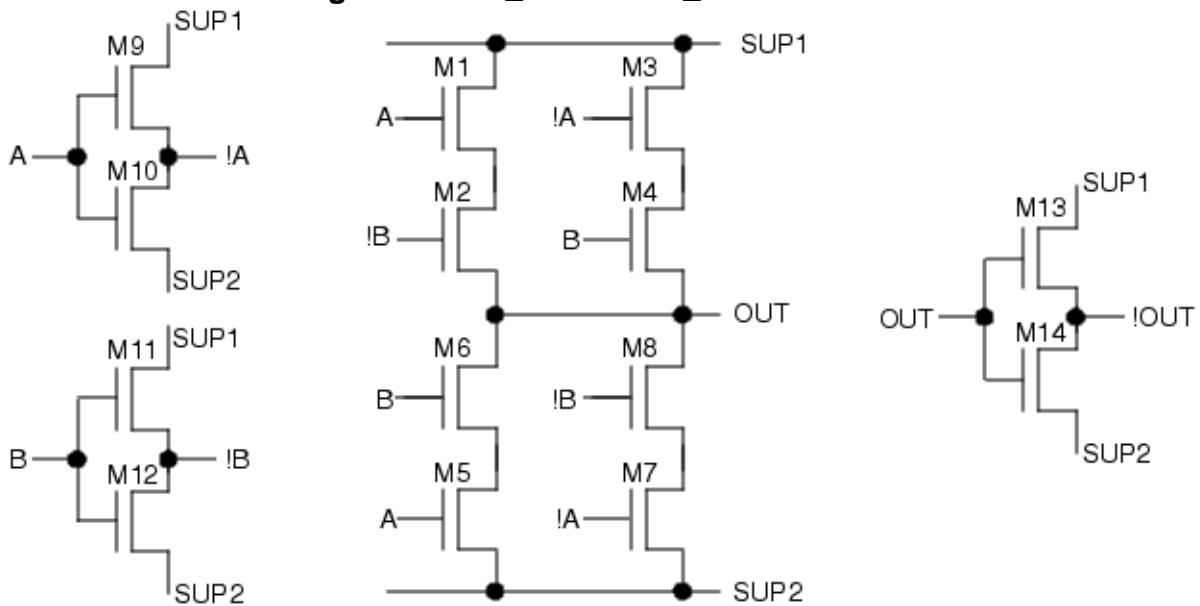
- SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports.
- Substrate connections are checked but discrepancies are reported at individual transistor level.

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 \times or 9 \times , respectively.

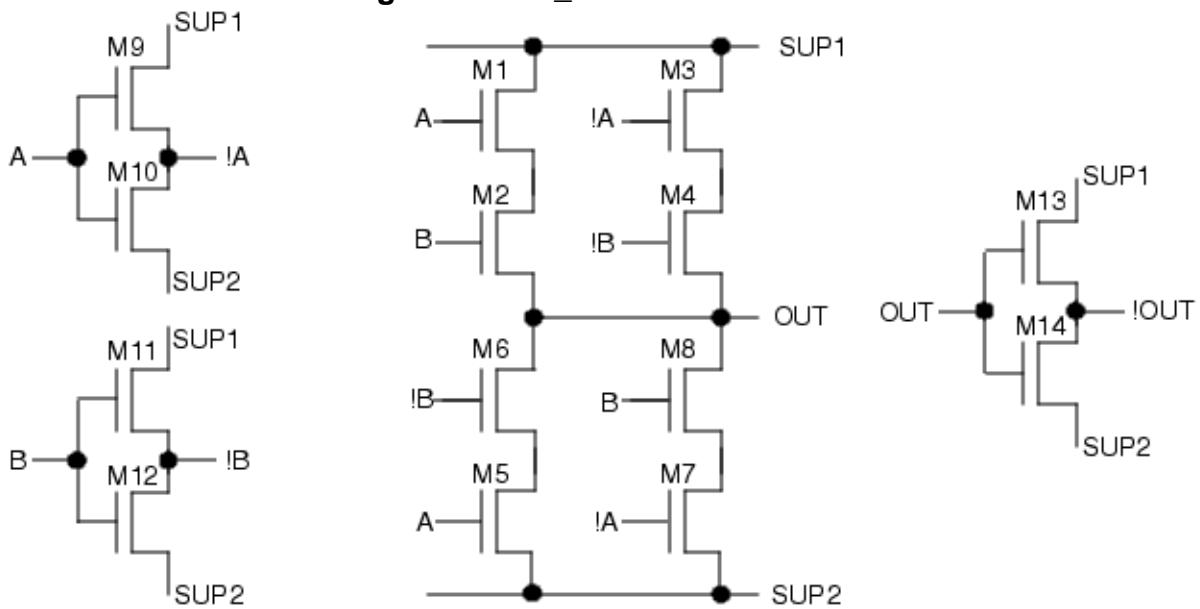
XOR and XNOR gates based on a series-parallel structure are recognized and injected. Figure 13-15 shows an XOR gate, xor2v. With a trailing inverter connected to the output net, this structure is recognized instead as a XNOR gate, xnori2v.

Figure 13-15. _xor2v and _xnori2 Gate



An alternate configuration of inputs results in a different XNOR gate which does not require a trailing inverter. Figure 13-16 shows an XNOR gate, xnor2v. With a trailing inverter connected to the output net, this structure is recognized as an XOR gate, _xori2v.

Figure 13-16. _xnor2v and xor12



All nets labeled 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 13-16, the names A, B, OUT, SUP1, and SUP2 are used for reference only. No text is required in the cell. 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, M6, M8, M13, and M14 must be CMOS P-type transistors. Devices M5, M7, M10, M12, and M14 must be CMOS N-type transistors.
- 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 examines types, subtypes, and properties of devices inside the injected gate.

Possible configurations:

_xor2v — Five-pin XOR gate (A, B, OUT, SUP1, SUP2), three-pin MOS devices or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xor2b — Seven-pin XOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), four-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.

_xori2v — Five-pin XOR gate (A, B, OUT, SUP1, SUP2), three-pin MOS devices or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xori2b — Seven-pin XOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), four-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 — Five-pin XNOR gate (A, B, OUT, SUP1, SUP2), three-pin MOS devices or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xnor2b — Seven-pin XNOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), four-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 — Five-pin XNOR gate (A, B, OUT, SUP1, SUP2), three-pin MOS devices or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xnor2b — Seven-pin XNOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), four-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 **_xnor2** structures. For comparison, if the XOR gate could be made into an hcell, memory consumption would be reduced by $12.6\times$ or $14.7\times$, respectively.

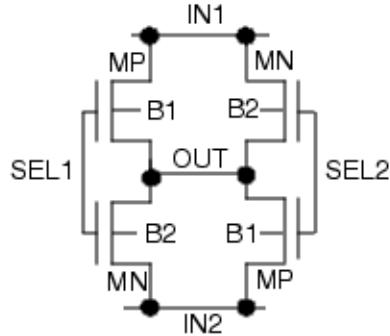
Related Topics

[Logic Injection](#)

Transmission Gate Multiplexers

A 4-MOS Transmission Gate Multiplexer (TGM) is recognized and injected.

Figure 13-17. _tgmb Gate Multiplexer



Requirements for successful logic injection:

- In Figure 13-17, the names SEL1, SEL2, IN1, IN2, OUT, B1, and B2 are for reference only; no text is required in the cell. 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 — Seven-pin transmission gate multiplexer (IN1, IN2, SEL1, SEL2, OUT, B1, and B2), formed from three-pin and four-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×. For comparison, if the TGM logic could be made into an hcell, memory consumption would be reduced by 2.0×.

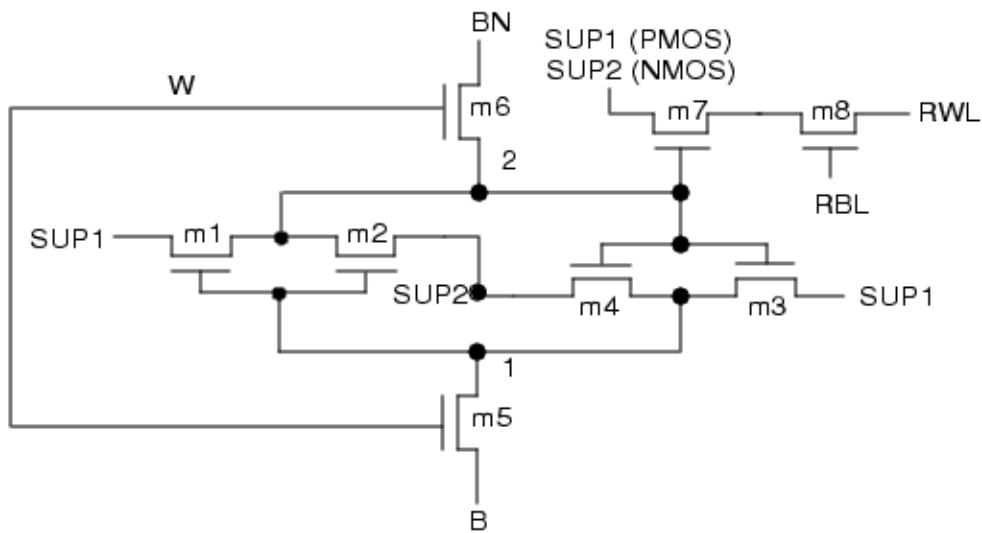
Related Topics

[Logic Injection](#)

Register File Bit

The Register File Bit (RFB) structure is recognized and injected.

Figure 13-18. `_bitrv` Structure



Requirements for successful logic injection:

- In Figure 13-18, the names `B`, `BN`, `W`, `RWL`, `RBL`, `SUP1`, and `SUP2` are for reference only; no text is required in the cell. 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 one 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.
- 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 — Seven-pin bit structure (B, BN, W, RWL, RBL, SUP1, SUP2), three-pin MOS devices or four-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_bitrb — Nine-pin bit structure (B, BN, W, RWL, RBL, SUP1, SUP2, SUB1, SUB2), four-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×. For comparison, if the bit structure could be made into an hcell, memory consumption would be reduced by 6.3×.

Related Topics

[Logic Injection](#)

Chapter 14

LVS Results

Calibre nmLVS and Calibre nmLVS generate an LVS Report along with various other results files such as a circuit extraction report and any requested results databases. The run transcript also has useful information for interpreting results.

LVS Transcript	538
Circuit Extraction Report.....	545
ERC, Short Isolation, and Softchk Results	546
LVS Summary Report.....	547
LVS Report.....	549
SPICE Syntax Check Report	619
SVDB Cross-Reference Files.....	623
Binary Polygon Format (BPF) Database.....	629
Mask Results Database	631

LVS Transcript

The LVS transcript is similar in many respects to those written by all Calibre batch tools.

The section “[Session Transcripts](#)” on page 206 describes the Calibre nmDRC transcripts in detail. Much of what is described there, with the exception of the DRC Executive portion, also applies to LVS runs when a physical layout database is used.

One statistic that appears in LVS transcripts that does not appear in DRC is MALLOC (memory allocation). It appears in lines such as this:

```
--- CALIBRE::LVS/xRC INITIALIZATION MODULE COMPLETED. CPU TIME = 0
REAL TIME = 1 LVHEAP = 65/85/85 MALLOC = 55/55/55 ELAPSED TIME = 32
```

The MALLOC numbers (megabytes) are used/current/maximum, with definitions as follows:

used — Size of used blocks but not free blocks (Linux resident set or VmRSS).

current — Size of used blocks and free blocks (Linux VmRSS plus the size of swap or VmSwap). This number can be larger than used when pin swapping occurs.

maximum — Maximum reported value of current.

LVHEAP is a subset of MALLOC maximum memory. (For very small designs, the reported values may appear to show the converse of this.)

The LVS run transcript has four major sections. The sections that appear depend upon which options you have specified on the command line. The major sections are these:

Rule file compilation — See “[Rule File Compilation](#)” on page 206 for details.

Calibre initialization module — For circuit extraction the hierarchical circuit extractor initialization module setup is shown. For flat LVS comparison, the LVS/Calibre xRC initialization module setup is shown.

Layout data input — This appears when circuit extraction is run. This section is similar to that discussed under “[Layout Data Input](#)” on page 208.

Executive processes — These include processes such as circuit extraction, short isolation, ERC checking, device recognition, and LVS comparison.

Hierarchical Circuit Extraction Transcript	539
Circuit Comparison Transcript	543

Hierarchical Circuit Extraction Transcript

The hierarchical circuit extraction run transcript has certain features unique to it that you should be aware of.

The section “[Transcript Features Based Upon Run Mode](#)” on page 221 shows various differences that appear during the hierarchical database construction phase that depend on hierarchical modes of operation such as MT, MTflex, and hyperscaling.

During the hierarchical database construction phase, hcell performance cost is reported in the section entitled HIGH-COST HCELLS. The estimated performance impact of an hcell is measured and rated from 0 to 100, with 100 representing the highest cost. The following example shows how such a report might appear:

```
HIGH-COST HCELLS
  blk1 (EXPANDING DENSE OVERLAPS) : 73.8
  route2 (EXPANDING DENSE OVERLAPS) : 70.8
  blk2 (EXPANDING DENSE OVERLAPS) : 51.2
  seg (EXPANDING DENSE OVERLAPS) : 25.7
```

The report shows the names of the hcells, the hierarchy-management optimization for which the cost is being estimated, and the cost score. You can manage automatic expansion of such hcells through the [LVS Auto Expand Hcell](#) specification statement.

After the initial three sections listed under [LVS Transcript](#), the CALIBRE::HIERARCHICAL CIRCUIT EXTRACTOR - EXECUTIVE MODULE section appears. This section shows layer derivations. The statistics that appear in this section are described under “[Layout Data Input](#)” on page 208, “[Layer Statistics in Hierarchical Processing](#)” on page 215, and “[LVHEAP Statistics](#)” on page 217.

Next, the CONNECTIVITY EXTRACTION module section appears. Statistics are shown such as these:

```
CONNECTIVITY EXTRACTION
-----
NC = Net Count  IPC = Internal Pin Count
CEC = Cell Edge Count  PEC = Promoted Edge Count
LVHEAP = Geometry heap memory allocation.
MALLOC = Total heap memory allocation.
CELL nand
  nand (NC=18 IPC=0 CEC=150 PEC=0 CFGC=63)
  nand CPU TIME = 0  REAL TIME = 0  LVHEAP = 3/5/67  MALLOC = 80/80/80
  ELAPSED TIME = 2
```

If the rule file specifies [LVS Isolate Shorts](#) YES, you will see something like the following in the transcript:

```
HIERARCHICAL SHORT ISOLATION started.  
HIERARCHICAL SHORT ISOLATION in cell TOPCELL started.  
GLOBAL HEURISTICS: HCC=3 FCC=5(5) HGC=117 FGC=225  
SHORT 0: HCC=3 FCC=5(5) HGC=40 FGC=79 PWR vdd (2/2)/3[1/3]/1/1/1  
CPU TIME = 0 REAL TIME = 0 LVHEAP = 3/5/67 MALLOC = 80/80/80  
ELAPSED TIME = 2  
HIERARCHICAL SHORT ISOLATION in cell TOPCELL completed. CPU TIME = 0  
REAL TIME = 0 LVHEAP = 3/5/67 MALLOC = 80/80/80 ELAPSED TIME = 2
```

Any shorts are written to the short isolation database. They are reported later in the transcript after device recognition.

The beginning of device recognition is indicated by this line:

```
DEVICE RECOGNITION
```

After device recognition has occurred, any circuit extraction errors are listed, such as this:

```
Extraction Errors and Warnings for cell "TOPCELL"  
-----  
WARNING: Short circuit - Different names on one net:  
Net Id: 5  
(1) name "PWR" at location (-1.75,82.5) on layer 10 "metal2"  
(2) name "vdd" at location (7,77) on layer 10 "metal2"  
The name "PWR" was assigned to the net.
```

The extraction of the SPICE netlist is indicated by the following lines:

```
HIERARCHICAL SPICE NETLISTER  
-----  
HIERARCHICAL SPICE NETLISTER completed. CPU TIME = 0 REAL TIME = 0  
LVHEAP = 3/5/67 MALLOC = 80/80/80 ELAPSED TIME = 2
```

If the rule file specifies that ERC should occur (see “[Electrical Rule Checks](#)” on page 351), then you will see something like this in the transcript:

```
CALIBRE HIERARCHICAL ERC
-----
no_supply::<1> = PATHCHK !POWER && !GROUND
-----
ERC PATHCHK initialization.

ERC NET GRAPH initialization started.

ERC NET GRAPH initialization completed CPU TIME = 0  REAL TIME = 0
LVHEAP = 3/5/67  MALLOC = 80/80/80
no_supply::<1> (HIER TYP=1 CFG=1 HGC=3 FGC=3 HEC=38 FEC=38 VHC=F VPC=F)
CPU TIME = 0  REAL TIME = 0  LVHEAP = 2/5/67  OPS COMPLETE = 20 OF 20
ELAPSED TIME = 2
```

The end of the transcript appears as follows:

```
--- CALIBRE::HIERARCHICAL CIRCUIT EXTRACTOR COMPLETED - Thu Jan 17
13:51:43 2013
--- TOTAL CPU TIME = 0  REAL TIME = 1  LVHEAP = 2/5/67  MALLOC = 80/80/80
ELAPSED TIME = 2
--- PROCESSOR COUNT = 1

--- SPICE NETLIST FILE = TOP.ext.sp
--- CIRCUIT EXTRACTION REPORT FILE = lvs.report.compare.ext
--- SHORT ISOLATION RESULTS DATABASE = lvs.report.compare.shorts
--- PERSISTENT HIERARCHICAL DATABASE(PHDB) = svdb/TOPCELL.phdb
--- QUERY DATABASE = svdb  TOP CELL = TOPCELL
--- TOTAL RULECHECKS EXECUTED = 1
--- TOTAL RESULTS GENERATED = 3 (3)
--- ERC RESULTS DATABASE FILE = erc.db (ASCII)
--- ERC SUMMARY REPORT FILE = erc.report.hier
```

This section shows the runtime statistics, the [Circuit Extraction Report](#), and the filename of the extracted SPICE netlist. It also shows the name of the [Mask SVDB Directory](#) (if specified), the location of the short isolation database (if specified), and the ERC results database and summary report (if specified). See “[ERC, Short Isolation, and Softchk Results](#)” on page 546 for additional information.

If the -hyper option is used, then statistics for the HDBs along with aggregate values are reported like this:

```
--- CALIBRE::HIERARCHICAL CIRCUIT EXTRACTOR COMPLETED - Thu Mar  4
13:37:27 2021
--- TOTAL CPU TIME = 8  REAL TIME = 8  LVHEAP = 18/50/67  SHARED = 0/32
MALLOC = 91/91/91  ELAPSED TIME = 9
--- HDB 0      : CPU TIME = 6      LVHEAP = 67      MALLOC = 91
--- HDB 1      : CPU TIME = 1      LVHEAP = 8       RX = 0      TX = 1
--- HDB 2      : CPU TIME = 1      LVHEAP = 16      RX = 0      TX = 1
--- HDB 3      : CPU TIME = 1      LVHEAP = 8       RX = 0      TX = 1
--- HDB 4      : CPU TIME = 1      LVHEAP = 8       RX = 0      TX = 1
--- EXTRACTION TOTAL HDB 0-4    CPU TIME = 9      REAL TIME = 8  LVHEAP = 107
.
.
.
--- GRAND TOTAL HDB 0-4 TOTAL  CPU TIME = 9      REAL TIME = 10   LVHEAP = 107
```

Circuit Comparison Transcript

The LVS circuit comparison transcript has certain features you should be aware of.

After the initial three sections listed under [LVS Transcript](#), the CALIBRE::LVS/xRC - EXECUTIVE MODULE section appears. This section shows various algorithms that are called for netlist comparison such as device reduction, logic gate recognition, logic injection, and so forth. The cumulative amounts of time spent for these algorithms are reported near the end of the transcript.

The end of the transcript appears as follows:

```
LVS completed. CORRECT. See report file: lvs.report

LVS completed. CPU TIME = 0 REAL TIME = 0 LVHEAP = 68/69/69 MALLOC =
46/46/46 ELAPSED TIME = 1

--- LVS REPORT FILE = lvs.report
--- CALIBRE::LVS COMPARISON MODULE COMPLETED. TOTAL CPU TIME = 0 REAL
TIME = 0 LVHEAP = 3/5/69 MALLOC = 46/46/46 ELAPSED TIME = 1

--- CALIBRE::LVS/xRC COMPLETED - Thu Mar 4 13:28:11 2021
--- XDB CROSS REFERENCE DATABASE = svdb/TOPCELL.xdb
```

This shows the overall comparison result, the name of the LVS Report file, the LVS Summary Report file (if specified), various runtime statistics, and the name of the Mask SVDB Directory cross-reference database, if requested. See “[Hierarchical Instance and Net Cross-Reference Files](#)” on page 627 for additional information.

Hierarchical Comparison with Hyperscaling Transcript 543

Hierarchical Comparison with Hyperscaling Transcript

When circuit extraction is run together with LVS comparison and hyperscaling is enabled (-hyper cmp command line option), then the complete transcript of the comparison portion of the run is written to a file called *lvs_report_name.log*, where *lvs_report_name* is taken from the LVS Report specification statement.

Here is an example of reporting of key performance statistics from an MT comparison run (following circuit extraction) with hyperscaling:

```
--- CALIBRE::HIERARCHICAL CIRCUIT EXTRACTOR COMPLETED - Thu Mar  4
15:06:24 2021
--- TOTAL CPU TIME = 7  REAL TIME = 3  LVHEAP = 26/74/76  SHARED = 0/32
MALLOC = 160/160/160  ELAPSED TIME = 4
--- HDB 0      : CPU TIME = 5      LVHEAP = 76      MALLOC = 160
--- HDB 1      : CPU TIME = 1      LVHEAP = 15      RX = 0      TX = 1
--- HDB 2      : CPU TIME = 0      LVHEAP = 10      RX = 0      TX = 1
--- HDB 3      : CPU TIME = 1      LVHEAP = 25      RX = 0      TX = 1
--- HDB 4      : CPU TIME = 1      LVHEAP = 10      RX = 0      TX = 1
--- EXTRACTION TOTAL HDB 0-4    CPU TIME = 8      REAL TIME = 3  LVHEAP = 136
.
.
.
--- CALIBRE::LVS COMPARISON MODULE COMPLETED. TOTAL CPU TIME = 1  REAL
TIME = 2  LVHEAP = 29/150/150  SHARED = 0/32  MALLOC = 243/243
/243  ELAPSED TIME = 5

--- CALIBRE::LVS/xRC COMPLETED - Thu Mar  4 15:06:25 2021

--- SPICE NETLIST FILE = layout.sp
--- CIRCUIT EXTRACTION REPORT FILE = lvs.report.ext

--- HDB 0/LVS COMPARE : CPU TIME = 6      LVHEAP = 150      MALLOC = 244
--- HDB 1            : CPU TIME = 1      LVHEAP = 15      RX = 0      TX = 1
--- HDB 2            : CPU TIME = 0      LVHEAP = 10      RX = 0      TX = 1
--- HDB 3            : CPU TIME = 1      LVHEAP = 25      RX = 0      TX = 1
--- HDB 4            : CPU TIME = 1      LVHEAP = 10      RX = 0      TX = 1
--- GRAND TOTAL HDB 0-4 TOTAL CPU TIME = 9  REAL TIME = 6  LVHEAP = 150
```

The EXTRACTION TOTAL line shows the subtotals from the circuit extraction portion of the run. The HDB 0/LVS COMPARE line shows the subtotals from the circuit comparison portion of the run. In an MTflex run, the LVS COMPARE information appears on its own line, like this:

```
LVS COMPARE : CPU TIME = 6      LVHEAP = 146      MALLOC = 238
```

See “[Time Statistics](#)” on page 213 and “[LVHEAP Statistics](#)” on page 217 for related information.

The transcript and the -hyper cmp log (if any) contain relevant error messages if any of the following statements are specified and circuit extraction causes comparison to abort:

```
LVS ABORT ON ERC ERROR YES
LVS ABORT ON SOFTCHK YES
LVS ABORT ON SUPPLY ERROR YES
LVS ABORT ON SUPPLY SHORT YES
```

Circuit Extraction Report

The circuit extraction report is written during the circuit extraction phase. This report contains a summary of circuit extraction warnings and errors that appear in the transcript or in the extracted layout netlist.

Items included in the report are these:

- Circuit extraction errors and warnings, such as short circuits, open circuits, and 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 SPICE netlister.

Various messages are discussed under “[Open and Short Circuits](#)” on page 265, “[Unattached Ports](#)” on page 285, “[Bad Device Reporting](#)” on page 320, and “[Circuit Extraction Messages](#)” in the *SVRF Manual*.

The report is written to a file named *lvs_report_name.ext*, where *lvs_report_name* 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.

Here is an example:

```
#####
##                               ##
##          C A L I B R E      S Y S T E M      ##
##                               ##
##          C I R C U I T      E X T R A C T I O N      R E P O R T      ##
##                               ##
#####
REPORT FILE NAME:           lvs_report.ext
LAYOUT NAME:                ./inv_nand.oas ('TOPCELL')
CREATION TIME:              Thu Jan 17 14:52:35 2013
CURRENT DIRECTORY:          /user/me/design
USER NAME:                  me
CALIBRE VERSION:            v2013.1_4      Fri Jan 11 10:16:10 PST 2013

Extraction Errors and Warnings for cell "TOPCELL"
-----
WARNING: Short circuit - Different names on one net:
Net Id: 3
(1) name "PWR" at location (-1.75,82.5) on layer 10 "metal2"
(2) name "VCC" at location (45,75) on layer 10 "metal2"
The name "PWR" was assigned to the net.
```

ERC, Short Isolation, and Softchk Results

ERC, short isolation, and soft-connection checking are performed during circuit extraction when any of them are enabled.

When [Electrical Rule Checks](#), [Short Isolation](#), or [Detection of Soft Connections](#) is requested, the application generates results in ASCII results database format.

For ERC, the results are sent to the database specified in the [ERC Results Database](#) specification statement. The ERC results database is readable by Calibre RVE for DRC. This format is discussed under “[ASCII DRC Results Database Format](#)” on page 236. An ERC summary report is written to the file specified in the [ERC Summary Report](#) specification statement.

Short isolation occurs when you specify [LVS Isolate Shorts](#) YES in your rule file. The results are written to a file called *lvs_report_name.shorts* if an LVS Report name is specified. Otherwise it is written to a file called *lvs.layout_cell_name.rep.shorts*. This file is readable by Calibre RVE for LVS and is similar to the DRC ASCII results format. Information regarding shorts is also written to the [Circuit Extraction Report](#).

Soft connection checking occurs when you specify [LVS Softchk](#) or [LVS Report Option S](#). The results are written to the circuit extraction report, and if LVS Softchk is specified, to a results

database called *layout_primary.softchk*. See “[LVS Softchk Results Database File Format](#)” on page 293.

LVS Summary Report

If LVS Summary Report is specified for a hierarchical comparison run, a summary report is generated, which contains a brief synopsis of the run.

Here is an example:

```
=====
== CALIBRE::LVS-H SUMMARY REPORT
==

Extraction Summary
-----
Extraction Start Time: Thu Jan 17 14:25:42 2013
Extraction Finish Time: Thu Jan 17 14:25:45 2013
Circuit Extraction Report: erc.lvs.rep.ext
Circuit Extraction SPICE Netlist: layout.spi
Short Circuit Count: 1
Ambiguous Name Pad Count: 2
Open Circuit Count: 3
Unattached Name Pad Count: 4
Unattached Port Pad Count: 8
Virtual Connections Made: 5
SCONNECT Conflicts Detected: 6
GLOBAL Name Conflicts Detected: 7
Bad Devices Detected: 9
Box Cells Present: 10

Electrical Rule Checking (ERC)
-----
ERC Results Database: ercdb.rep
ERC Summary Report File: erc_summary.rep
TOTAL ERC RuleChecks Executed: 3
TOTAL ERC RuleCheck Results Generated: 12 (24)
TOTAL ERC PRINT NETS Results Generated: 8
TOTAL ERC PRINT POLYGONS Results Generated: 3
TOTAL DFM RDB Results Generated: 1
TOTAL ERC PATHCHK Operations Executed: 11
TOTAL ERC PATHCHK POLYGON Results Generated: 12 (24)
TOTAL ERC PATHCHK NET Results Generated: 14
--- WARNING: POWER, GROUND, or LABELED nets required by ERC operations
do not exist.
--- WARNING: ERC operation was unable to generate complete output because
of insufficient memory.
```

LVS Results

LVS Summary Report

```
ERC PATHCHK POLYGON Files:          erc.pathchk-1.rep  
                                      erc.pathchk-2.rep  
  
ERC PATHCHK NET Files:             erc.pathchk-1.nets.rep  
                                      erc.pathchk-2.nets.rep  
  
ERC DFM RDB Files:                dfmrdb-1.rep  
                                      dfmrdb-2.rep  
  
Comparison Summary  
-----  
Comparison Start Time:            Thu Jan 17 14:25:45 2013  
Comparison Finish Time:           Thu Jan 17 14:25:45 2013  
  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
!!!  
!!! Warning: Comparison was run before extraction! !!!  
!!!  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
  
LVS Comparison Report:            erc.lvs.rep  
LVS Comparison Status:           INCORRECT.  
  
#   #      ~   ~  
# #      x   x  
#          @  
# #      / \ \\  
#   #  
  
LVS Input Notes Detected:         2  
LVS Input Warnings Detected:     4  
LVS Input Errors Detected:       6  
Comparison Warning Types Detected: 10  
Comparison Error Types Detected:  3  
Box Cells Used during Comparison: 40
```

This report is not generated for a circuit extraction (calibre -spice) run alone. The circuit extraction run data shown in the report is taken from the [Mask SVDB Directory](#). The **QUERY** option must be specified in order for the [LVS Summary Report](#) to be generated.

The Query Server Tcl Shell can also generate this report, as well as pull specific elements of information that appear in the report from the SVDB database. See the [qs::create_lvs_summary_report](#) and [qs::get_run_summary_data](#) commands in the *Calibre Query Server Manual*.

LVS Report

The LVS Report contains the results of an LVS run in human-readable text form.

The report is specified using the [LVS Report](#) statement in your rule file, which is required for LVS. You can use this report, along with the layout, LVS results database, and SPICE netlists to locate discrepancies. Calibre RVE for LVS enables access to all of these items. (See “[Using Calibre RVE for LVS](#)” in the *Calibre RVE User’s Manual*)

Overall Report Structure	549
Analysis of the LVS Report.....	558
Errors and Warnings.....	560
Overall Comparison Results.....	562
LVS Report Listing Conventions	570
LVS Discrepancy Types	575
Absolute Trace Property Tolerances Out of Range	584
Component Types with Non-Identical Signal Pins.....	585
Detailed Instance Connections	587
Errors in Names Given for Power/Ground Nets.....	588
Hierarchical Cells Forming a Cycle.....	588
Incorrect Substrate Connections	589
Input Errors	590
Instances of Cells With Non-Floating Extra Pins	591
Name Errors.....	594
Property Errors	595
Unmatched Objects	596
Information and Warnings.....	599
Detailed Error Analysis Reporting	603

Overall Report Structure

The LVS Report contains information about the LVS run. The contents are similar for hierarchical and flat runs, but in flat runs all results are from the top-level cell perspective only.

The LVS Report contains the following items:

- LVS netlist compiler errors and warnings, if any were found.
- LVS header section, specifying the report filename, 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 these items:
 - Primary comparison status message.

For hierarchical runs, this status is CORRECT if all individual cells are correct and INCORRECT if at least one cell is incorrect or not compared. The overall status is NOT COMPARED if all cells have a status of 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 562 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 “[LVS Netlist Compiler](#)” for the meanings of individual status messages.
 - A Cell Summary, listing the primary comparison status for each individual cell (CORRECT, INCORRECT, or NOT COMPARED). The respective layout and source cell names are indicated. Cells that exist only in the layout or only in the source but contain input errors (for example, missing property errors) appear in the CELL SUMMARY section as well. In that case, the layout or source cell name is indicated with no corresponding cell name in the other design. This section may be omitted if global problems are found in the design.
 - Optional sections describing global problems found in the design, such as hierarchy cycles.
 - LVS Parameters section. This section shows the LVS program configuration.
- Cell-by-cell Comparison Results section (-hier runs). 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 objects for the cell, and so on.

You can specify [LVS Report Option NOK](#) to limit this section of the report to only cells that have discrepancies in them. By default, all cells have a detailed report. The status of all cells is always reported in the “Overall Comparison Results” section.

A hierarchical LVS report example is shown here:

```
#####
##          C A L I B R E      S Y S T E M      ##
##          L V S      R E P O R T      ##
##          #####
REPORT FILE NAME:           lvs.report.compare
LAYOUT NAME:                TOP.ext.sp ('TOPCELL')
SOURCE NAME:                ./src.spi ('TOPCELL')
RULE FILE:                 rules
CREATION TIME:              Thu Jan 17 14:52:45 2013
CURRENT DIRECTORY:          /user/me/lvs
USER NAME:                  me
CALIBRE VERSION:            v2013.1_4      Fri Jan 11 10:16:10 PST 2013
```

OVERALL COMPARISON RESULTS

```
#   #
# #
#   #     #####
#   #     #     #
#   #     #     INCORRECT    #
#   #     #     #
#   #     #####
#   #
```

Error: Cells with non-floating extra pins.
 Warning: Extra ports in source.
 Warning: Ambiguity points were found and resolved arbitrarily.

```
*****
***** CELL SUMMARY *****
*****
```

Result	Layout	Source
-----	-----	-----
CORRECT	inv	inv
CORRECT	nand	nand
INCORRECT	TOPCELL	TOPCELL

```
*****
LVS PARAMETERS
*****
```

- o LVS Setup:

```
// LVS COMPONENT TYPE PROPERTY
// LVS COMPONENT SUBTYPE PROPERTY
// LVS PIN NAME PROPERTY
LVS POWER NAME           "PWR"  "VDD"
LVS GROUND NAME          "GND"  "VSS"
LVS CELL SUPPLY          NO
LVS RECOGNIZE GATES      ALL
LVS IGNORE PORTS         NO
LVS CHECK PORT NAMES     NO
LVS IGNORE TRIVIAL NAMED PORTS NO
LVS BUILTIN DEVICE PIN SWAP YES
LVS ALL CAPACITOR PINS SWAPPABLE NO
LVS DISCARD PINS BY DEVICE NO
LVS SOFT SUBSTRATE PINS   NO
LVS INJECT LOGIC         NO
LVS EXPAND UNBALANCED CELLS YES
LVS FLATTEN INSIDE CELL  NO
LVS EXPAND SEED PROMOTIONS NO
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 ALLOW INLINE PARAMETERS
LVS SPICE ALLOW UNQUOTED STRINGS NO
LVS SPICE CONDITIONAL LDD  NO
LVS SPICE CULL PRIMITIVE SUBCIRCUITS NO
LVS SPICE IMPLIED MOS AREA NO
// LVS SPICE MULTIPLIER NAME
LVS SPICE OVERRIDE GLOBALS NO
LVS SPICE REDEFINE PARAM   NO
LVS SPICE REPLICATE DEVICES NO
LVS SPICE SCALE X PARAMETERS NO
LVS SPICE STRICT WL       NO
// LVS SPICE OPTION
LVS STRICT SUBTYPES        NO
LVS EXACT SUBTYPES         NO
LAYOUT CASE                NO
SOURCE CASE                NO
LVS COMPARE CASE           NO
LVS DOWNCASE DEVICE         NO
LVS REPORT MAXIMUM          50
LVS PROPERTY RESOLUTION MAXIMUM 32
// LVS SIGNATURE MAXIMUM
// LVS FILTER UNUSED OPTION
LVS REPORT OPTION           S
LVS REPORT UNITS            YES
// LVS NON USER NAME PORT
// LVS NON USER NAME NET
```

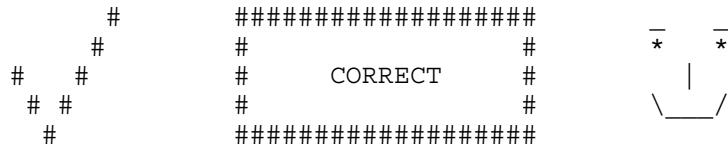
```
// LVS NON USER NAME INSTANCE

// Reduction

LVS REDUCE SERIES MOS           NO
LVS REDUCE PARALLEL MOS          YES
LVS REDUCE SEMI SERIES MOS       NO
LVS REDUCE SPLIT GATES           YES
LVS REDUCE PARALLEL BIPOURAL    YES
LVS REDUCE SERIES CAPACITORS    YES
LVS REDUCE PARALLEL CAPACITORS  YES
LVS REDUCE SERIES RESISTORS     YES
LVS REDUCE PARALLEL RESISTORS   YES
LVS REDUCE PARALLEL DIODES      YES
LVS REDUCTION PRIORITY          PARALLEL

LVS SHORT EQUIVALENT NODES      NO
```

CELL COMPARISON RESULTS



Warning: Extra ports in source.

LAYOUT CELL NAME: inv
SOURCE CELL NAME: inv

INITIAL NUMBERS OF OBJECTS

	Layout	Source	Component Type
	-----	-----	-----
Ports:	4	4	
Nets:	4	4	
Instances:	1	1	MN (4 pins)
	1	1	MP (4 pins)
Total Inst:	2	2	

NUMBERS OF OBJECTS AFTER TRANSFORMATION

	Layout	Source	Component Type
	-----	-----	-----

LVS Results

Overall Report Structure

Ports:	2	3	*
Nets:	4	4	
Instances:	1	1	INV (2 pins): output input
Total Inst:	1	1	

* = Number of objects in layout different from number in source.

INFORMATION AND WARNINGS

	Matched Layout	Matched Source	Unmatched Layout	Unmatched Source	Component Type
Ports:	2	2	0	1	
Nets:	4	4	0	0	
Instances:	1	1	0	0	INV
Total Inst:	1	1	0	0	

- o Statistics:

2 layout trivial ports were deleted.
1 source trivial port was deleted.

- o Extra Ports in Source:

3

CELL COMPARISON RESULTS

Warning: Extra ports in source.

Warning: Ambiguity points were found and resolved arbitrarily.

LAYOUT CELL NAME: nand
SOURCE CELL NAME: nand

INITIAL NUMBERS OF OBJECTS

	Layout	Source	Component Type
Ports:	5	5	
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:	2	3	*
Nets:	5	5	
Instances:	1	1	NAND2 (3 pins)
Total Inst:	1	1	

* = Number of objects in layout different from number in source.

INFORMATION AND WARNINGS

	Matched Layout	Matched Source	Unmatched Layout	Unmatched Source	Component Type
Ports:	2	2	0	1	
Nets:	5	5	0	0	
Instances:	1	1	0	0	NAND2
Total Inst:	1	1	0	0	

o Statistics:

3 layout trivial ports were deleted.
2 source trivial ports were deleted.

LVS Results

Overall Report Structure

1 net was matched arbitrarily.

o Extra Ports in Source:

4

o Ambiguity Resolution Points:

(Each one of the following objects belongs to a group of indistinguishable objects.

The listed objects were matched arbitrarily by the Ambiguity Resolution feature of LVS.

Arbitrary matching may be prevented by assigning names to these objects or to adjacent nets).

Layout	Source
-----	-----

Nets

5

5

CELL COMPARISON RESULTS (TOP LEVEL)

```
#   #      #####
# #      #
#      #      INCORRECT      #
# #      #
# #      #####
```

Error: Cells with non-floating extra pins.

LAYOUT CELL NAME: TOPCELL
SOURCE CELL NAME: TOPCELL

INITIAL NUMBERS OF OBJECTS

	Layout	Source	Component Type
Ports:	2	2	
Nets:	7	6	*
Instances:	1	1	inv (2 pins): 1 2
	1	1	nand (2 pins)

Total Inst: 2 2

NUMBERS OF OBJECTS AFTER TRANSFORMATION

	Layout	Source	Component Type
Ports:	2	2	
Nets:	2	2	
Instances:	1	1	inv (2 pins): 1 2
	1	1	nand (2 pins)
Total Inst:	2	2	

* = Number of objects in layout different from number in source.

INCORRECT OBJECTS

LEGEND:

ne = Naming Error (same layout name found in source circuit, but object was matched otherwise).

INSTANCES OF CELLS WITH NON-FLOATING EXTRA PINS

DISC#	LAYOUT NAME	SOURCE NAME
1	X1(24.500,0.000) inv ** missing pin ** X1/3	X1/X0 inv X1/X0/3:1
2	X0(-4.250,0.000) nand ** missing pin ** X0/4	X0/X0 nand X0/X0/4:1

INFORMATION AND WARNINGS

	Matched Layout	Matched Source	Unmatched Layout	Unmatched Source	Component Type
Ports:	2	2	0	0	

Nets:	2	2	0	0
Instances:	1	1	0	0
	1	1	0	0
Total Inst:	2	2	0	0

o Statistics:

1 source net had all its pins removed and was deleted.

o Initial Correspondence Points:

Ports: PWR GND

```
*****
***** SUMMARY *****
*****
Total CPU Time:      0 sec
Total Elapsed Time:  1 sec
```

Analysis of the LVS Report

The LVS Report is useful for understanding discrepancies in the design. Read it with the following things in mind.

- Read the error and warning messages at the beginning of the report.
 - The report first shows circuit 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 file, 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 rule 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, layout problem, or source netlist 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, indicating an open circuit.
 - 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, an instance may be missing, or any other type of discrepancy.

Often, data are redundant between the two sections because of interdependent causes for discrepancies.

- 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 nmLVS results can be viewed graphically with Calibre RVE when the [Mask SVDB Directory](#) specification statement is specified in the rule file. This statement writes an SVDB database to the pathname you specify. This database is used for LVS results analysis. For more information, refer to “[Using Calibre RVE for LVS](#)” in the *Calibre RVE User’s Manual*.

For information about the Query Server, which accesses SVDB databases, see the [Calibre Query Server Manual](#).

Errors and Warnings

The LVS Report can contain a number of different types of errors and warnings.

- Circuit Extraction

Circuit extraction information is in the [Circuit Extraction Report](#). You can cause a number of incorrect objects that appear in the circuit extraction report to appear in the LVS Report by using [LVS Report Option](#) EB, EC, EO, and ES.

- Stamp Operation

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, its syntax is checked. 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.

The SPICE netlist syntax-check-mode is controlled by the -cl and -cs command line options. See “[SPICE Syntax Check Report](#)” on page 619 for information about the report these options generate.

- LVS discrepancies

The principal categories of LVS discrepancies are listed under “[Secondary Messages](#)” on page 563.

- 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 these:

INCORRECT NETS
INCORRECT PORTS
INCORRECT INSTANCES
DETAILED INSTANCE CONNECTIONS
INSTANCES OF CELLS WITH NON-FLOATING EXTRA PINS
INCORRECT SUBSTRATE CONNECTIONS

Overall Comparison Results

The overall results of the LVS run are listed in the opening portion of the OVERALL COMPARISON RESULTS section of the LVS Report.

OVERALL COMPARISON RESULTS

```
#   #
# #
#           INCORRECT
# #
#   #
```

Error: Cells with non-floating extra pins.

Warning: Extra ports in source.

Warning: Ambiguity points were found and resolved arbitrarily.

***** CELL SUMMARY *****		
Result	Layout	Source
-----	-----	-----
CORRECT	inv	inv
CORRECT	nand	nand
INCORRECT	TOPCELL	TOPCELL

Calibre nmLVS-H also shows these results on a per-hcell basis. See “[Hcells](#)” on page 474 for additional information about hcells.

Primary Messages	562
Secondary Messages	563
Overall SPICE Syntax Check Results	570

Primary Messages

Primary messages in the LVS Report indicate the overall result for the design. These messages are also given for each hcell if the run is hierarchical.

```
#   #
# #
#
#   #   INCORRECT   #
# #
#   #

#####
```

Error: Cells with non-floating extra pins.

Warning: Extra ports in source.

Warning: Ambiguity points were found and resolved arbitrarily.

```
*****
CELL SUMMARY
*****
Result      Layout          Source
-----      -----          -----
CORRECT    inv            inv
CORRECT    nand           nand
INCORRECT  TOPCELL        TOPCELL
```

Table 14-1 lists the primary comparison status messages.

Table 14-1. Primary Messages

Message	Description
CORRECT	The layout connectivity is equivalent to the source connectivity.
CORRECT except for naming or swap-override errors	The layout connectivity is equivalent to the source connectivity, except for differences in element names or violations of restrictions on swapping of pins.
INCORRECT	Discrepancies were detected between the two circuits.
NOT COMPARED	LVS aborted prior to the comparison stage because of problems in the layout or source data. Every cell in the circuit must have this status for the overall result to be NOT COMPARED. The actual problems are listed further down in the report.

Secondary Messages

Secondary messages in the LVS Report, if any, follow the primary comparison status message. Secondary messages appear in the OVERALL COMPARISON RESULTS section, as well as in the CELL COMPARISON RESULTS sections of hierarchical runs.

```

#   #
# #
#
#   INCORRECT
# #
#   #####

```

Error: Components with non-identical signal pins.

Warning: Components with non-identical power or ground pins.

Table 14-2 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. Additional details related to secondary messages appear later in the report.

Error conditions are certainly wrong and must be fixed or the manufactured design will not function properly.

Table 14-2. Secondary Messages—Errors

Error	Description
Error: Cell hierarchy contains a cycle	Indicates that the cell hierarchy, consisting of the layout and source hierarchies as well as cell correspondence information, contained a cycle. This error appears only in Calibre nmLVS-H. It is a global error that causes Calibre nmLVS-H to abort without comparing individual cells. See “ Hierarchical Cells Forming a Cycle ” on page 588 for reporting details.
Error: Cells with non-floating extra pins.	A cell contains instances of other cells that have extra pins in the source or layout. The extra pins are caused by connections across the hierarchy that are present in one design but not in the other. In those instances, the extra pins are induced by connections to other design elements (they are not floating). These discrepancies can indicate a short circuit or be related to some other discrepancy. See “ Instances of Cells With Non-Floating Extra Pins ” on page 591 for reporting details.
Error: Components with non-identical signal pins were found.	The number of signal pins or the signal pin names of some layout components differ from the corresponding schematic components. See “ Component Types with Non-Identical Signal Pins ” on page 585 for reporting details.
Error: Connectivity errors.	Connectivity errors were found, for example, incorrect nets, incorrect instances, or unmatched elements.
Error: Different numbers of instances (see next).	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 later in the report.
Error: Different numbers of nets (see next).	The number of nets in the layout differ from the number of nets in the source. The actual numbers of nets appear later in the report.

Table 14-2. Secondary Messages—Errors (cont.)

Error	Description
Error: Different numbers of ports (see next).	The number of ports in the layout differ from the number of ports in the source. The actual numbers of ports appear later in the report.
Error: Duplicate cell name “cell” in cell lists “list1” and “list2” after wildcard substitution.	LVS Cell List is used with wildcard matching, which results in a conflict in the cell lists. The matched cells must be unique.
Error: Errors in layout.	LVS found errors related to the layout netlist. The “LAYOUT INPUT ERRORS” section lists the actual problems. See “ Input Errors ” on page 590.
Error: Errors in source.	LVS found errors related to the source netlist. The “SOURCE INPUT ERRORS” section lists the actual problems. See “ Input Errors ” on page 590.
Error: Incorrect names for power/ground nets.	This error causes LVS to abort prior to the comparison stage. See “ Errors in Names Given for Power/Ground Nets ” on page 588 for reporting details.
Error: Instances of different types or subtypes were matched.	Some layout instances were matched to some schematic instances which had different component types or subtypes. These instances are listed as discrepancies.
Error: Layout names missing in source.	This error appears when LVS Report Option N is used. In addition, a NAME ERRORS section appears in the report for instance, net, or port names, respectively. See “ Name Errors ” on page 594 for the report formats.
Error: Layout passthrough ports missing in source (see INFORMATION AND WARNINGS).	There are passthrough nets present in the layout and LVS Report Option LPE is specified. The passthrough nets are reported in the INFORMATION AND WARNINGS section. See “ Information and Warnings ” on page 599 for related details.
Error: Many-to-many correspondence in cell names.	Indicates that the cell correspondence specified to LVS-H in the hcell list leads to a many-to-many relationship. This is a global error that causes LVS to abort without comparing individual cells. This error appears only in Calibre nmLVS-H. Following is an example of an hcell list with many-to-many correspondence: aa aa aa bb bb bb See “ Hierarchical Cells Forming a Cycle ” on page 588 for reporting details.

Table 14-2. Secondary Messages—Errors (cont.)

Error	Description
Error: Pins of instances of specified models were swapped during comparison.	The LVS Report Pinswapped statement is specified and pin swapping occurred for instances having device model names in the statement. The “Information and Warnings” section of the report contains PINSWAPPED INSTANCE CONNECTIONS reports for cells. See “ Information and Warnings ” on page 599 for related details.
Error: Power and/or ground ports connected to opposite voltage nets and/or ports in layout. Power and/or ground ports connected to opposite voltage nets and/or ports in source.	Indicates a power or ground conflict in a cell due to propagation of port signals. These errors can occur when LVS Cell Supply YES is specified, and the propagation of power and ground signals from lower levels of the hierarchy cause a conflict. These messages are reported in any cell where such a conflict occurs.
Error: Power net missing in layout. Power net missing in source. Ground net missing in layout. Ground net missing in source.	Indicates that a power or ground net is missing in the layout or source. These errors cause LVS to abort prior to the comparison stage. These errors are issued only when they are relevant and only when the absence of a power 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 in the layout, and the source contains logic gates, then a “Power net missing” or “Ground net missing” error is issued for the layout in cases where such nets are missing. But if the source contains no logic gates, then these error messages may not be issued. These error messages may appear together on one line.
Error: Power or ground net missing.	Indicates that “Power net missing” or “Ground net missing” errors were issued for one or more cells in Calibre nmLVS-H. This error may appear in the “Overall Comparison Results” section in Calibre nmLVS-H, with more information provided in the report sections pertaining to the individual cells.
Error: Properties missing on instances in layout. Error: properties missing on instances in source.	Indicates that device properties are missing from instances in one of the compared designs. The missing properties are required for comparison. See the Properties Missing on Instances section under “ LVS Discrepancy Types ” on page 575 for reporting details.
Error: Property errors.	LVS found differences in values of source and layout properties. The “Property Errors” section lists these instances with their corresponding property values and error percentages. “ Property Errors ” on page 595.

Table 14-2. Secondary Messages—Errors (cont.)

Error	Description
Error: Property <name> was declared string in TRACE PROPERTY user program, but (layout) input contains numeric data.	LVS Property Initialize is specified, but the corresponding Trace Property statement specifies a string property.
Error: Property tolerances out of range of actual values.	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 “ Absolute Trace Property Tolerances Out of Range ” on page 584 for reporting details.
Error: 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, and LVS Report Option W is specified. For example, in Calibre nmLVS-H, if the -spice command line option is not used, then the error 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 error.
Error: Source passthrough ports missing in layout (see INFORMATION AND WARNINGS).	There are passthrough nets present in the source and LVS Report Option SPE is specified. The passthrough nets are reported in the INFORMATION AND WARNINGS section of the LVS Report. See “ Information and Warnings ” on page 599 for related details.
Error: Substrate pin errors.	LVS found instances with incorrect substrate connections. This error is reported only when LVS Soft Substrate Pins YES is indicated in the rule file. See “ Incorrect Substrate Connections ” on page 589 for reporting details.
BAD DEVICE(s) detected by extraction in this cell. See extraction report for details.	This error appears when LVS Report Option EB is specified. Circuit extraction detected bad devices in the cell. The error appears in the LAYOUT INPUT ERRORS section. See “ Input Errors ” on page 590 for related details.
Open circuit(s) detected by extraction in this cell. See extraction report for details.	This error appears when LVS Report Option EO is specified. Circuit extraction detected text shorts in the cell. The error appears in the LAYOUT INPUT ERRORS section. See “ Input Errors ” on page 590 for related details.
SCONNECT error(s) detected by extraction in this cell. See extraction report for details.	This error appears when LVS Report Option EC is specified. Circuit extraction detected Sconnect conflicts in the cell. The error appears in the LAYOUT INPUT ERRORS section. See “ Input Errors ” on page 590 for related details.

Table 14-2. Secondary Messages—Errors (cont.)

Error	Description
Short circuit(s) detected by extraction in this cell. See extraction report for details.	This error appears when LVS Report Option ES is specified. Circuit extraction detected text shorts in the cell. The error appears in the LAYOUT INPUT ERRORS section of the LVS Report. See “ Input Errors ” on page 590 for related details.

The following are LVS Warnings. These do not cause an INCORRECT comparison status.

Table 14-3. Secondary Messages—Warnings

Warning	Description
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. See “ Information and Warnings ” on page 599 for related details.
Warning: Bad devices in layout.	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. See “ Information and Warnings ” on page 599 for related details.
Warning: Components with non-identical power or ground pins were found.	The number of power or ground pins, or the power or ground pin names of some layout components differ from the corresponding schematic components. The “Information and Warnings” section lists these component types. See “ Information and Warnings ” on page 599 for related details.
Warning: Corresponding hcells have different property rules (see transcript).	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.
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: Duplicate subckt definition "<cell>" at line <N> in file "<pathname>" previously defined at line <N> in file "<pathname>"	A subcircuit is defined twice in a netlist. The first instance encountered is used.

Table 14-3. Secondary Messages—Warnings (cont.)

Warning	Description
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. A discrepancy is a true LVS error and should not be ignored. See “ Instances of Cells With Non-Floating Extra Pins ” on page 591 for reporting details.
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. A discrepancy is a true LVS error and should be fixed. See “ Instances of Cells With Non-Floating Extra Pins ” on page 591 for reporting details.
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: Hcells “<cell>” and “<cell>” are of different kinds (“LVS box cell or empty subcircuit”) and (“Subcircuit”) - correspondence ignored. (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. This warning is also given if one hcell of a corresponding pair is an empty subckt or an LVS Box cell. Detailed information appears in the transcript. The correspondence in such cases is ignored.
Warning: LVS property resolution maximum exceeded.	This warning is issued if the ambiguity resolution by properties, or any other type of ambiguity resolution affected by the LVS Property Resolution Maximum specification statement, is needed during the LVS comparison, and the value specified in that statement is insufficient to ensure that all ambiguous nets or instances are processed by the ambiguity resolution. If this warning is issued, a simplified form of the ambiguity resolution is used for some nets or instances, which can result in false property errors.
Warning: Some hcells were expanded with recoverable errors (LVS EXPAND ON ERROR).	This appears when LVS Expand On Error YES is specified, and certain hcells could be removed from the hcell list to mitigate LVS discrepancies after comparison has been performed and the results analyzed.

Table 14-3. Secondary Messages—Warnings (cont.)

Warning	Description
Warning: Some hcells were predicted bad prior to comparison and expanded (LVS EXPAND ON ERROR).	This appears in an LVS report when LVS Expand On Error YES is specified, and certain hcells could be removed from the hcell list to mitigate LVS discrepancies before comparison has been performed.
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 nmLVS-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: Unbalanced smashed mosfets were matched.	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. See “ Information and Warnings ” on page 599 for related details.
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. See “ Information and Warnings ” on page 599 for related details.

Overall SPICE Syntax Check Results

The SPICE syntax check results appear only in reports created with the -cl or -cs command line options of Calibre nmLVS (SPICE syntax check mode).

See “[SPICE Syntax Check Report](#)” on page 619.

LVS Report Listing Conventions

The LVS Report uses a number of conventions for listing various objects.

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 listed as well.

Text Case Conventions

For comparison results in the LVS Report, the following conventions are used:

- Built-in component types and internally-recognized logic gates (when [LVS Recognize Gates](#) is set to something other than NONE) are shown in uppercase.
- Injected logic structures are shown in lowercase (see “[Injected Components and the LVS Report](#)” on page 507).
- Other elements in the LVS Report are shown using the names as they are written in the source or layout netlist, depending on the context of what is being reported.

Discrepancy Listing Conventions

Calibre nmLVS reports differences between the layout and source circuits as discrepancies, which cause the overall report status to be INCORRECT. 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.

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 **.

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 that results in a change in the number of nets or instances, the port, net and instance counts are shown both for the original circuits (INITIAL NUMBERS OF OBJECTS section) and for the new modified circuits (NUMBERS OF OBJECTS AFTER TRANSFORMATION section).

Net, Instance, and Port Identification

Nets, instances, and ports have defined syntactical forms as shown in this section.

- **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.

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 this:

```
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 SPICE netlister 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

Calibre nmLVS forms logic gates internally by the logic gate recognition feature. A logic gate instance is identified by its type, in parentheses, followed by a list of transistor instances forming the gate.

For example, if LVS Inject Logic NO is specified, logic gate recognition is enabled, and there is some error associated with an instance, you may see something like this:

```
(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\$786 form an inverter.

See “[Logic Gate Recognition](#)” on page 439.

Injected Component Identification

Injected components are formed internally by the logic injection feature of Calibre nmLVS-H. An injected-component instance is identified by its type (always with a leading underscore character) 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.

Here is an example of what you might see if there is an error associated with an injected instance:

```
(_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.

See “[Logic Injection](#)” on page 505 for more information about injected components.

Instance Pin Identification

Calibre nmLVS identifies an instance pin by the instance name or ID and location; followed by a colon (:) and the pin name.

For example:

I\$702:G

This identifies pin G of schematic instance I\$702.

27(230,540):B

This identifies pin B of an extracted mask device with an ID of 27, and a location of X=230, Y=540.

Component Pin Identification

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:

```
F (5 pins) : (a b) c (d e)
```

Logically equivalent or swappable pins are shown in parentheses. 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.

Logic Gate Pin Identification

Logic gates are formed internally by the logic gate recognition feature.

Calibre nmLVS identifies a pin of an internally-generated logic gate with the following format:

```
(gate_type) : { INPUT | OUTPUT }
```

In the case of an INPUT pin, this identification is followed by a list of transistors, which are part of the logic gate implementation and whose gate (G) pins form that particular input of the logic gate.

In the case of an OUTPUT pin, this identification is followed by a list of transistors, which are part of the logic gate implementation, and whose source (S) or drain (D) pins form the output of the gate. For example:

```
(INV) :OUTPUT  
/I$767/MP/MP/I$702:D  
/I$767/MN/MN/I$786:S
```

This shows the output pin of an inverter (INV). The D (drain) pin of transistor /I\$767/MP/MP/I\$702 and the S (source) pin of transistor /I\$767/MN/MN/I\$786 form the inverter.

See “[Logic Gate Recognition](#)” on page 439.

Injected Component Instance Pin Identification

Injected components are formed internally by the logic injection feature of Calibre nmLVS-H.

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 “s” pin of transistor x2/m4.

See “[Logic Injection](#)” on page 505 for more information about injected components.

Unconnected Instance Pin Identification

Calibre nmLVS 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 this unconnected instance pin:

```
INST1:IN(34)
```

LVS Discrepancy Types

The LVS Report contains a number of discrepancy types that are listed in the individual CELL COMPARISON RESULTS sections of the hierarchical report, or at the primary level of the flat report.

- **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.

- **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.

Calibre nmLVS matches instances of different component types when they have similar connections in the source and layout circuits.

The report format shows, for regular instances, the layout instance and the source instance followed by their corresponding component type names. For logic gates generated internally by LVS, the layout and source gate types are indicated in parentheses followed by a list of the transistors forming each gate. For example:

```
-----  
      5  I75  MP          I$34  MN  
      bad component type: MP    component type: MN  
-----
```

Layout instance I75 is an instance of an 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 is matched to the NOR gate that is formed by the 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 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 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:

<pre>** layout_net_name ** ** missing net ** ** no similar net ** ** unmatched net **</pre>	<pre>power supply: source_net_name power supply: source_net_name power supply: source_net_name power supply: source_net_name</pre>
---	--

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:

```
-----  
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.

• **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:

<i>layout_pin_name:layout_net_name</i>	<i>** src_net_name **</i>
<i>layout_pin_name:layout_net_name</i>	<i>** missing net **</i>
<i>layout_pin_name:layout_net_name</i>	<i>** no similar net **</i>
<i>layout_pin_name:layout_net_name</i>	<i>** unmatched net **</i>

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:

<i>** layout_net_name **</i>	<i>src_pin_name:src_net_name</i>
<i>** missing net **</i>	<i>src_pin_name:src_net_name</i>
<i>** no similar net **</i>	<i>src_pin_name:src_net_name</i>
<i>** unmatched net **</i>	<i>src_pin_name:src_net_name</i>

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  
    input: SIGA  
    output: NET1  
    ** SIGB **  
-----  
I$34  BUFF  
    input: /SIGA  
    ** missing net **  
    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.

- **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.

- **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.

In certain cases, a good way to debug this type of discrepancy is to set LVS Recognize Gates NONE and LVS Inject Logic NO in the rule file. This configuration may reveal an underlying connectivity problem that causes gates not to form.

- **Missing Injected Instance**

This 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 Calibre nmLVS-H.

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.

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.

In certain cases, a good way to debug this type of discrepancy is to set LVS Inject Logic NO and LVS Recognize Gates NONE in the rule file. This configuration may reveal an underlying connectivity problem.

- **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 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.

- **Nets With Power/Ground Conflict**

This indicates a power or ground conflict in a layout or source cell due to the propagation of port signals from lower-level cells throughout the hierarchy. This occurs when **LVS Cell Supply YES** is specified. The discrepancy appears in the detailed section of the report for each cell in which there is a conflict. The nets involved in the conflict are reported. For example:

DISC#

```
*****  
1    nets with power/ground conflict caused by propagation of  
     cell port signals:   45 105 VSS5 VCC6
```

In the statistics sections for these cells, the conflicting nets are summarized:

45 layout power/ground conflicts were caused during propagation of cell supplies.

- **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 Calibre nmLVS-H 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.

- **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.

- **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.

- **Short Circuit**

This indicates a short-circuit in the layout. A short circuit is detected when two source nets are connected together in the layout.

The report format shows the layout net on the left and two corresponding source nets on the right. For example:

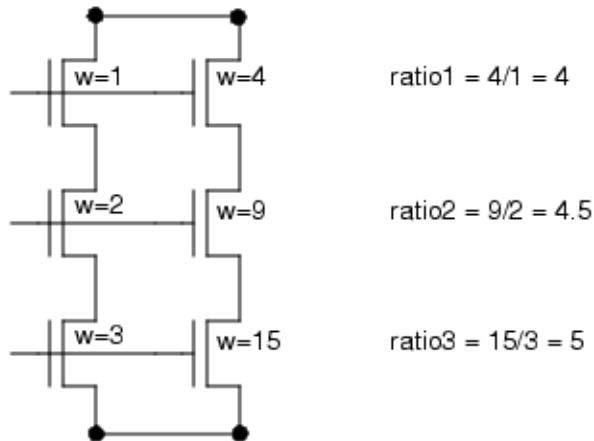
```
-----  
5   Net VDD          VDD  
                 N$87  
-----
```

For the previous example, source nets VDD and N\$87 are connected in the layout to form the single layout net VDD.

- **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 14-1. 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.

- Unmatched objects are objects that cannot be classified by any of the preceding discrepancy types. See “[Unmatched Objects](#)” on page 596 for a listing.

Absolute Trace Property Tolerances Out of Range

This section of the LVS 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.

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
```

The out-of-range warnings can be disabled for property values that are exactly zero using the [LVS Out Of Range Exclude Zero](#) specification statement.

Component Types with Non-Identical Signal Pins

This section of the LVS Report is present only if LVS found component types with different numbers of signal pins or different signal pin names in the layout and source. Note that only primitive component types are reported in this way (for example, primitive devices or LVS Box cells). This is a legitimate error, so it should not be ignored.

Pin differences in hcells are not reported unless the differences cause a discrepancy. Hcell pin differences that cause discrepancies are reported as incorrect instances or nets; the pins themselves are not reported.

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 runs.

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.)

Layout Pin Name	Source Pin Name	Component Type
-----	-----	-----
** missing pin **	B	MN
RESET	** missing pin **	REG5

```
*****
```

In this example, component type MN has a pin B in the source but not in the layout. 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).

In LVS-H, 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

When **LVS Report Option X** is specified, LVS-H also prints a representative list of instances for each configuration of pins being reported. (Note that LVS Report Option X does not operate in flat LVS.) 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 preceding 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.

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. See “[LVS Discrepancy Types](#)” on page 575 for details.

Related Topics

[LVS Report](#)

Errors in Names Given for Power/Ground Nets

This section of the LVS Report contains a list of badly formed power or ground names. A power or ground name is badly formed if it is classified as a non-user-given net name by LVS criteria in both layout and source.

Table 14-4 describes the two sub-sections of this discrepancy.

Table 14-4. Power/Ground Net Errors

Error	Description
Badly Formed Power/Ground Net Names	Contains a list of badly formed power or ground names found in the LVS Power Name and LVS Ground Name specification statements. A name is badly formed if it is not considered user-given by LVS criteria; namely, if it starts with n\$, N\$, i\$, I\$, or contains a slash (/) character. Badly formed power or ground names cause LVS to abort prior to the comparison stage.
Contradictory Power/Ground Net Names	Contains a list of names that are specified as both power and ground names; for example, names that appear both in the LVS Power Name and LVS Ground Name specification statements. Contradictory power or ground names cause LVS to abort prior to the comparison stage.

Hierarchical Cells Forming a Cycle

This section appears in the 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 LVS-H to abort prior to the comparison stage.

The report indicates cell names that form the cycle. For example:

```
*****
CELLS IN HIERARCHY FORMING A CYCLE
*****
Layout Cells           Source Cells
-----
lay3
lay2
lay1
lay4
lay3
               src4
               src3
               src2
               src1
```

This means the following:

- Layout cell lay3 contains an instance of lay2
- lay2 contains an instance of lay1
- lay1 corresponds to source cell src4 in the hcell list

Source cell src4 contains an instance of src3

src3 contains an instance of src2

src2 contains an instance of src1

src1 corresponds to layout cell lay4 in the hcell list

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 -automatch command line switch.)

Here is another example:

Layout Cells	Source Cells
-----	-----
lay3	src3
lay4	src2
lay3	src1
	src3

Here is what happened:

Layout cell lay3 corresponds to source cell src3

Source cell src3 contains an instance of src2

src2 contains an instance of src1

src1 corresponds to layout cell lay4

lay4 contains an instance of lay3 (closing the cycle)

Note that in this example, it is not immediately apparent from the transcript how the cycle is formed. For example, with this transcript, it is possible to go through src1 --> src3 --> lay3 instead of src1 --> lay4 --> lay3. In cases like this, you need to examine the layout, source netlist, or the hcell list to determine how the cycle is formed.

Similar information appears in the transcript of the LVS-H circuit comparison module.

Incorrect Substrate Connections

This section of the LVS Report lists errant substrate connections.

This discrepancy is 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
```

Input Errors

Input errors appear in two sections of the LVS report: LAYOUT INPUT ERRORS and SOURCE INPUT ERRORS. These types of errors must be rectified or comparison will abort or give an INCORRECT status.

In addition to the following messages, errors triggered by [LVS Report Option](#) EB, EC, EO, or ES are reported under LAYOUT INPUT ERRORS but do not cause comparison to abort.

Table 14-5. Input Errors

Error	Description
AMBIGUOUS SERIES PIN NAMES SPECIFIED	A list of instances that are subject to a specification statement of the form LVS Reduce <i>component</i> SERIES <i>pin-name-1</i> <i>pin-name-2</i> but have more than one pin called <i>pin-name-1</i> or more than one pin called <i>pin-name-2</i> . For each instance, LVS indicates the component type, optional component subtype, instance name and ambiguous pin names.
BAD INSTANCES	A list of built-in device type instances that have numbers of pins or pin names that do not follow the LVS conventions. The instances are grouped by component type. Component subtypes are indicated in parentheses, if specified. Refer to the section “ Built-In Device Types ” on page 379.
CONFLICTING INSTANCES	A list of instances with conflicting pin configurations. The instances are grouped by component type. The first instance in each group represents one configuration of pins. The other instances in each group have numbers of pins, pin names, values of the swap_set property, or values of the my_net property, which differ from the first instance. Component subtypes are indicated in parentheses, if specified. Refer to “ Instance Pins and Pin Names ” on page 372.

Table 14-5. Input Errors (cont.)

Error	Description
INCORRECT SERIES PIN SWAP SETS	A list of instances that are subject to a specification statement of the form LVS Reduce <i>component</i> SERIES <i>pin-name-1</i> <i>pin-name-2</i> but in which pins <i>pin-name-1</i> and <i>pin-name-2</i> of the instance are swappable with other pins (that is, pins not named in the particular LVS Reduce <i>component</i> SERIES statement). For each instance, LVS indicates the component type, optional component subtype, instance name and offending pin names (one pin per line).
MISSING COMPONENT TYPES	A list of instances whose component types cannot be determined.
MISSING PIN NAMES	A list of instances, grouped by component type, that have pins whose names cannot be determined. Component subtypes are indicated in parentheses, if specified.
SERIES PIN NAMES NOT FOUND	A list of instances that are subject to a specification statement of the form LVS Reduce <i>component</i> SERIES <i>pin-name-1</i> <i>pin-name-2</i> but have no pins called <i>pin-name-1</i> or have no pins called <i>pin-name-2</i> . For each instance, LVS indicates the component type, optional subtype, instance name and expected pin names (one pin per line).

Instances of Cells With Non-Floating Extra Pins

This section of the LVS Report indicates an instance of a cell with extra pins that are not floating in the layout or source. The extra pins are induced by connections between elements in one design that are not present in the other. This condition is a legitimate error and should be fixed.

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 with its coordinates in parentheses. The corresponding source instance is shown on the right. This is followed by list of extra pins in the form:

```
<instance_path>/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 X1(448.000,-106.200) ff_d X1 ff_d
  ** missing pin ** X1/D ? 138 X1/D:P
```

Here, cell ff_d instance X1 has an extra pin D on net P in the source. This possibly corresponds to pin D of cell ff_d instance X1 in the layout. Net 138 possibly corresponds with net P in the source.

This report is useful in conjunction with the [LVS Report Trivial Ports](#) YES specification statement. Often the missing pin corresponds to a trivial port in the layout.

Layout Discrepancy Reporting Details

When this discrepancy appears in the layout, additional elements are added to the report when there is a chain of non-floating extra pins going down the hierarchy from a given instance. Here is an example:

```
1 X2(-0.578,0.000) block_d X2 block_d
  X2/D:138 ** missing pin ** ? P

LOWER LEVEL NON-FLOATING EXTRA PINS ON THIS NET:
X2/X0/2 (logic_2x)
X2/X0/X3/1 (latch)
```

In this case, layout cell block_d instance X2 has a non-floating pin D on net 138 that is not found in the source. Net 138 is connected to other non-floating extra pins further down the hierarchy within instance X2. This is indicated by the LOWER LEVEL NON-FLOATING EXTRA PINS ON THIS NET section. In that section, instance paths to non-floating pins connected down the hierarchy from X2 are listed. The cell names of the instances with extra pins are given in parentheses.

If more than one instance could be chosen as the representative for a given LOWER LEVEL discrepancy, one is chosen arbitrarily.

The reported chains of non-floating extra pins may represent an actual short in the layout, or they may be related to some other LVS discrepancy.

The presentation of this information in Calibre RVE is somewhat different from what is shown previously, but the intent is the same: show any chains of non-floating extra pins going down the hierarchy from a given net. See “[Tracing Possible Hierarchical Short Chains to Non-Floating Extra Pins](#)” in the *Calibre RVE User’s Manual* for details.

In addition to the discrepancy reports at the sub-cell level, a summary of all non-floating extra pins short chains is included in the top-level cell. This provides a convenient location for all of these discrepancies to be found in addition to being the logical place to reference these types of discrepancies due to all the ways these types of discrepancies can arise.

Here is an example of the top-level summary (formatted to page width):

```
*****
POSSIBLE HIERARCHICAL SHORT CHAINS TO NON-FLOATING EXTRA PINS

DISC#  LAYOUT NET          TO NET
*****
1      B (TOP)           X9/138 (d_core)
                  TO          X9/X2/D (block_d)
                  TO          X9/X2/X0/2 (logic_2x)
                  TO          X9/X2/X0/X3/1 (latch)
```

The instance paths are all with reference to the top level. Not all instances in the paths are necessarily involved in the discrepancy, but they are included for completeness and uniformity of the report. If there are paths of varying lengths for a given discrepancy, the hierarchical path of shortest length is usually used.

There may be cases where a hierarchical short chain proceeds up and then back down in the hierarchy. The trace feature does not specifically handle such cases, but the information it provides should be useful in quickly debugging them. In some cases, high-short resolution may have eliminated some of the pins in the hierarchical short chain. For example, consider this connectivity:

```
TOP      [net 1] <-+<-+
Z       [net 2] --+  |
Y       [net 3] -----+
```

Here both net 3 of cell Y and net 2 of cell Z have been inadvertently shorted to net 1 of cell TOP. But LVS may report the short chain this way:

```
TOP      [net 1]
Z       [net 2] <----+
Y       [net 3] -----+
```

In this case the only function of net 1 in TOP is to connect net 2 in Z to net 3 in Y, so net 1 has been eliminated in a circuit transformation step. But the physical path between the two lower-level nets still proceeds through geometries in TOP, which is why it is essential that the path isolation take place in the context of the top-level cell of the design.

There may be other cases, however, in which the circuit graph is not simplified in this way. In such cases the tool reports each hierarchical short chain from top to bottom. Consider this example:

```
TOP      [net 1] <-+<-+
Z       [net 2] --+   |
Y       [net 3] -----+
```

In this case, it may be that net 2 of an instance of Z is shorted directly to net 3 of an instance of Y, its peer in the design hierarchy. But in order to establish connectivity, the circuit extractor would have to add net 1 of TOP, as well as an extra pin on both Z and Y, with both of them connected to this new net 1. Generally, the tracing feature would report two separate possible hierarchical short chains, one for each of the two non-floating extra pin discrepancies in TOP. You could then determine that net 1 in TOP does not actually correspond to any geometries in the layout, but that it is at the root of two possible hierarchical short chains. Based on that observation, you could isolate a path between net 2 of Z and net 3 of Y, since these nets are at the bottom of the two short chains.

Name Errors

These sections of the LVS Report detail discrepancies in instance, port, and net names.

These discrepancies appear when [LVS Report Option N](#) is used and there are layout names missing in the source. In addition to this error:

```
Error:      Layout names missing in source.
```

The following sections can appear in the LVS Report:

```
*****
                INSTANCE NAME ERRORS

DISC#      LAYOUT NAME                      SOURCE NAME
*****
1          Mfive                           ** missing name **

*****
                NET NAME ERRORS

DISC#      LAYOUT NAME                      SOURCE NAME
*****
2          PORT1                          ** missing name **
3          INPUT1                         ** missing name **

*****
                PORT NAME ERRORS

DISC#      LAYOUT NAME                      SOURCE NAME
*****
4          PORT1                          ** missing name **
```

Property Errors

This section of the LVS Report 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 423 for more information. An *unknown* property value is indicated with a question mark and the words “reduced instance.” For example:

```
p: ? (reduced instance)
```

indicates that the value of property p is unknown. This could be because p is not one of the standard properties for that device type, or because there was not sufficient data to compute an effective value for p. For example, in the latter case:

```
*****
PROPERTY ERRORS
DISC# LAYOUT SOURCE ERROR
*****
1 27(230,540) MD /I$867 MD
w: 5.2u w = 5u 4%
l: 2.2u l = 2u 10%
2 35(120,70) ME /I$302 ME
L = 12.2u L = 12u 2%
*****
```

Two discrepancies are listed. Discrepancy number 1 involves layout instance 27 at location (X=230, Y=540), with component type MD and source instance /I\$867. The layout width value is 5.2 um, the source width value is 5 um, and the error is 4 percent. The layout length value is 2.2 um, the source length value is 2 um, and the error is 10 percent. Discrepancy number 2 involves layout instance 35 at location (X=120, Y=70), with component type ME, and source instance /I\$302. The layout length value is 12.2 um, the source length value is 12 um, and the error is 2 percent. The width value for this instance is not listed because it is not involved in an error.

Unmatched Objects

Unmatched objects in the LVS Report 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. Unmatched objects are reported 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 OBJECTS 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 objects usually disappear from the report once all discrepancies are corrected.

- **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.

- **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 Calibre nmLVS-H.

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 `_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.

- **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 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  
-----
```

Information and Warnings

The INFORMATION AND WARNINGS section of the LVS Report provides messages 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 14-6 lists these statements.

Table 14-6. Information and Warnings

Message	Description
Ambiguity Resolution Points	Provides a list containing pairs of nets, instances, and ports, which belong to interchangeable parts of the circuit, and are matched arbitrarily by LVS. For each pair of arbitrarily matched elements, the layout element is reported on the left and the source element is reported on the right. See the section “ Ambiguity Resolution ” on page 394 for more information.
Bad Devices	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 320.
Component Types With Non-Identical Power Or Ground Pins	Provides a list of component types that have different power or ground pins in the layout and source. LVS lists the component types and pin names and indicates each pin as missing in either the layout or source component type. The format is similar to the one used for “ Component Types with Non-Identical Signal Pins .” While LVS treats differences in signal pins as errors, differences in power or ground pins are treated as warnings.
Conflicting Layout Names	Provides a list of name conflicts in the layout caused by the representation of several circuit elements by a single virtual element. For example, all ports found on a single net are represented by a single virtual port or a group of parallel transistors can be represented by a single virtual transistor. The new virtual element inherits all user-given names from the original elements. Each name listed appears with another name on a single virtual element in the layout, but the two names appear on different elements in the source. LVS does not use these names as initial correspondence points.

Table 14-6. Information and Warnings (cont.)

Message	Description
Conflicting Source Names	Provides a list of name conflicts in the source caused by the representation of several circuit elements by a single virtual element. For example, all ports found on a single net are represented by a single virtual port or a group of parallel transistors can be represented by a single virtual transistor. The new virtual element inherits all user-given names from the original elements. Each name listed appears with another name on a single virtual element in the source, but the two names appear on different elements in the layout. LVS does not use these names as initial correspondence points.
Cpoints	Pairs of nets that were matched by LVS Cpoint specification statements are reported in this section. Only Cpoints that are actually used by LVS appear in this section. Cpoints that could not be used appear in the “Failed Cpoints” section. Note that the format used for reporting Cpoint net names in the LVS report is the same as for nets in general. This may be different from the way Cpoints are entered in the rule file. Specifically, flat LVS omits the X subcircuit call designators in hierarchical pathnames in SPICE.
Failed Cpoints	Cpoints that could not be used by LVS are reported in this section. Note that the format used for reporting Cpoint net names in the LVS report is the same as for nets in general. This may be different from the way Cpoints are entered in the rule file. Specifically, flat LVS omits the X subcircuit call designators in hierarchical pathnames in SPICE.
Initial Correspondence Points	Provides a list containing pairs of identically named nets, ports, and instances used as initial correspondence points. See the section “ Initial Correspondence Points ” on page 369, for more information.
Isolated Layout Nets	Provides a list of layout nets that are not connected to any instances nor connected to any ports. LVS discards these nets during comparison unless they are at the top level and have user-given names that match names in the source. Discarded isolated layout nets give this message in the Statistics section: <code><n> layout net [s] discarded.</code> See “ Isolated Nets ” on page 430 for more details.
Isolated Source Nets	Provides a list of source nets that are not connected to any instances nor connected to any ports. These are handled in a similar way as described under Isolated Layout Nets.

Table 14-6. Information and Warnings (cont.)

Message	Description
Layout Names That Appear On More Than One Element	Provides a list of user-given names that appear on more than one layout net, more than one layout instance, or more than one layout port. LVS does not use these names as initial correspondence points.
Layout Names That Are Missing In The Source	Provides a list of user-given net, instance, and port names in the layout, which are not present in the source.
Layout/Source FY, GY, M, and N Filtered Devices That Did Not Connect S And D Pins	These two sections (one for layout and one for source) report transistors for which unused device filter options FY, GY, M, or N could not connect together the source and drain nets because the source and drain were connected to different pads. For each transistor, the report shows the instance name, the respective filter option, and the source and drain nets.
Layout/Source Instances With Undetermined Reduction TOLERANCE Properties	These two sections (one for layout and one for source) report instances that caused device reduction to cease locally. This stoppage happened because a device reduction TOLERANCE keyword was specified, but the value for the respective property was unknown on the instance. The instance in question was reduced from other instances, and the effective property value could not be computed. For each instance, LVS reports the instance name, component type, and respective property name. See “ Tolerance in Device Reduction ” on page 424.
Matched Mosfets Which Have Been Unequally Reduced	Provides a list of MOS transistor groups (component types MN, MP, ME, MD, LDDN, LDDP, LDDE, and 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 “ Parallel MOS Transistor Reduction ” on page 398 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 where they are missing.
Matching Deleted Isolated Layout Nets	Provides names of isolated layout nets from cells other than the top level that match source names. These nets are discarded from the internal database during comparison.
Numbers of Matched And Unmatched Elements	Provides the numbers of matched and unmatched ports and nets, and instances of each component type and subtype in the layout and source. LVS reports Instance counts by component type and subtype. When an instance with a subtype is matched to an instance with no subtype, the one with a specified subtype determines the subtype of both. In case of conflict, the source instance determines the subtype.

Table 14-6. Information and Warnings (cont.)

Message	Description
Passthrough Layout Nets And Their Ports	Provides a list of layout nets that are connected only to ports of the top-level cell; the ports are also reported. Calibre nmLVS-H ignores these nets during comparison unless they or their ports have user-given names that are also present in the source. This condition can result in an INCORRECT comparison if LVS Report Option LPE is specified.
Passthrough Source Nets And Their Ports	Provides a list of source nets that are connected only to ports of the top-level cell; the ports are also reported. Calibre nmLVS-H ignores these nets during comparison unless they or their ports have user-given names that are also present in the source. This condition is only reported if LVS Report Option SP or SPE is specified. This condition can result in an INCORRECT comparison if LVS Report Option SPE is specified.
Pinswapped Instance Connections	Provides a list of swapped pins for device instances. The model names of the instances match those specified in an LVS Report Pinswapped statement. The swapped pins are indicated by asterisks, like this: M0 (12.000, -2.800) M(aaa) g: B10I *s: I *d: VSS b: VSS
Source Names That Appear On More Than One Element	Provides a list of user-given names that appear on more than one source net, more than one source instance, or more than one source port. LVS does not use these names as initial correspondence points.
Statistics	Provides various statistical information about the LVS run.
Voltage Names Matched by Wildcard	Provides a list of supply net names that have characters matched by a wildcard (?) in an LVS Ground Name or LVS Power Name statement. The matched supply net names are reported as follows: <ul style="list-style-type: none"> o Voltage Names Matched by Wildcard: <p style="padding-left: 20px;">Power Names from Layout:</p> <p style="padding-left: 40px;">VDD_XYZ</p> <p style="padding-left: 20px;">Power Names from Source:</p> <p style="padding-left: 40px;">VDD_XYZS</p>

Detailed Error Analysis Reporting

The detailed error analysis functionality is enabled using LVS Report Option FX in the rule file. When detailed error analysis is enabled, Calibre nmLVS performs additional analysis of discrepancies between layout and source.

This analysis detects many common circuit errors, such as open circuits, short circuits, and cross-connected instance pins. Detailed error descriptions appear for each error. The source is always assumed to be correct.

The heading of the section is DETAILED ERROR ANALYSIS, and it contains one record per each identified error. Each error record contains a brief description of the error type, the list of the instances involved in the error, the associated pins, and the connected nets. The report format is similar to the DETAILED INSTANCE CONNECTIONS section. Lastly, there is a detailed description of the error that can assist in fixing it.

The errors for which a detailed report is generated are not repeated in the DETAILED INSTANCE CONNECTIONS section of the LVS report. However, the same instances could appear in the INCORRECT NETS section if some of the nets connected to these instances are also in error.

The errors recognized by the detailed error analysis are described in subsequent sections. Note that error analysis is based on the partial circuit matching done by Calibre nmLVS and is affected by the context of the error. In particular, when the decision about which parts of the circuit should be considered matched is ambiguous, error analysis can detect apparently different errors in very similar circuits. Note also that Calibre nmLVS cannot interpret the intent of the design; therefore, an error fixing suggestion, if followed, makes that particular error disappear. However, the fix may or may not be the right one for the overall design.

Fix suggestions involving logic gates or logic injection circuits report a component instance (such as _invv:x0/m1) rather than simply the component type. This makes relating the instances mentioned in the report to the layout and source instances easier.

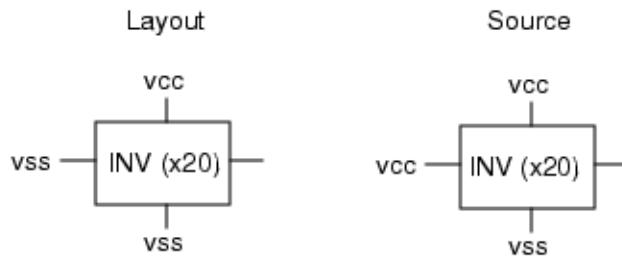
Bad Instance Connections Error Detailed Reporting	604
Bad Instance Pair Error Detailed Reporting	604
Circular Connection Error Detailed Reporting	605
Cross-Connect and Pin-Swap Errors Detailed Reporting.....	606
Cross-Connect Error Detailed Reporting	608
Logic Gate Bulk Pin Error Detailed Reporting	608
Logic Gate Supply Error Detailed Reporting.....	609
Net Connection Error Detailed Reporting	610
Open Circuit Detailed Reporting	611
Pin Connection and Open Circuit Error Detailed Reporting	613

Pin Connection and Short Circuit Error Detailed Reporting	614
Pin Connection Error Detailed Reporting.....	615
Pin Swap Error Detailed Reporting.....	615
Short Circuit Detailed Reporting.....	616

Bad Instance Connections Error Detailed Reporting

Bad instance connections are detected when a layout instance pin is connected to a wrong net.

Figure 14-2. Bad Instance Connections Error



When LVS Report Option FX is used, the error is reported as follows:

```
-----  
Bad instance connections  
-----  
x20 inv  
OUT: out1  
VCC: vcc  
VSS: vss  
IN: vss  
** vcc **  
x20 inv  
OUT: out1  
VCC: vcc  
VSS: vss  
** vss **  
IN: vcc  
-----  
Pin IN of LAYOUT instance x20[inv] connects to net vss instead of net vcc  
-----
```

When several pins of the same instance are wrongly connected, the report lists all bad connections for the instance in one section, for example:

```
Pin IN1 of LAYOUT instance x10[cell1] connects to net a instead of net b  
Pin IN2 of LAYOUT instance x10[cell1] connects to net c instead of net d
```

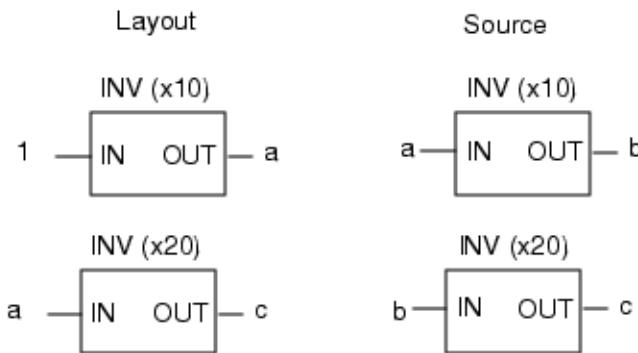
When the same pin of two instances connects to the same incorrect net, then the two instances can be reported together as one error:

```
Pin g of LAYOUT instance m1[MP(P)] and pin g of instance m2[MN(N)] connect  
to net a instead of net b
```

Bad Instance Pair Error Detailed Reporting

Bad instance pairs are detected when a pair of instances in series is incorrectly connected.

Figure 14-3. Bad Instance Pair Error



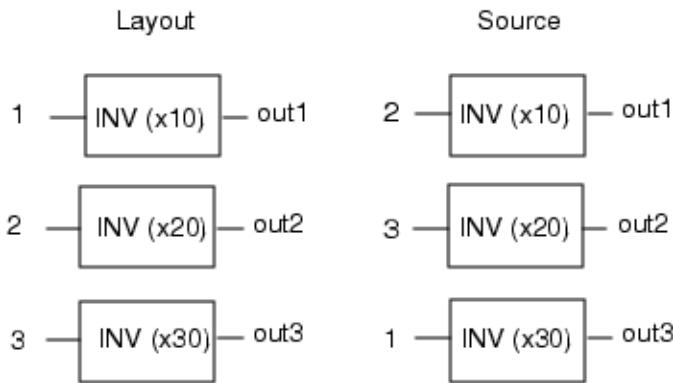
When LVS Report Option FX is used, the error is reported as follows:

```
-----  
Bad instance pair  
  
x10 inv  
VCC: vcc  
VSS: vss  
IN: 1  
OUT: a  
** a **  
** no similar net **  
  
x20 inv  
OUT: c  
VCC: vcc  
VSS: vss  
IN: a  
** no similar net **  
  
x10 inv  
VCC: vcc  
VSS: vss  
** no similar net **  
** a **  
IN: a  
OUT: b  
  
x20 inv  
OUT: c  
VCC: vcc  
VSS: vss  
** a **  
IN: b  
  
Pin OUT of LAYOUT instance x10[inv] and pin IN of LAYOUT instance  
x20[inv] connect to net a instead of (missing) LAYOUT net corresponding  
to SOURCE net b  
Pin IN of LAYOUT instance x10[inv] connect to net 1 instead of net a  
-----
```

Circular Connection Error Detailed Reporting

Circular connections are detected when several layout instances are incorrectly connected in a cyclical fashion.

Figure 14-4. Circular Connection Error



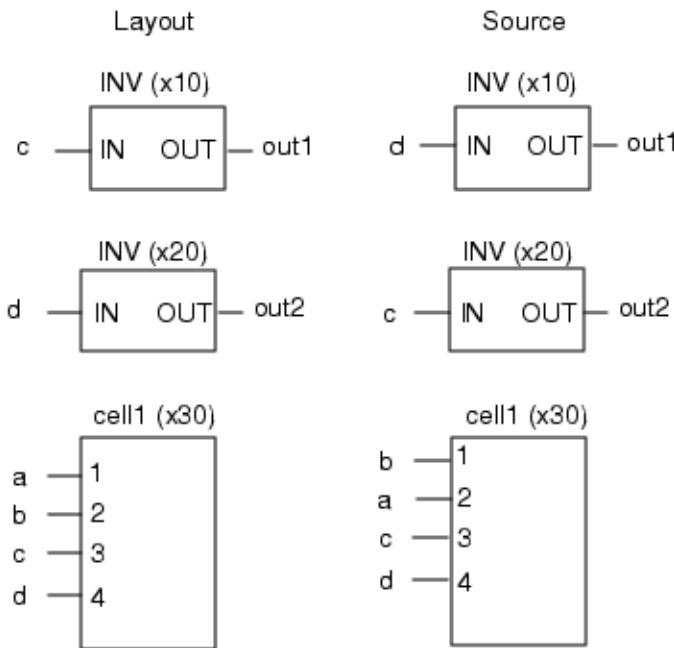
When LVS Report Option FX is used, the error is reported as follows:

```
-----  
Circular connection error (three instances)  
  
x10 inv  
  OUT: out1  
  VCC: vcc  
  VSS: vss  
  IN: 1  
  ** 2 **  
  
x30 inv  
  OUT: out3  
  VCC: vcc  
  VSS: vss  
  IN: 3  
  ** 1 **  
  
x20 inv  
  OUT: out2  
  VCC: vcc  
  VSS: vss  
  IN: 2  
  ** 3 **  
  
x10 inv  
  OUT: out1  
  VCC: vcc  
  VSS: vss  
  ** 1 **  
  IN: 2  
  
x30 inv  
  OUT: out3  
  VCC: vcc  
  VSS: vss  
  ** 3 **  
  IN: 1  
  
x20 inv  
  OUT: out2  
  VCC: vcc  
  VSS: vss  
  ** 2 **  
  IN: 3  
  
LAYOUT net 3 connects to instance x30[inv], pin IN instead of instance  
x20[inv], pin IN  
LAYOUT net 2 connects to instance x20[inv], pin IN instead of instance  
x10[inv], pin IN  
LAYOUT net 1 connects to instance x10[inv], pin IN instead of instance  
x30[inv], pin IN  
-----
```

Cross-Connect and Pin-Swap Errors Detailed Reporting

This error is detected when there is a combination of layout cross-connections and pin-swapping.

Figure 14-5. Cross-Connect and Pin Swap Errors



When LVS Report Option FX is used, the errors are reported as follows:

```
-----
                                         Cross-connect and pin swap errors

          x10  inv
          OUT: out1
          IN: c
          ** d **

          x20  inv
          OUT: out2
          IN: d
          ** c **

          x30  cell1
          3: c
          4: d
          1: a
          2: b
          ** b **
          ** a **

          x10  inv
          OUT: out1
          ** c **
          IN: d

          x20  inv
          OUT: out2
          ** d **
          IN: c

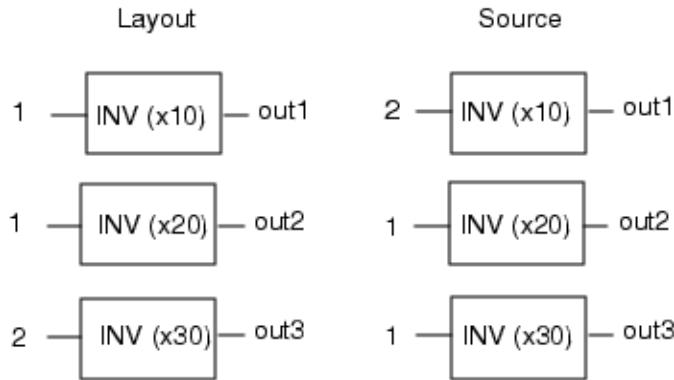
          x30  cell1
          3: c
          4: d
          ** a **
          ** b **
          1: b
          2: a

          LAYOUT nets 1(pin IN) and 2(pin IN) are cross-connected to instances
          x10[inv] and x20[inv]
          Swapped pins 1 and 2 of LAYOUT instance x30[cell1] (connected nets a and
          b, respectively)
-----
```

Cross-Connect Error Detailed Reporting

Cross-connects are detected when two layout nets are interchanged to two instances.

Figure 14-6. Cross-Connected Instances Error



When [LVS Report Option FX](#) is used, the error is reported as follows:

```
-----  
Cross-connect error  
  
X10 inv  
VSS: vss  
VCC: vcc  
IN: 1  
OUT: out1  
** out3 **  
  
X30 inv  
VSS: vss  
VCC: vcc  
IN: 1  
OUT: out2  
** out1 **  
  
X30 inv  
VSS: vss  
VCC: vcc  
IN: 2  
OUT: out3  
** out1 **  
  
X10 inv  
VSS: vss  
VCC: vcc  
IN: 2  
OUT: out1  
** out3 **  
  
-----  
LAYOUT nets out1(pin OUT) and out3(pin OUT) are cross-connected to  
instances X10[inv] and X30[inv]
```

Logic Gate Bulk Pin Error Detailed Reporting

Logic gate bulk pin errors are detected when a layout bulk pin of a logic gate is connected to the wrong net.

This error is reported similarly to the one shown under [Logic Gate Supply Error Detailed Reporting](#). Instead of the power and ground being incorrect, the bulk pins of the transistors in the gate are connected incorrectly.

When LVS Report Option FX is used, the errors are reported as follows:

```

-----  

Bad instance connections  

-----  

( INV )  

  output: 1  

  input: in  

Transistors:  

  m1      MP (P)  

  m2      MN (N)  

  m1      MP (P)  

  m2      MN (N)  

Bulk pin b of LAYOUT device m1 in gate (INV) connects to net vcc1 instead  

of net vcc  

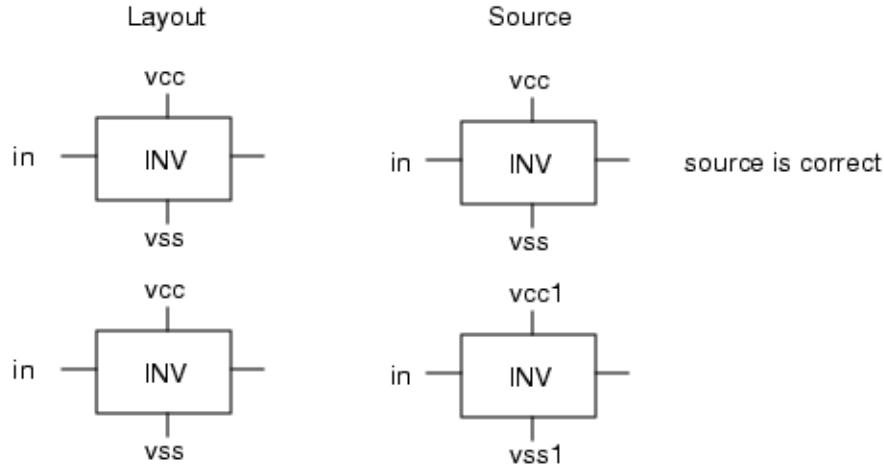
Bulk pin b of LAYOUT device m2 in gate (INV) connects to net vss1 instead  

of net vss
-----
```

Logic Gate Supply Error Detailed Reporting

Logic gate supply errors are detected when a layout supply pin of a logic gate is connected incorrectly. If the correct layout nets can be identified, they are listed in the fix suggestion. Otherwise, the report refers to a “missing net” and gives the corresponding source net. This error is detected only when logic gate recognition is enabled.

Figure 14-7. Logic Gate Supply Error



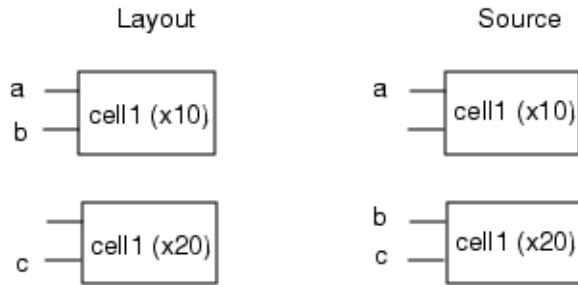
When LVS Report Option FX is used, the errors are reported as follows:

```
-----  
                                Bad instance connections  
  
(INV)                               (INV)  
  output: 1                           output: 1  
  input: in                            input: in  
  
Transistors:  
  m1      MP (P)                      m1      MP (P)  
  m2      MN (N)                      m2      MN (N)  
  
Power supply of LAYOUT gate (INV) connects to net vcc instead of  
(missing) LAYOUT net corresponding to SOURCE net vcc1  
Ground supply of LAYOUT gate (INV) connects to net vss instead of  
(missing) LAYOUT net corresponding to SOURCE net vss1  
-----
```

Net Connection Error Detailed Reporting

Net connection errors are detected when a layout net is connected to the wrong instance pin.

Figure 14-8. Net Connection Error



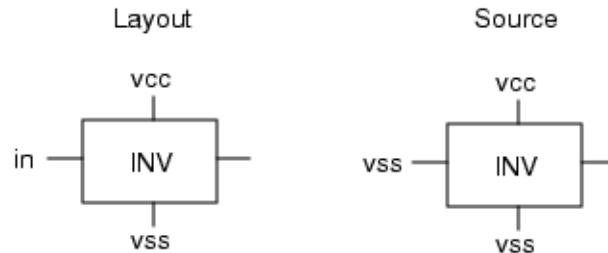
When LVS Report Option FX is used, the error is reported as follows:

```
-----  
                                Net connection error  
  
x10  cell1                               x10  cell1  
 1: a                                     1: a  
 2: b                                     ** b **  
 ** no similar net **                     2: 11  
  
x20  cell1                               x20  cell1  
 2: c                                     2: c  
 1: 10                                    ** no similar net **  
 ** b **                                   1: b  
  
LAYOUT net b(pin 2) connects to instance x10[cell1], pin 2 instead of  
x20[cell1], pin 1  
-----
```

Open Circuit Detailed Reporting

Open circuit detailed reports are written when two layout nets match one source net.

Figure 14-9. Open Circuit Error

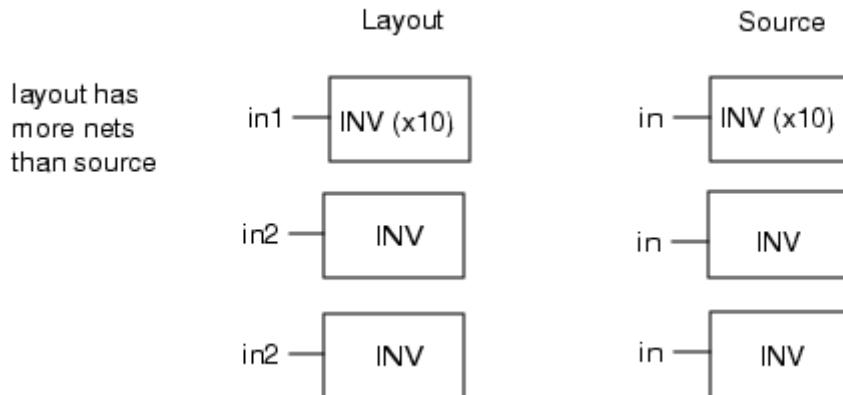


When [LVS Report Option FX](#) is used, the error is reported as follows:

```
-----  
          Open circuit error  
-----  
LAYOUT nets vss and in must be connected to match SOURCE net vss  
-----
```

The previous open circuit error is reported when both layout nets involved in the discrepancy can be completely analyzed, and it is established that connecting them together results in the correct match with the corresponding source net.

However, in more complex cases only a partial analysis is performed. Consider the following case. The source instance pins are all on the same net, whereas the layout has two nets for the corresponding instances.



The error is reported as follows:

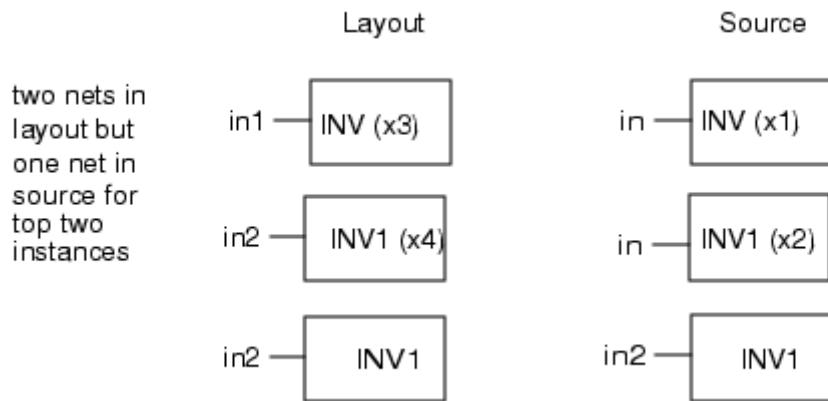
```
-----  
Pin open-circuit error  
  
x10 inv  
  OUT: out  
  VCC: vcc  
  VSS: vss  
  IN: in1  
  ** in2 **  
  
          x10 inv  
          OUT: out  
          VCC: vcc  
          VSS: vss  
          ** missing net **  
          IN: in
```

Pin IN of LAYOUT instance x10[inv] connects to net in1 instead of net in2
Possibly LAYOUT nets in2 and in1 must be connected to match SOURCE net in

The report shows the matched subcircuits and gate types, followed by the “pin name: net name” pairings. The missing elements from the corresponding design are reported between the ** markers.

Because a complete analysis was not possible in this case, the report can only suggest what might resolve the error. Additional knowledge of the design and other errors are required to determine what the correct course of action would be.

If the layout contains an open circuit, but the source contains nets that could match to one of the layout nets in question, then the open circuit may be reported in the context of the pins involved, without the suggestion to connect the layout nets. Consider the following case. The bottom instances in layout and source could be matched, but the other two contain an open circuit.



The error is reported as follows:

```
-----  
Open-circuit error  
  

x3 inv  
OUT: c  
VCC: vcc  
VSS: vss  
IN: in1  
** in **  
  

x4 inv1  
OUT: c  
VCC: vcc  
VSS: vss  
IN: in2  
** in **  
  

x1 inv  
OUT: c  
VCC: vcc  
VSS: vss  
** in1 **  
IN: in  
  

x2 inv1  
OUT: c  
VCC: vcc  
VSS: vss  
** in2 **  
IN: in  
  

Pin IN of LAYOUT instance x3[inv] connects to net in1 instead of net in  

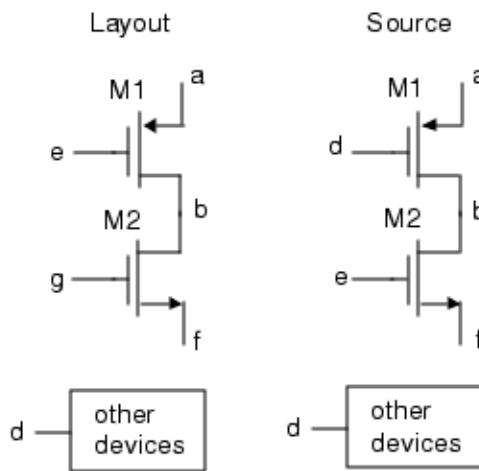
Pin IN of LAYOUT instance x4[inv1] connects to net in2 instead of net in
-----
```

When there are multiple pin open circuit errors, then the errors are grouped by net in a single section.

Pin Connection and Open Circuit Error Detailed Reporting

This error is caused by an incorrectly connected layout pin, possibly combined with an open circuit error.

Figure 14-10. Pin Connection and Open Circuit Error



This errors are reported as follows:

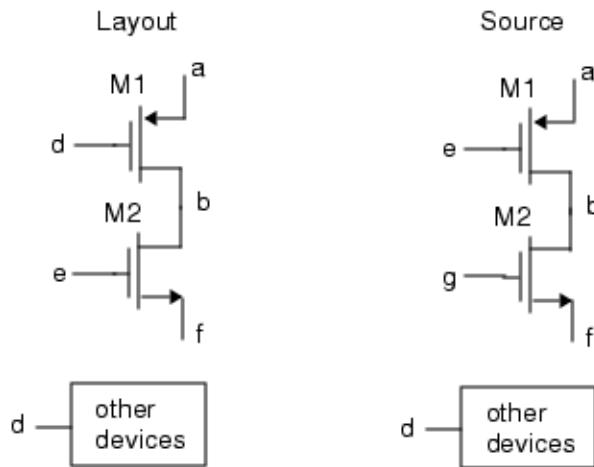
```
-----  
Pin connection and open-circuit errors  
  
m1 MP(P)  
  s: a  
  d: b  
  b: c  
  g: e  
  ** d **  
  
m2 MP(P)  
  s: f  
  d: b  
  b: c  
  g: g  
  ** e **  
  
m1 MP(P)  
  s: a  
  d: b  
  b: c  
  ** e **  
  g: d  
  
m2 MP(P)  
  s: f  
  d: b  
  b: c  
  ** missing net **  
  g: e
```

Pin g of LAYOUT instance m1 [MP(P)] connects to net e instead of net d
Possibly LAYOUT nets d and g must be connected to match SOURCE net e

Pin Connection and Short Circuit Error Detailed Reporting

This error is caused by an incorrectly connected layout pin, possibly combined with a short-circuit error.

Figure 14-11. Pin Connection and Short Circuit Error



Net d is connected to other devices in both layout and source but net g does not have an equivalent net in the layout

This errors are reported as follows:

```
-----  
Pin connection and short-circuit errors  
  
m1 MP(P)  
  s: a  
  d: b  
  b: c  
  g: d  
  ** e **  
  
m2 MP(P)  
  s: f  
  d: b  
  b: c  
  g: e  
  ** missing net **  
  
m1 MP(P)  
  s: a  
  d: b  
  b: c  
  ** d **  
  g: e  
  
m2 MP(P)  
  s: f  
  d: b  
  b: c  
  ** e **  
  g: g
```

Pin g of LAYOUT instance m1 [MP(P)] connects to net d instead of net e
Possibly LAYOUT net e is a shorted net matching SOURCE nets e and g

Pin Connection Error Detailed Reporting

Pin connection errors are detected when a layout net is connected to a different pin of the same instance than in the source.

Figure 14-12. Pin Connection Error



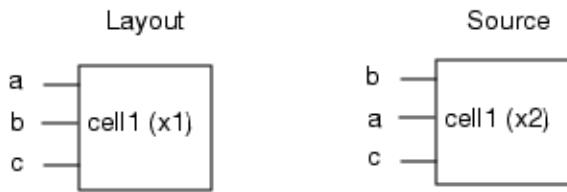
When LVS Report Option FX is used, the error is reported as follows:

```
-----  
Pin connection error  
  
x10 cell1  
  1: a  
  2: 1  
  ** no similar net **  
  ** a **  
  
x10 cell1  
  ** a **  
  ** no similar net **  
  1: 2  
  2: a  
  
LAYOUT net a connects to pin 1 instead of pin 2 of instance x10[cell1]
```

Pin Swap Error Detailed Reporting

Pin swap errors are reported when two layout pins of an instance are cross-connected.

Figure 14-13. Pin Swap Error



When LVS Report Option FX is used, the error is reported as follows:

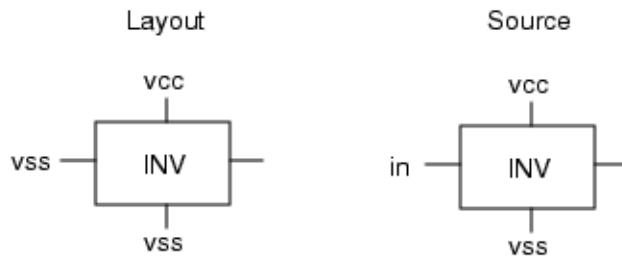
```
-----  
Pin swap error  
  
x1  cell1  
3: c  
1: a  
2: b  
** b **  
** a **  
  
x2  cell1  
3: c  
** a **  
** b **  
1: b  
2: a  
  
Swapped pins 1 and 2 of LAYOUT instance x1[cell1] (connected nets a and b,  
respectively)  
-----
```

Short Circuit Detailed Reporting

Short circuit detailed reports are written when two source nets match one layout net.

Detailed short circuit error reports are similar to the open circuit errors described under “[Open Circuit Detailed Reporting](#)” on page 611.

Figure 14-14. Short Circuit Error

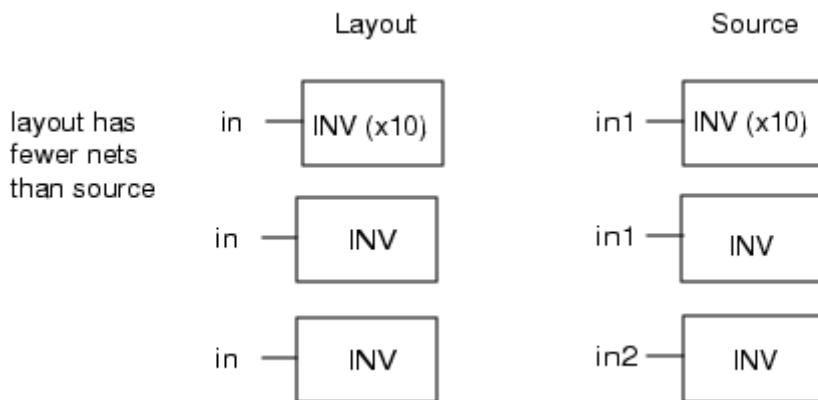


When [LVS Report Option FX](#) is used, the error is reported as follows:

```
-----  
Short circuit error  
  
LAYOUT net vss is a shorted net matching SOURCE nets vss and in  
-----
```

The previous short circuit error is reported when both layout nets involved in the discrepancy can be completely analyzed, and it is established that they are shorted together.

However, in more complex cases only a partial analysis is performed. Consider the following case. The layout instance pins are all on the same net, whereas the source has two nets for the corresponding instances.



The error is reported as follows:

```

-----  

          Pin short-circuit error  

-----
```

x10 inv OUT: out VCC: vcc VSS: vss IN: in ** missing net **	x10 inv OUT: out VCC: vcc VSS: vss ** in2 ** IN: in1
---	--

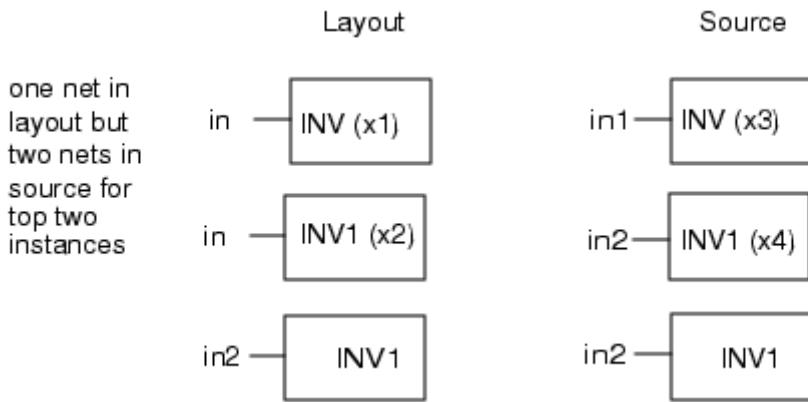
```

Pin IN of LAYOUT instance x10[inv] connects to net in instead of (missing)
LAYOUT net corresponding to SOURCE net in1
Possibly LAYOUT net in is a shorted net matching SOURCE nets in2 and in1
-----
```

The report shows the matched subcircuits and gate types, followed by the “pin name: net name” pairings. The layout is missing a net, so the report indicates this. The tool sees that the missing layout net ought to be connected to in2, so that appears on the source side of the report.

Because a complete analysis was not possible in this case, the report can only suggest what might resolve the error. Additional knowledge of the design and other errors is required to determine what the correct course of action would be.

In a more complex context, the short circuit may be reported in the context of the pins involved, without the suggestion to disconnect the layout nets. Consider the following case. The bottom instances in layout and source could be matched, but the other two contain a short circuit.



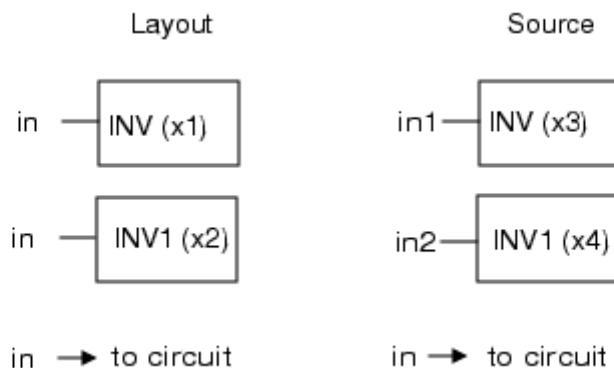
The error is reported as follows:

Short-circuit error

<pre>x1 inv OUT: c VCC: vcc VSS: vss IN: in ** in1 **</pre>	<pre>x3 inv OUT: c VCC: vcc VSS: vss ** in ** IN: in1</pre>
<pre>x2 inv1 OUT: c VCC: vcc VSS: vss IN: in ** in2 **</pre>	<pre>x4 inv1 OUT: c VCC: vcc VSS: vss ** in ** IN: in2</pre>

Pin IN of LAYOUT instance x1[inv] connects to net in instead of net in1
 Pin IN of LAYOUT instance x2[inv1] connects to net in instead of net in2

In some cases, the layout nets corresponding to some of the source nets involved in a short or open circuit error are not present or cannot be detected. This is possibly due to other errors. In this case, the layout error description refers to a “missing net” that must correspond to the specified source net. In such cases, the error analysis can usually analyze the shorted net and issue the likely fix.



The error is reported as follows:

```
-----
                                         Short-circuit error

x1  inv
    OUT: c
    VCC: vcc
    VSS: vss
    IN: in
    ** missing net **

x2  inv1
    OUT: c
    VCC: vcc
    VSS: vss
    IN: in
    ** missing net **

x3  inv
    OUT: c
    VCC: vcc
    VSS: vss
    ** in **
    IN: in1

x4  inv1
    OUT: c
    VCC: vcc
    VSS: vss
    ** in **
    IN: in2

Pin IN of LAYOUT instance x1[inv] connects to net in instead of (missing)
LAYOUT net corresponding to SOURCE net in1
Pin IN of LAYOUT instance x2[inv1] connects to net in instead of (missing)
LAYOUT net corresponding to SOURCE net in2
Possibly shorted LAYOUT net in should match SOURCE nets in1, in2, and in
-----
```

SPICE Syntax Check Report

This kind of report is written only from the -cl and -cs command line options of Calibre nmLVS-H (SPICE syntax check mode).

A SPICE syntax check report consists of these elements:

- LVS netlist compiler errors and warnings, if any were found. Errors are written to the log file and to the LVS report. Warnings are written to the log file only.
- A header section, specifying the report filename, the layout 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.

When an error appears, it is written to the log file and to the LVS report file. Here is an example of an error in the LVS report file:

```
LVS Netlist Compiler - Errors and Warnings for "./src.spi"
-----
Error: Syntax Error in file "./src.spi" at line 19.
Expected "Mxxx nd ng ns <nb> mname <L=l> <W=w> <AD=ad> <AS=as> <PD=ps>
<PS=ps> <NRD=nrd> <NRS=nrs>
+ <OFF> <IC=vds, vgs, vbs> <M=m> <parnam=pval> ...
<$LDD<[type]>> <$X=x> <$Y=y> <$D=d>"
```

```
#####
##          C A L I B R E      S Y S T E M      ##
##          L V S      R E P O R T      ##
##          #####
#####
```

REPORT FILE NAME: lvs_report
LAYOUT NAME:
SOURCE NAME: ./src.spi ('TOPCELL')
RULE FILE: rules
CREATION TIME: Thu Jan 17 14:52:45 2013
CURRENT DIRECTORY: /user/me/lvs
USER NAME: me
CALIBRE VERSION: v2013.1_4 Fri Jan 11 10:16:10 PST 2013

```
*****
***** OVERALL SYNTAX CHECK RESULTS *****
```

```
#
#      #####
# #      #
#      #  SYNTAX CHECK FAILED  #
# #      #
#      #      #####
#      #
```

Syntax errors were found in the source.

```
*****
***** SUMMARY *****
```

Total CPU Time: 0 sec
Total Elapsed Time: 0 sec

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).

See “[Calibre nmLVS and Calibre nmLVS-H Command Line](#)” on page 45 for command line options.

SVDB Cross-Reference Files

Calibre LVS applications can generate Mask SVDB Directory (commonly called SVDB) instance and net cross-reference files. These are useful in identifying layout and source object correspondence.

SVDB Header	623
Flat Instance Cross-Reference File	624
Flat Net Cross-Reference File	626
Hierarchical Instance and Net Cross-Reference Files	627
Source and Layout Placement Hierarchy Files	628

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 an instance cross-reference file:

```
# SVDB: Instance Cross Reference (ixf) (File format 1)
# SVDB: Layout Primary mix
# SVDB: Rules -0 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 the 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 this:

```
file_type path_name date_time_stamp
```

where these are the fields:

- *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 pathname 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
```

Flat Instance Cross-Reference File

You can use the Calibre -ixf command line switch to generate flat instance cross-reference files.

The IXF secondary keyword of the [Mask SVDB Directory](#) specification statement also creates an instance cross-reference file within the SVDB directory. If IXF is specified, the *layout_primary*.ixf file is written to the SVDB only. The *layout_primary* is the primary cell name.

The instance cross-reference file is a text file that contains matched instances. The file contains one line per instance in the following form:

```
layout_id  layout_name(x,y)  source_id  source_name [ SL | SS ] [X]
```

with these definitions:

- *layout_id* is a number that represents the instance (ID) in the layout.
- *layout_name(x,y)* is a user-given name that represents the layout instance. The value of *layout_id* is used if no user-given name was specified. The coordinates are given in parentheses.
- *source_id* is a number that represents the instance (ID) in the source.
- *source_name* is a user-given name that represents the source instance. The value of *source_id* is used if no user-given name was specified.
- SL indicates a reduced layout device. SL is short for smashed layout.
- SS indicates a reduced source device. SS is short for smashed source.
- X indicates a MOS device with swapped source and drain pins.

Matched Devices

The instance cross-reference file represents a reduced device by listing its original devices. When a reduced layout device is matched to a reduced source device, all original layout devices

are listed in consecutive lines on the left, with the original source device repeated on the right. All the remaining original source devices are listed in consecutive lines on the right, with a representative original layout device repeated on the left. The representative devices are chosen at random.

For example:

```
0 0 (-12.000,-1.000) 4 R1
1 1 (-18.000,-1.000) 4 R1 SL
0 0 (-12.000,-1.000) 5 R2 SS
0 0 (-12.000,-1.000) 6 R3 SS
```

In this example, layout devices 0 and 1 are reduced to a single device. The reduced layout device corresponds to source devices R1, R2, and R3, which are also reduced to a single device. Layout device 0 and source device R1 are chosen as representative devices.

Swapped Pins

Capacitor and resistor instances with swapped pos and neg pins, and MOS devices with swapped source and drain pins are indicated with the letter X.

For example:

```
% M test LAY 5 M test SRC 5
0 MMO 0 MMM0
0 MM1 0 MMM1 X
0 RR0 0 RRR0
0 RR1 0 RRR1 X
0 CC0 0 CCC0
0 CC1 0 CCC1 X
```

For capacitor and resistor devices, the trailing X indicates the pos pin of the layout device corresponds to the neg pin of the source device and vice versa.

For MOS devices, the trailing 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.
```

Related Topics

[SVDB Cross-Reference Files](#)

Flat Net Cross-Reference File

The Calibre -nxf command line option instructs LVS to generate a flat net cross-reference file. The NXF secondary keyword of the Mask SVDB Directory specification statement also creates a net cross-reference file within the SVDB directory.

When the -nxf option is used, Calibre creates a file named *lvs_report_name.nxf*, where *lvs_report_name* is the [LVS Report](#) name.

When the NXF keyword is used, the file 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(x,y) source_id source_name
```

where these are the definitions:

- *layout_id* is a number that represents the net (ID) in the layout.
- *layout_name(x,y)* is a user-given name that represents the layout net. The value of *layout_id* is used if no user-given name was specified. The coordinates are given in parentheses.
- *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.

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
```

Calibre nmLVS 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.

Unmatched nets do not appear in the net cross-reference file.

Related Topics

[SVDB Cross-Reference Files](#)

Hierarchical Instance and Net Cross-Reference Files

The comparison stage of Calibre nmLVS-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 an SVDB header, see section “[SVDB Header](#)” on page 623. After the SVDB header, the file contains one section for each LVS correspondence cell (or 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 624 and “[Flat Net Cross-Reference File](#)” on page 626.

Within NXF and IXF files, the slash character (/) is a hierarchy delimiter. However, Calibre also permits instance and net names to contain slashes. Thus, delimiters in the IXF and NXF files can be ambiguous—some slashes are hierarchy delimiters, while others are parts of instance or net names. This can cause problems with some third-party tools that attempt to read these files.

Within the Calibre connectivity interface (CCI), the hierarchy delimiter used in output SPICE netlists can be controlled with the dfm::set_layout_netlist_options -hier_separator or HIERARCHY_SEPARATOR commands. These apply only to CCI.

Source and Layout Placement Hierarchy Files

The comparison stage of Calibre nmLVS-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. Pin count is the only connectivity information present.

Each file begins with an 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:

```
# 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
...
...
```

Binary Polygon Format (BPF) Database

The BPF database provides an interface from flat Calibre nmLVS to external tools. It provides access to shapes on layers involved in circuit extraction and device recognition, and to some other related information. You create the BPF database with the -bpf command line option in flat LVS (-lvs with no -flatten option). You cannot create the BPF database in Calibre nmLVS-H.

The database consists of a set of files. File names in the BPF database are based on the rule file [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 Polygon Files — BPF files contain polygons from certain layers of interest. The BPF files created have names of the form *lvs_report_name.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 LVS -dblayers command line option to select layers for generation explicitly.

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:

```
ngate1 = COPY ngate
DEVICE MN ngate1 ngate(G) nact(S) nact(D) dbsub(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.
```

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>
```

BPF Layout Cross Reference File — The layout cross-reference file, *lvs_report_name.lxf*, is a text file that provides a cross reference between internal net numbers and layout text names.

BPF Ports File — The *lvs_report_name.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:

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 — The coordinates (dbu) of the database text object when using Port Layer Text, or of a vertex on the port polygon marker when using [Port Layer Polygon](#), or [Port Layer Merged Polygon](#).

port_layer_attached — The 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 option 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
```

Mask Results Database

The LVS mask results database is optional and is specified by the Mask Results Database rule file statement. It is used in ICtrace only and cannot be loaded into Calibre RVE. This database is not related to the Standard Verification Database (SVDB) and is not used in Calibre-only flows.

You can load the results from flat Calibre nmLVS into Pyxis Layout for interactive debugging in ICtrace™ Mask mode. You can also load and view results with the Verification DataPort tool.

The database contains extracted nets and devices resulting from the execution of connectivity-related operations and statements contained in a rule file. You use this database type when interpreting the results graphically in Pyxis Layout. You can exclude information from this database by using the NOPROBE and NOCONTACT options, and the -dblayers command line option.

Chapter 15

SPICE Format

A SPICE netlist serves as a source of connectivity information for hierarchical or flat verification for various Calibre tools that process such netlists. Calibre tools support a subset of SPICE 2 syntax with extensions. Some of these extensions are proprietary to Calibre, and some coincide with selected ELDO, HSPICE, or Cadence CDL constructs.

When reading this material, knowledge of the SPICE language is assumed.

SPICE Syntax Conventions	634
Control Statements	647
Element Statements	677

SPICE Syntax Conventions

Certain syntactical conventions are used for SPICE that are distinct from the syntax conventions used elsewhere in the book.

These conventions apply to the SPICE element syntax shown in this chapter.

Table 15-1. SPICE Netlist Notational Conventions

Syntax	Description
UPPERCASE	Uppercase letters indicate literal keywords.
lowercase	Lowercase letters indicate arguments to be substituted by other values.
<>	Indicates an optional argument.
+	SPICE continuation character used when the syntax description spans across several lines.
<ARG=val>	Unless indicated otherwise, optional parameters preceded by literal names can appear in any order.
<val>	Parameters not preceded by literal names must appear in exactly in the orders shown relative to other such parameters as defined under “ Element Statements ” on page 677.

Case Sensitivity	635
Characters Allowed	635
Numeric Value Types	636
Arithmetic Expressions	638
Comments	639
Comment-Coded Extensions.....	639
String Parameters	640
Inline Parameters in Subcircuit Calls	641
X Instantiated Devices.....	642
Dollar Signs in Cell Names	644
\$D Parameters	644
Cell Statistics	644
SPICE Syntax Checking	645

Case Sensitivity

Unless indicated otherwise, all SPICE 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.

Characters Allowed

Any characters not listed as restricted are allowed for names.

Names include node names, element names, property names, subcircuit names, and subcircuit call names. Property names (device element parameter names) must begin with a letter. Certain names, such as built-in element names (like C, D, M, Q, and so forth) must begin with the appropriate letters. The required forms of such names are listed under “[Element Statements](#)” on page 677.

The following characters are problematic for use in names in Calibre SPICE netlists:

- Whitespace characters (space, tab, carriage-return, newline, vertical-tab, form-feed, and so forth) cannot be used.
- Comma (,) and equals sign (=) should be avoided.

These may be accepted in instance names and subcircuit call names, but they are best avoided:

- Dollar sign (\$) cannot appear at the beginning of a name but may be used elsewhere.
- Control characters (^) are not recommended.
- Slash (/) may be used, but is not recommended because it has special meaning and normally splits up names (can be overridden with [LVS Spice Slash Is Space NO](#)).

The following table shows the differences in how slash (/) characters are handled in various user names by the circuit extractor and by the SPICE netlist parser. The indicated behavior for the SPICE netlist parser is the default (LVS Spice Slash Is Space YES).

Table 15-2. Handling of Slash Characters in Names

Name with Slashes	Circuit Extractor	SPICE Parser
Cell (translated as subcircuit)	Y	Y ¹
Primitive device	Y	Y ¹
Call to subcircuit	Y	N
Instance (as in X0, M0, etc.)	N/A	N
Net	Y ²	N

Table 15-2. Handling of Slash Characters in Names (cont.)

Name with Slashes	Circuit Extractor	SPICE Parser
Port	Y ²	N
Primitive device pin	Y	N
Device property	Y	Y
Device property value	Y	Y

1. Although slashes are allowed in subcircuit definition names for both cells and primitive devices, the calls to such names do not work by default.
2. Slashes are allowed by the circuit extractor for nets and ports; however, the names are not netlisted. The underlying numeric node IDs are.

Continuation Characters

Continuation characters allow a statement to span more than one line.

The continuation character (+) continues any line and must be the first non-white-space character of every line of a statement, other than the first line.

Continuation of CDL netlist lines is supported through the “*+” prefix. When this is the first character sequence of a line, it continues a preceding CDL line (and no others). For backward compatibility, the + character in the scope of a CDL statement behaves as *+.

Numeric Value Types

Numeric values can take a number of forms and can employ scale factors.

Numeric value types:

Type	Examples
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

The Calibre netlister formats precise integral values with no roundoff when the value of the integer fits in the range of +/-9007199254740990. Outside that range, scientific notation is used for integers. (However, during LVS comparison, all numeric property values, including precise integral values, are handled as floating-point numbers.)

Scale factors:

Abbreviation	Multiplication factor
T	1E12
G	1E9
MEG	1E6
K	1E3
M	1E-3
MIL	2.54E-5
U	1E-6
N	1E-9
P	1E-12
F	1E-15
A	1E-18

The following are syntactical considerations for common device types:

- **MOS transistors** — The width and length values can appear in any order anywhere in the device parameter set, as follows:

... <L=VAL> <W=VAL> ...

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 may contain optional scaling factors.

MOS elements may not have more than one parameter that begins with either L or W by default. Parameters that begin with these letters are interpreted as length and width parameters regardless of what follows the L or W, unless **LVS Spice Strict WL YES** is specified in the rule file. Without this statement, if the SPICE parser encounters more than one parameter that begins with L or W, the parser issues an error. To avoid this problem, either include LVS Spice Strict WL YES, or prefix any additional parameters with a letter other than L or W. For example, instead of LOD, use xLOD.

The L and W values may also appear without L= and W= prefixes, if the values appear as the first two tokens in the string. In this case, the two tokens are interpreted as length and width, in that order.

It is not required to start the device parameter set with a model name.

Parameter set 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 set should contain the capacitance value as the first token. The capacitance is a numeric value with units of farads and may contain an optional scaling factor. The scaling factor may be followed by an optional unit name, such as in 1PF. This optional unit name is ignored.

Parameter set examples:

```
1PF
10P IC=3V
```

- **Resistors** — The device parameter set should contain the resistance value as its first token. The resistance is a numeric value with units of ohms and may contain an optional scaling factor.

Parameter set examples:

```
100
1K TC=0.001,0.015
```

- **Diodes** — The first token of the device parameter set should 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. The area is specified in square meters. The perimeter is specified in meters. If a non-numeric value is found in the first or second position, the corresponding area or perimeter values are assumed to be missing. The device parameter line must not contain a model name.

```
<AREA <PERIM>> ...
```

Parameter set examples:

```
3.0P IC=0.2
3.0P 5.0U IC=0.2
```

Related Topics

[Arithmetic Expressions](#)

Arithmetic Expressions

Arithmetic expressions can appear anywhere a number can appear. Valid arithmetic operators are: +, -, *, and /. Standard operator precedence in the order of operations is observed, with parentheses () having the highest precedence. The sqrt() function is permitted and returns the

positive square root of its argument. 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))  
m2 1 2 3 4 p w='2+3' l='k*(3+2*(a+5))'  
r1 1 2 0.5 h='sqrt(l*l + w*w)'
```

Comments

Lines that start with an asterisk (*) or a dollar sign (\$) are treated as comments. The \$ may also be used to start comments later in a line, but this is not true for *.

Examples:

```
* This is a comment.  
$ This is a comment, too.  
C1 a 2 100p $ This is also a comment.
```

In most cases, everything following the \$ comment character is ignored. There are exceptions to this behavior. Built-in element statements including C, D, M, and R, each have standard parameters that are calculated by default during device recognition and these can be coded with comments. For example, \$W=<value> and \$L=<value> for M elements are properties coded as comments that are netlisted as W and L by default. The dollar signs in such properties do not cause everything following them to be ignored. However, this does not apply to user-defined devices, or to non-default parameters that appear with built-in elements.

The built-in devices and their default parameters are discussed under [Device](#) in the *SVRF Manual*.

Comment-Coded Extensions

The Calibre SPICE netlister uses two types of comment-coded extensions to extend the basic SPICE syntax: *. extensions and \$ extensions. The *. extensions add entire statements to the language. The \$ extensions add new fields to existing SPICE statements.

Examples:

```
*.CONNECT 10 20  
C1 1 2 100 $A=100 $P=40
```

Because both * (leading) 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 parameter mixing is valid in Calibre, but it may not be valid for SPICE simulation because anything following a \$ comment may be ignored by SPICE simulators. Therefore, mixing of comment-coded and regular parameters is not recommended.

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 a 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
```

String Parameters

The Calibre SPICE parser supports character-string parameters in addition to standard SPICE numerical parameters.

The general form for parameter assignment is this:

parnam=“*pval*”

where *parnam* is a parameter name and *pval* may be a literal value or expression of either numeric or string type. The string may be empty. By default, string values must be enclosed in double quotes. For example:

```
Z1 = "string"
```

Unquoted values may also be used, but you must explicitly allow this by specifying [LVS Spice Allow Unquoted Strings](#) YES in the rule file. Unquoted values must begin with a letter or underscore character (_).

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 preceding values enclosed in arbitrary number of parentheses, and any of the preceding values 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 parnam=pval as shown previously; they may not appear in implicit assignments where the parameter name is not explicitly specified.

Inline Parameters in Subcircuit Calls

By default, the SPICE parser does not process inline parameter references.

For example, if the SPICE file contains the following line:

```
X1 node1 node2 subname a=1 b=a
```

the instantiation of subcircuit subname has “a” with the value of 1 and “b” with the value of “a”. The parameter b does not get the value 1 from the b=a assignment in this case.

To allow inline parameter references, you can specify [LVS Spice Allow Inline Parameters YES](#) in your rule file. The order of parameter assignments in a given line is important. In the previous example, parameter b gets the value 1 from the b=a assignment when inline parameter references are allowed. However, if you have this:

```
X1 node1 node2 subname b=a a=1
```

then parameter b gets its value from the calling environment and not the a=1 assignment on the same line.

X Instantiated Devices

Some tools netlist various types of devices with an X letter prefix instead of the letter preferred by SPICE, for example, a MOSFET named XMOS1. The primitive devices perform scaling (using .OPTION SCALE) and other operations on their parameters, but these operations are lacking in the X-instantiated devices.

Assume these representations of MP devices in the netlist:

```
.OPTIONS SCALE .3e-06
...
mp1 a b c d p w=2 l=3
xp1 a b c d MP width=.2e-06 length=.1e-06
```

where the intent is to scale the mp1 instance, but not the xp1 instance.

By default, Calibre does not apply such scaling to X-instantiated devices. The use of this scaling is controlled through the [LVS Spice Scale X Parameters](#) specification statement in the rule file. Specifying MOS in this statement means scaling applies only to MOS devices; specifying YES means scaling applies to all built-in devices.

If parameter scaling is performed, it applies for the various X instantiated device types as in this table:

Table 15-3. X Instantiated Device Parameter Scaling

X Device Element	Parameter	Scaled Parameter
Bipolar Junction Transistor	L	L * SCALE
	W	W * SCALE
	PJ	PJ * SCALE
	AREA	AREA * SCALE * SCALE
	EA	EA * SCALE * SCALE
Note: Parameter EA overrides AREA, but both appear in the netlist for an X-instantiated device. A primitive BJT device would only have an EA parameter, if one appears.		
Capacitor	L	L * SCALE
	W	W * SCALE
	P	P * SCALE
	A	A * SCALE * SCALE

Table 15-3. X Instantiated Device Parameter Scaling (cont.)

X Device Element	Parameter	Scaled Parameter
Diode	L	L * SCALE
	W	W * SCALE
	PJ	PJ * SCALE
	AREA	AREA * SCALE * SCALE
JFET	L	L * SCALE
	W	W * SCALE
	AREA	AREA * SCALE * SCALE
MOSFET	L	L * SCALE
	W	W * SCALE
	PD	PD * SCALE
	PS	PS * SCALE
	AD	AD * SCALE * SCALE
	AS	AS * SCALE * SCALE
Resistor	L	L * SCALE
	W	W * SCALE

While these parameter conventions scale the values, none of these parameters appear as SPICE comments, unlike some of the equivalent parameters in the normal primitive device statements (specifically A and P for a Capacitor, L and W for a Diode, and L and W for a BJT).

X Instantiated MOSFET Devices

An X instantiated MOSFET is an X element in a SPICE netlist that serves a similar purpose as an M element.

Example:

```
.OPTIONS SCALE .3e-06
...
xp1 a b c d mp width=2 l=3
```

The scaling of “L” and “W” parameters for such devices is shown in [Table 15-3](#). The interpretation of what defines the L and W parameters for a MOS device is controlled by the [LVS Spice Strict WL](#) specification statement. The following options are available:

NO — For X-instantiated devices, only literal “L” and “W” parameters are scaled. For primitive MOS devices, any string that begins with L or W is interpreted as L or W, respectively, and are scaled as necessary. This is the default.

NONE — For both X-instantiated devices and primitive MOS devices, any string that begins with L or W is interpreted as L or W, respectively, and are scaled as necessary.

YES — For X-instantiated devices, the behavior is the same as NO. For primitive MOS devices, only literal “L” and “W” parameters are interpreted as L and W and are scaled as necessary.

Dollar Signs in Cell Names

The Calibre SPICE netlister lists 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.

\$D Parameters

The SPICE 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
```

\$D=0 means the first Device statement encountered in the rule file was used to classify the device.

This information is intended for use by certain downstream applications, for example, Calibre xRC. It is not used in Calibre nmLVS or Calibre PERC. 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.

Cell Statistics

.SUBCKT statements in an extracted layout netlist are each followed with a comment line containing statistics about the respective layout cell.

For example:

```
.SUBCKT PVDD2 1 2
** N=452 EP=2 IP=516 FDC=249
```

N is the number of nets in the cell. EP is the number of external pins in the cell (pins of the cell). IP is the number of internal pins in the cell (placement pins in the cell). Note that N and IP may not be identical to what is actually present in the netlist because not all layout nets and placement pins are represented in the netlist. For example, the netlist normally does not contain floating nets or placements of cells that have no devices.

FDC is the flat device count in the cell. This is the number of all primitive devices in the cell, including the sub-hierarchy of the cell, counted flat. In this context, primitive devices are objects formed with rule file [Device](#) operations. [LVS Box](#) cells are treated as normal cells. The FDC number is an indicator 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.

SPICE Syntax Checking

The Calibre SPICE parser reports errors or warnings for various incorrect usages in a SPICE netlist. For warnings that could be repeated for multiple instances of an incorrect usage, the parser stops reporting the condition after 256 instances are found. Any error or warning conditions should be understood and fixed as necessary.

Ambiguous Element Checks

In certain rare cases, SPICE elements can have ambiguous syntax.

Consider the following BJT element statement:

```
Q1 a b c d e
```

Here, the syntax is ambiguous. The parser attempts to interpret this as a three-pin device first, and failing that, it attempts to interpret it as a four-pin device. In any case, a, b, and c are taken as pin net names. However, there is a three-pin interpretation, in which “d” is the model name and “e” is a parameter reference for the AREA parameter. There is also a four-pin interpretation, in which “d” is the substrate net name and “e” is the model name. Given this statement, the parser selects the three-pin interpretation only if “e” is the name of a numeric parameter in the calling environment suitable for the AREA parameter. Otherwise, the four-pin interpretation is used. Suppose the previous element statement is found in a subcircuit definition:

```
.SUBCKT sub a b c d e
Q1 a b c d e
.ENDS
```

and this subcircuit is called as follows:

```
.SUBCKT top 1 2 3 4 5
X1 1 2 3 4 5 sub e=2.72
X2 1 2 3 4 5 sub
.ENDS
```

Since the first call is parameterized, sub will be flattened by the parser. In X1, “e” is defined as a numeric parameter. Hence, the chosen interpretation of Q1 would be the three-pin interpretation. In X2, “e” is not defined as a parameter. Therefore, in X2, Q1 would be interpreted as a four-pin BJT with “e” as its model name.

The same subcircuit should not have differing interpretations in the same netlist, so the parser detects such ambiguities and produces an error. Such errors can be resolved by ensuring that the element has the same interpretation in every call to the subcircuit that contains it.

Consider the following BJT element statement:

```
Q1 a b c d e f g
```

If this is taken as a four-pin device, a, b, c, and d are net names, “e” is the model name, and “f” is the AREA parameter. Since the BJT element statement defines only one positional parameter, AREA, there is no meaningful way to interpret “g”. Such statements generate syntax errors. This avoids having an element with differing interpretations depending on context.

Ambiguities detected by the SPICE parser produce errors like this:

Ambiguous interpretation of {BJT | JFET | MOSFET} element (3-pin vs. 4-pin) in file "<file>" at line <line> (path <path>)

Ambiguous interpretation of {capacitor | inductor | resistor} element (model versus expression) in file "<file>" at line <line> (path <path>)

Control Statements

Calibre tools observe certain control statements in the SPICE netlist. Some of these statements are unique to Calibre SPICE. Others may be recognized by third-party tools. Control statements used in other SPICE dialects than Calibre, and that are ignored, are indicated as such.

*.BUSDELIMITER	648
*.CALIBRE	651
*.CAPA	652
*.CONNECT	653
*.DIODE	656
.ENDL	657
.ENDS or .EOM	658
*.EQUIV	659
.GLOBAL	661
.INCLUDE	663
*.LDD	665
.LIB	666
*.MEGA	668
.OPTION SCALE	670
.PARAM	672
*.SEEDPROM	673
*.STRIP_TYPE_CHAR	674
*.XPINS	675
Ignored Control Statements	676

*.BUSDELIMITER

The *.BUSDELIMITER SPICE statement specifies the character that identifies 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 Verilog structural netlist. Only port names for called SPICE subcircuits are affected by the *.BUSDELIMITER statement.

Syntax

`*.BUSDELIMITER symbol [IMPLIED | DECREASING | INCREASING]`

Parameters

- *symbol*

Required replaceable character: [, {, <, or (.

- IMPLIED

Optional parameter that causes specified ports in a .SUBCKT that are not arranged in a particular way to take the range of the last port definition. This is the default.

- DECREASING

Optional parameter that arranges the port numerically from high to low.

- INCREASING

Optional parameter that arranges the port numerically from low to high.

Description

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.

See “[*.BUSDELIMITER Statement Handling](#)” on page 756 for related details.

Examples

SPICE Library

```
*.BUSDELIMITER <
.SUBCKT A I1<3> I1<2> I1<1> I1<0>
...
.ENDS

*.BUSDELIMITER {
.SUBCKT B I2{3} I2{2} I2{1} I2{0}
...
.ENDS
```

Verilog Netlist

```
module TOP;
  wire [3:0] w1;
  wire [3:0] w2;
  A A1 (.I1(w1));
  B B1 (.I2(w2));
endmodule
```

When a SPICE library is specified with v2lvs::load_spice -range_mode, 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. Using -range_mode creates the following Verilog interface interpretation of the previous SPICE library:

```
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
```

DECREASING and INCREASING Parameters

The following SPICE library:

```
*.BUSDELIMITER < DECREASING
.SUBCKT TOP1 I1<2> I1<3> I1<1> I1<0>
.ENDS
```

is equivalent to the following Verilog module:

```
module TOP1 ( I1 );
  inout [3:0] I1;
endmodule
```

The following SPICE library:

```
*.BUSDELIMITER < INCREASING
.SUBCKT TOP2 I1<2> I1<3> I1<1> I1<0>
.ENDS
```

is equivalent to the following Verilog module:

```
module TOP2 ( I1 );
  inout [0:3] I1;
endmodule
```

The v2lvs::load_spice -detect_bus_delimiter option finds the bus delimiter character automatically in a SPICE library subcircuit and causes a *.BUSDELIMITER statement that covers the included SPICE netlist in the output. The v2lvs::combine_interface_info can also be useful in such applications.

*.CALIBRE

Indicates a condition in a subcircuit that is tracked by Calibre. The specific condition is indicated by additional keywords.

Syntax

```
*.CALIBRE {{ABORT_INFO SUPPLY_SHORT} | CLONE_CELL |
{ISOLATED NETS: net_ID [net_ID ...]} | {WARNING {BADDEV | PORTS} message} |
{v2lvs warning: message}}
```

Parameters

- ABORT_INFO SUPPLY SHORT

Indicates that circuit extraction detected a supply net short when [LVS Abort On Supply Short YES](#) was specified. The comment in a netlist causes LVS comparison to stop when LVS Abort On Supply Short YES is specified.

- CLONE_CELL

Indicates the subcircuit was cloned by [Layout Clone Rotated Placements YES](#) or [Layout Clone Transformed Placements YES](#). It triggers evaluation of the cell as a possible LVS Box cell or hcell in the LVS comparison run. In this case, its name is evaluated against the standard Calibre clone cell renaming convention, and, if its name matches the clone cell naming conventions used by Calibre, then the cell functions as the specified hcell or box cell.

- ISOLATED NETS: *net_ID*

Indicates at least one isolated layout net was found during circuit extraction in a physical cell. The *net_ID* is an identifier that corresponds to the net. Isolated layout nets are not connected to a port or to an instance pin. This information is traced by LVS netlist-to-netlist comparison to identify isolated physical nets.

- WARNING {BADDEV | PORTS} *message*

Indicates a discrepancy during circuit extraction. The *message* contains relevant information to the type of warning. Related details appear in the circuit extraction report.

BADDEV —Indicates a bad device.

PORTS — Indicates an unattached port. This can appear when LVS Report Option UP is specified in the rules.

- v2lvs warning: *message*

Indicates a discrepancy during a V2LVS run when generating the netlist. The details are in the *message*.

*.CAPA

*.CAPA instructs the tool to ignore capacitor (C) elements in the netlist. It is coded as a comment and has no arguments.

Syntax

*.CAPA

Parameters

None.

*.CONNECT

Connects nets together. The .CONNECT statement takes two or more node names as arguments and is coded as a comment.

Syntax

`*.CONNECT node1 node2 [...]`

`.CONNECT node1 node2 [...]`

`*.J node1 node2 [...]`

Parameters

- *node1*

Node name. Hierarchical paths are delimited by commas.

- *node2*

Node name. Hierarchical paths are delimited by commas.

Description

The Calibre SPICE reader 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. (The *.EQUIV statement specifies correlation only between different source or layout names; it does not connect nets together.)

Shorts propagate in both hierarchical and flat execution. 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). 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, but they do not apply to non-global nets outside of the subcircuit definition in which these statements occur unless a hierarchical path connection is specified.

Hierarchical Paths in *.CONNECT

Hierarchical paths in *.CONNECT and its aliases allow the net or pin names specified in the statements to refer down to nets in subcircuit(s) instantiated by the current subcircuit rather than just nets or pins in the current subcircuit.

Hierarchical paths are delimited using a comma (,) character. (The slash character “/” is not used since this character is commonly used in certain generated netlists using LVS Spice Slash Is Space YES, and its meaning would then be ambiguous in that situation.)

Using a hierarchical path has the same effect as creating additional pins on the referenced subcircuit’s definition and all interposed subcircuits definitions and connecting them appropriately. For example, consider this design:

```
.subckt sub0 e f g h
*.connect h x1,h
m1 c d e h mn $$ connected to net h in sub1 by *.connect
x1 e f g sub1
x2 e f g sub1
.ends

.subckt sub1 e f g
m2 e f g h mn $$ connected to net h in sub0
.ends
```

The *.connect in sub0 connects net h in that subcircuit to net h in sub1 instance x1, also in the scope of sub0. Instance x2 in sub0 is unaffected. This is similar to the following netlist that does not have the *.CONNECT statement:

```
.subckt sub0 e f g h
m1 c d e h mn
x1 e f g h sub1
x2 e f g sub1
.ends

.subckt sub1 e f g h
m2 e f g h mn
.ends
```

Examples

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
```

*.DIODE

*.DIODE instructs the SPICE parser to ignore diode (D) elements in the netlist. It is coded as a comment and has no arguments.

Syntax

*.DIODE

Parameters

None.

.ENDL

The .ENDL statement indicates the end of a .LIB section. Any statements after a .ENDL are ignored, but a valid comment may appear after .ENDL.

Syntax

.ENDL [*comment*]

Parameters

- *comment*

Optional text comment.

.ENDS or .EOM

Indicates the end of a subcircuit (or module). This statement must be the last one for any subcircuit definition. The statement has two forms.

Syntax

.ENDS [*subname*]
.EOM [*subname*]

Parameters

- *subname*
An optional subcircuit name.

Examples

```
.SUBCKT opamp
$$ components ...
.ENDS opamp
```

*.EQUIV

Translates existing layout or source object names to new names.

Syntax

```
*.EQUIV new_name=old_name [new_name=old_name ...]
```

Parameters

- *new_name*
A required new name of the model or node; replaces the old name.
- *old_name*
A required old name of the model or node.

Description

This statement 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*. The SPICE parser 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.
- Node names equal to *old_name* are replaced with *new_name*. The tool 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.

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. Calibre nmLVS issues a warning and does not use the name as an initial correspondence point.

Examples

In the following example, the netlist parser replaces model names TP and TN by P and N, respectively. This allows the tool 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
```

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
```

.GLOBAL

Specifies nodes shared globally by all subcircuits. The form *.GLOBAL is coded as a comment and is equivalent to .GLOBAL.

Syntax

```
.GLOBAL node1 ...
*.GLOBAL node1 ...
```

Parameters

- *nodeN*

A node name defined as an external reference for all subcircuits in the netlist.

Description

The .GLOBAL statement provides a convenient means of defining power supplies and clocks in the netlist.

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. It does not matter where .GLOBAL appears in a netlist, the behavior is always the same.

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 (;).

Subcircuit Pin Preferences

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. For this excerpt:

```
.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 nets A and B of subcircuit call X1.

You can specify that subcircuit pin assignments override global signals throughout subcircuits and their sub-hierarchies by using the LVS Spice Override Globals statement.

Examples

```
$$ global nets. VDD and VSS are treated as ports.  
.GLOBAL VDD VSS
```

.INCLUDE

Causes the named file to be included in the netlist in the precise location the .INCLUDE statement appears.

Syntax

.INCLUDE *pathname*

.INC *pathname*

Parameters

- *pathname*

Specifies a file path. You can enclose pathname in single or double quotation marks. Multiple levels of inclusion are allowed. Pathnames may include “~” characters and environment variables.

Description

When a filename is given in a .INCLUDE statement, it is searched for in this order:

1. As an absolute path.
2. As a relative path from the current directory, that is, the one from which the tool 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. Thus, the library can be relocated without internal changes and even continue to operate if different users see it through different nfs mount points. On the other hand, you can override specific library files with files in the current directory because it is searched first.

It is preferable to reorder the file so the .INCLUDE statement appears near the top of the file.

You can specify environment variables as components of filename (or pathname) parameters in .INCLUDE statements. Environment variables are signified by preceding them with the dollar sign (\$) character. They are evaluated and substituted into the filename string. For example, these are all valid:

```
.INCLUDE $file1
.INCLUDE "$file2"
.INCLUDE "$ENV1/dir2/$ENV2/file"
```

If \$ENV1 is defined as “/dir1” and \$ENV2 is defined as “dir3”, then the last statement in the example is evaluated as:

```
.INCLUDE "/dir1/dir2/dir3/file"
```

To make the notion of filename component precise, a filename 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.

An error results when an environment variable required by filename evaluation is undefined, or is defined with a null value, except as described next.

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 \$file, then the statement:

```
.INCLUDE $file
```

includes that file, regardless of the value of the \$file environment variable, and regardless of whether \$file is defined.

If you use a .INCLUDE statement in a compressed SPICE file (ending in .Z or .gz), it is best to place the .INCLUDE statement near the beginning of the file. This is because the compressed file must always be read from the beginning when returning from processing the included file or the referenced library. For example, it is not advisable to compress files that look like this:

```
$ Long SPICE file:
...
many lines of SPICE ...
...
$ Followed by .INCLUDE:
.INCLUDE somefile
$ Reorder this netlist or you could experience performance issues.
```

Examples

```
$$ params file is read according to location precedence
.INCLUDE params
C1 1 2 10P
$$ file read from absolute path and inserted at this location
.INC /net/user1/circuitfile
```

*.LDD

Controls how the Calibre SPICE parser processes \$LDD designators in M elements in SPICE.

Syntax

*.LDD

Parameters

None.

Description

When you specify the LVS Spice Conditional LDD YES specification statement in the rule file, the SPICE reader processes \$LDD designators in M elements only if a *.LDD statement is present somewhere in the netlist. The parser 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.

Examples

The following example shows how to use the *.LDD statement in SPICE syntax:

```
* .LDD
M2 4 5 6 7 P $LDD [PPP]
```

.LIB

Defines or references a SPICE library. It comes in two forms.

Syntax

Definition Form

.LIB *section_name*

Reference Form

.LIB *lib_name section_name*

Parameters

- *section_name*

A required single-word name of a SPICE library section.

- *lib_name*

A pathname of a SPICE library file. This argument is used only in the reference form of the statement. Environment variables and ~ characters are permitted in a path.

Description

The definition form has only a *section_name* parameter. This form defines a library section with the given name, which includes all lines of the current source file from the .LIB statement until the next occurrence of a .ENDL statement, or the end of the file. Note that nesting of library 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 form causes the file *lib_name* to be opened and the contents of the *section_name* within that file to be included in the precise location .LIB is called. 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_name*, the contents of all sections are concatenated and included. If file *lib_name* does not contain a section with *section_name*, the SPICE parser 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 *lib_name* is given in a .LIB statement, it is searched for in this order:

1. As an absolute path.
2. As a relative path from the current directory, that is, the one from which the tool 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. Thus, the library can be relocated without internal changes and even continue to operate if different users see it through different nfs mount points. You can override specific library files with files in the current directory, because it is searched first.

If you use a .LIB statement in a compressed SPICE file (ending in .Z or .gz), it is best to place the .LIB statement near the beginning of the file. This is because the compressed file must always be read from the beginning when returning from processing the included file or the referenced library. For example, it is not advisable to compress files that look like this:

```
$ Long SPICE file:  
...  
many lines of SPICE ...  
...  
$ Followed by .LIB:  
.LIB somefile  
$ Reorder this netlist or you could experience performance issues.
```

It is preferable to reorder the file so the .LIB statement appears near the top of the file.

*.MEGA

Causes the SPICE parser to interpret uppercase “M” scale factors as “Mega” (1E+6) instead of the usual “milli” (1E-3). Lowercase “m” scale factors are interpreted as milli. The default in the absence of *.MEGA statements is to interpret both uppercase M and lowercase m as milli.

Syntax

*.MEGA.

Parameters

None.

Description

Generally, the *.MEGA statement has global scope, and it applies to numeric expressions that appear anywhere in the netlist. 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 string parameters 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. (However, when LVS Spice Redefine Param YES is specified, any .PARAM statements contained within subcircuit definitions have local scope, and *.MEGA applies to them regardless of their relative position in the netlist.)

Examples

```
.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 this example, *.MEGA applies to the .PARAM BB statement, but not to the .OPTIONS SCALE statement nor 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
```

.OPTION SCALE

Specifies the scale factor for length measurements of device properties. You can specify any option, but Calibre ignores all options except SCALE.

Syntax

.OPTIONS *option* ...
.OPTION *option* ...
.PC *option* ...
.CONTROL *option* ...

Parameters

- *option*

An option name and an optional value assignment. The general form of a value assignment is *name*=*value*. The default *value* is 1 when not explicitly assigned. The only option name recognized by Calibre is SCALE along with a numeric scale factor.

Description

The SCALE factor applies to all units in the netlist. The SCALE factor can be changed within the netlist, and the last definition that is read in the current scope is used. Options specified within subcircuit definitions are used locally in those subcircuits only; those specified outside of subcircuit definitions apply globally.

The option:

SCALE=*value*

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

The SPICE parser multiplies one-dimensional length 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, if scale is 1E-6 (length units are microns); areas are then in square microns.

Examples

In this example, M1 and M2 have equal sizes:

```
.OPTIONS SCALE=1E-6          $ Sets scale to 1E-6.  
.SUBCKT AAA  
M1 1 2 3 4 P W=4 L=1 AS=4 AD=4  $ No scale factors, values in um and um^2  
.ENDS  
.OPTIONS SCALE=1          $ Sets scale to 1.  
.SUBCKT BBB  
M2 1 2 3 4 P W=4U L=1U AS=4E-12 AD=4E-12  $ With scale factors, values  
$ in um and um^2  
.ENDS  
X1 AAA  
X2 BBB
```

.PARAM

Defines parameter names and values. The values override those set in a .SUBCKT or .MACRO statements, or in subcircuit calls.

Syntax

.PARAM *parname* = *pval* ...

Parameters

- *parname*
Parameter name assigned to numeric or string parameter value. Each name must begin with a letter followed by any number of allowed SPICE characters.
- *pval*
Value assigned to *parname*.

Description

When you use the parameter name in a subcircuit description, the specified *pval* is automatically substituted. This type of value is referred to as a global parameter.

[“String Parameters”](#) on page 640 discusses parameters defined as strings.

See [LVS Spice Redefine Param](#) in the *SVRF Manual* as a related specification statement.

Examples

```
.PARAM width=1U pp="abc"
X1 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 netlist, M1 has the following values:

```
w  = 1U (from the .PARAM statement)
l  = 6U (from the subcircuit call)
ad = 30P (from the .SUBCKT definition)
as = 40P (from the .SUBCKT definition)
rr = "abc" (from the .PARAM statement)
```

Related Topics

[Arithmetic Expressions](#)

*.SEEDPROM

Indicates that seed promotion occurred during hierarchical device recognition in the layout cell represented by the extracted subcircuit. This statement is coded as a comment and takes no arguments.

Syntax

*.SEEDPROM

Parameters

None.

Description

The SPICE netlister places *.SEEDPROM statements in the extracted layout netlist to communicate seed promotion events to the LVS circuit comparison module. Seed promotion occurs when hierarchical device recognition is unable to process devices completely within a cell, so device recognition promotes those devices out of the cell and places them at a higher level of hierarchy. This statement has no meaning when it appears outside of subcircuit definitions. When placed inside an otherwise empty subcircuit, *.SEEDPROM renders the subcircuit non-primitive.

The statement is ignored when it appears inside [LVS Box](#) cells.

Hierarchical circuit comparison can be instructed to expand cells containing *.SEEDPROM statements through the [LVS Expand Seed Promotions YES](#) specification statement.

***.STRIP_TYPE_CHAR**

Instructs the SPICE parser to strip the leading character from all names read in after deciding on the device type, but before any further processing. This statement is useful in certain Calibre PERC flows.

Syntax

`*.STRIP_TYPE_CHAR`

Parameters

None.

Description

In some cases when flattening of the netlist is required, the Query Server LAYOUT NETLIST WRITE command writes a `*.STRIP_TYPE_CHAR` directive to the netlist, which instructs the parser to strip the leading character from all names read in after deciding on the device type, but before any further processing. The `*.STRIP_TYPE_CHAR` directive applies to all SPICE read in for the current circuit, including SPICE read in previously during the run.

As an example, if `*.STRIP_TYPE_CHAR` is present, an ROUT device would be treated as a resistor with the name OUT.

*.XPINS

Supports the E2LVS EDIF converter and similar programs. The keyword XPINS stands for explicit pin connections.

Syntax

*.XPINS is coded as a comment and takes no arguments.

Parameters

None.

Description

This 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.

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
*.XPINS
C1 A B
.ENDS
```

When the SPICE parser 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, 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
```

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 ==. The == is not part of the pin name and merely indicates the type of reference.

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.

Examples

```
.SUBCKT SS1 A B      $ Subcircuit SS1 with pins A and B.  
.XPINS               $ Subcircuit has explicit pin connections.  
.J A ==A              $ Join net A with pin A of SS1.  
.J B ==B              $ Join net B with pin B of SS1.  
.J 1 ==B              $ Join net 1 with pin B of SS1.  
.J ==A ==B             $ Join pins A and B of SS1.  
C1 A B                $ A capacitor hooked to nets A and B.  
.ENDS
```

Ignored Control Statements

Certain control statements are silently ignored by the SPICE parser.

.AC	.IC	.OP	.PZ	.TRAN
.DC	.MEASURE	*.PININFO	.SAMPLE	.UNPROTECT
.DCVOLT	.MODEL	.PLOT	.SENS	.WIDTH
.DISTO	.NET	.PRINT	.TEMP	
.FOUR	.NODESET	.PROBE	.TF	
.GRAPH	.NOISE	.PROTECT	.TITLE	

These control statements are ignored by the SPICE parser with warning messages:

.ALTER	.DATA	.DEL	.END
--------	-------	------	------

Element Statements

SPICE uses element statements to represent circuit components. Most of these components are primitive devices. A subcircuit definition frequently specifies a more complex structure.

For each element discussed in this section, there is a syntactical representation followed by a table describing the individual elements of the statement. Strings shown in bold are the minimum arguments for a complete element string. General syntax rules are shown under “[SPICE Syntax Conventions](#)” on page 634.

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.

Capacitor Element	677
Junction Diode Element	680
JFET Element	681
Inductor Element	683
MOSFET Element	685
BJT Element	690
Resistor Element	692
Voltage Source Element	694
Subcircuit Definitions	695
Subcircuit Calls	699
Floating Pins in Subcircuit Calls	704
Block-Level Chip Assembly Using Scoped SPICE Subcircuits	705
Multiplier (m) Parameters	707
Eldo Format Y Element	708

Capacitor Element

The following strings define a capacitor.

```
Cxxx n1 n2 <mname> c <tc1 <tc2 <scale <ic <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>

Cxxx n1 n2 <mname> c <TC=tc1 <tc2 <scale>>> <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>
```

```
Cxxx n1 n2 <mname> C=c <TC1=tcl> <TC2=tcl> <SCALE=scale>
+ <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>
```

The mname and c parameters in the first two syntaxes may be interchanged. If one of these two strings cannot be evaluated numerically (for example, if it is not a numeric value or a previously defined parameter) then it is assumed to be the mname parameter.

The following are some examples:

```
C1 1 2 10P
C3 n1 n2 10P M=4 W=10U L=20U AAA=5 BBB="zz" $ [mc] $comment
C4 n1 n2 10P 1 1 4 $SUB=3 $.MODEL=mc $A=10P $P=40U
```

To define capacitor devices with more than one substrate pin, define a primitive subcircuit as described in the section “[.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO](#)” on page 696.

Table 15-4. Capacitor Element

Argument Name	Description	Trace Property Name
Cxxx	Capacitor element name. Must begin with a C followed by any number of allowed characters.	
n1	Positive terminal node name. String of any number of allowed characters.	
n2	Negative terminal node name. String of any number of allowed characters.	
mname	Optional model name. String of any number of allowed characters. May not be identical to the argument c. ¹	
c	Capacitance in farads. May not be identical to mname.	c
tcl	Ignored.	
tc2	Ignored.	
scale	Optional scale factor. Multiplies capacitance (c).	
ic	Optional initial voltage across the capacitor in volts.	ic
m	Optional multiplier factor used to simulate multiple parallel capacitors. Property m is evaluated immediately and not handed down through parameter passing. 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.	m
l	Optional length.	l

Table 15-4. Capacitor Element (cont.)

Argument Name	Description	Trace Property Name
w	Optional width.	w
a	Optional area.	a
p	Optional perimeter.	p
parnam= pval	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 allowed characters. You can specify any number of parnam=pval pairs. See “ String Parameters ” on page 640 for complete details on string parameters.	parnam
ns	Optional substrate terminal node name coded as a comment. String of any number of allowed characters.	
\$[mname] or \$.MODEL= mname	Optional model name, coded as a comment. String of any number of allowed characters. Overrides the regular optional mname parameter	
\$A=a	Optional area. Coded as comment. Overrides A, if present.	a
\$P=p	Optional perimeter. Coded as comment. Overrides P, if present.	p
\$X=x \$Y=y	Optional x,y coordinates coded as comments. Integer numbers in database units. Used in hierarchical applications only.	
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.	

1. If you have mname and “C” parameters of the same name, this was allowed in version 2010.2 and earlier. To restore that behavior, set the CALIBRE_OLD_PASSIVE_MODELS environment variable to any non-null value.

LVS component type: C

LVS component subtype: mname

LVS pin names: pos (positive), neg (negative), sub (optional substrate)

The Dracula CDL statement [*.CAPA](#) is supported. It instructs the tool to ignore capacitor elements in the netlist.

Junction Diode Element

The following strings define a diode.

```
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>
```

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 define diode devices with more than one substrate pin, define a primitive subcircuit as described in the section “[.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO](#)” on page 696.

Table 15-5. Junction Diode Element

Argument Name	Description	Trace Property Name
Dxxx	Diode element name. Must begin with a D followed by any number of allowed characters.	
nplus	Positive (anode) terminal node name. String of any number of allowed characters.	
nminus	Negative (cathode) terminal node name. String of any number of allowed characters.	
mname	Model name. String of any number of allowed characters.	
a	Optional diode area.	a
pj	Optional periphery of junction.	p
m	Optional multiplier factor to simulate multiple diodes. Property m is evaluated immediately and not handed down through parameter passing. 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.	m
OFF	Ignored.	

Table 15-5. Junction Diode Element (cont.)

Argument Name	Description	Trace Property Name
parnam= pval	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 allowed characters. You can specify any number of parnam=pval pairs. See “ String Parameters ” on page 640 for complete details on string parameters.	parnam
ns	Optional substrate terminal node name coded as a comment. String of any number of allowed characters.	
\$X=x \$Y=y	Optional x,y coordinates coded as comments. Integer numbers in database units. Used in hierarchical applications only.	
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.	

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 the SPICE parser to ignore diode elements in the netlist.

JFET Element

The following strings define a junction gate field effect transistor.

```
Jxxx nd ng ns <nb> mname <AREA=a> <W=w> <L=l> <M=m> <OFF>
+ <IC=vds, vgs> <parnam=pval> ... <$SUB=nb> <$X=x> <$Y=y> <$D=d>

Jxxx nd ng ns <nb> mname <a> <w> <l> <M=m> <OFF> <IC=vds, vgs>
+ <parnam=pval> ... <$SUB=nb> <$X=x> <$Y=y> <$D=d>
```

The following are some examples:

```
J1 10 24 13 JM1
Jabc netd netg nets jmod AREA=8P W=3U L=7U M=2 AAA=5 BBB="zz"
J234 netd netg nets jmod 8P 3U 7U M=2 $SUB=netb
```

To define JFET devices with more than one substrate pin, define a primitive subcircuit as described in the section “[.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO](#)” on page 696.

Table 15-6. JFET Element

Argument Name	Description	Trace Property Name
Jxxx	JFET element name. Must begin with a J followed by any number of allowed characters.	
nd	Drain terminal node name. String of any number of allowed characters.	
ng	Gate terminal node name. String of any number of allowed characters.	
ns	Source terminal node name. String of any number of allowed characters.	
nb	Optional bulk connection node name. String of any number of allowed characters.	
mname	Model name. String of any number of allowed characters.	
a	Optional area.	a
w	Optional gate width.	w
l	Optional gate length.	l
m	Optional multiplier factor to simulate multiple JFETs. Property m is evaluated immediately and not handed down through parameter passing. 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
OFF	Ignored.	
vds	Ignored.	
vgs	Ignored.	
parnam= pval	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 allowed characters. You can specify any number of parnam=pval pairs. See “ String Parameters ” on page 640 for complete detail on string parameters.	parnam
\$SUB=nb	Optional bulk terminal node name coded as a comment. String of any number of allowed characters. Overrides the regular fourth terminal field if both are present.	
\$X=x \$Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in hierarchical applications only.	

Table 15-6. JFET Element (cont.)

Argument Name	Description	Trace Property Name
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.	

LVS component type: J

LVS component subtype: mname

LVS pin names: d (drain), g (gate), s (source), b (optional bulk).

Inductor Element

The following strings define an inductor.

```
Lxxx n1 n2 <mname> <l <tc1 <tc2>>> <SCALE=scale> <M=m> <R=r>
+ <parnam=pval> ... <$SUB=ns> <$[mname] | $.MODEL=mname>
+ <$X=x> <$Y=y> <$D=d>
```

```
Lxxx n1 n2 <L=l <mname> <tc1 <=tc2>>> <SCALE=scale> <M=m> <R=r>
+ <parnam=pval> ... <$SUB=ns> <$[mname] | $.MODEL=mname>
+ <$X=x> <$Y=y> <$D=d>
```

The mname and “l” parameters in the first syntax may be interchanged. If one of these two strings cannot be evaluated numerically (for example, if it is not a numeric value or a previously defined parameter), then it is assumed to be the mname parameter.

As with all other Calibre devices, a model specified as \$[mname] or as \$.MODEL=mname overrides the model specified earlier in the line. A model name is distinguished from a ‘tc1’ value by a failure to evaluate the name at that position as a numeric value because of an undefined parameter. Therefore, you should avoid name conflicts between your parameters and models. Failure to avoid such conflicts may cause a false recognition of an inductor that should cause an undefined parameter error for tc1, and, instead, make that parameter the name of an inductor model.

The following are some examples:

```
L1 1 2 100
12 n1 n2 100 10 20 SCALE=2 IC=30 M=4 DTEMP=40 R=200 F=300
+ BAR="zzz" $SUB=n3 $[lmod]
L3 na nb L=100 TC1=10 TC2=20 SCALE=2 IC=30 M=4 DTEMP=40 R=200
+ F=300 $SUB=nc $.MODEL=lmod $X=1000 $Y=2000
```

To define inductor devices with more than one substrate pin, define a primitive subcircuit as described in the section “[.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO](#)” on page 696.

Table 15-7. Inductor Element

Argument Name	Description	Trace Property Name
Lxxx	Inductor element name. Must begin with an L followed by any number of allowed characters.	
n1	Positive terminal node name. String of any number of allowed characters.	
n2	Negative terminal node name. String of any number of allowed characters.	
mname	Optional model name. String of any number of allowed characters. May not be identical to argument l. ¹	
l	Optional inductance in henrys. May not be identical to mname.	l
tc1	Ignored.	
tc2	Ignored.	
scale	Optional scale factor. Multiplies inductance (l) and resistance (r) by the factor.	
m	Optional multiplier factor used to simulate multiple parallel inductors. Property m is evaluated immediately and not handed down through parameter passing. 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.	m
r	Optional resistance in ohms.	r
parnam= pval	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 allowed characters. You can specify any number of parnam=pval pairs. See “ String Parameters ” on page 640 for complete details on string parameters.	parnam
ns	Optional substrate terminal node name coded as a comment. String of any number of allowed characters.	
\$[mname] or \$.MODEL= mname	Optional model name, coded as a comment. String of any number of allowed characters.	

Table 15-7. Inductor Element (cont.)

Argument Name	Description	Trace Property Name
\$X=x \$Y=y	Optional x,y coordinates coded as comments. Integer numbers in database units. Used in hierarchical applications only.	
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.	

1. If you have mname and “l” parameters of the same name, this was allowed before 2010.3. To restore the old behavior, set the CALIBRE_OLD_PASSIVE_MODELS environment variable to a non-null value.

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 statements 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.

MOSFET Element

The following strings define a metal-oxide-semiconductor field effect transistor.

```
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>
+ <nrd> <nrs> <rdc> <rsc> <OFF> <IC=vds, vgs, vbs> <M=m>
+ <parnam=pval> ... <$LDD<[type]>> <$X=x> <$Y=y> <$D=d>
```

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
```

To define MOS devices with non-standard numbers of pins, define a primitive subcircuit as described in the section “[.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO](#)” on page 696.

Table 15-8. MOSFET Element

Argument Name	Description	Trace Property Name
Mxxx	MOSFET element name. Must begin with an M followed by any number of allowed characters.	
nd	Drain terminal node name. String of any number of allowed characters.	
ng	Gate terminal node name. String of any number of allowed characters.	
ns	Source terminal node name. String of any number of allowed characters.	
nb	Optional bulk terminal node name. String of any number of allowed characters.	
mname	Model name. String of any number of allowed characters.	
l	Optional channel length.	l
w	Optional channel width.	w
ad	Optional drain area.	ad
as	Optional source area.	as
pd	Optional drain perimeter.	pd
ps	Optional source perimeter.	ps
nrd	Optional number of squares of drain diffusion.	nrd
nrs	Optional number of squares of source diffusion.	nrs
rdc	Optional additional drain resistance due to contact resistance.	rdc
rsc	Optional additional source resistance due to contact resistance.	rsc
OFF	Ignored.	
vds	Ignored.	
vgs	Ignored.	

Table 15-8. MOSFET Element (cont.)

Argument Name	Description	Trace Property Name
vbs	Ignored.	
m	Optional multiplier factor to simulate multiple MOSFETs. Property m is evaluated immediately and not handed down through parameter passing. Multiplies w, ad, as, pd, and ps. If LVS Spice Replicate Devices YES is specified in the rule file, then m parallel copies of the MOSFET are created instead, and m for each copy is set to 1. Defaults to 1 if not specified. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement.	m
parnam=pval	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 allowed characters. You can specify any number of parnam=pval pairs. See “ String Parameters ” on page 640 for complete details on string parameters.	parnam
\$LDD	Optional LDD device designator, coded as a comment.	
(none)	Optional implied W and L values. Computed if LVS Spice Implied MOS Area is specified in the rule file.	
type	Optional LDD device type. String of any number of allowed characters. Allowed (but not required) only in conjunction with \$LDD. If specified, replaces mname.	
\$X=x \$Y=y	Optional x,y coordinates coded as comments. Integer numbers in database units. Used in hierarchical applications only.	
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.	

Calibre interprets the MOSFET element as a 4-pin device when the syntax is ambiguous and can be interpreted as either a 3-pin or a 4-pin device. MOS devices with more than one substrate pin are considered user-defined devices.

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 384). Nevertheless, in the hierarchical layout netlist they are represented, when possible, with SPICE MOS elements, not primitive subcircuit calls.

For example, the following in a rule file:

```
DEVICE m(h) gate gate(g) sd(s) sd(d) bulk(b) [  
    property w, l ...  
]
```

generates something like this in the layout netlist:

```
M1 1 2 3 4 h w=2e-6 l=1e-6 $X=210000 $Y=200000
```

To qualify for the M representation, the Device operation must meet the following conditions: the element name must be M or m; the device must have 3 or 4 pins named G, S, D, and optional B (upper or lower case, in any order); pins 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:

```
.SUBCKT m g s d b  
.ENDS  
  
X1 1 2 3 4 m $[h]
```

Transistors having more than four pins are user-defined devices and are represented this way.

The SPICE parser 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.

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. By default, duplicate parameters that begin with W or L are not allowed. 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
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).

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 15-9. Specification Statements for SPICE M Elements

Statement	Purpose
LVS Spice Implied MOS Area	Indicates whether LVS should compute implied area values for MOS elements (element M) in SPICE netlists.
LVS Spice Multiplier Name	Specifies parameter names that should serve as multiplier factors in SPICE netlists instead of the standard SPICE parameter name M.
LVS Spice Replicate Devices	Indicates whether the LVS SPICE parser should physically replicate device elements in a SPICE netlist that own the M parameter.

BJT Element

The following strings define a bipolar junction transistor.

```
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>
```

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 define BJT devices with more than one substrate pin, define a primitive subcircuit as described in the section “[.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO](#)” on page 696.

Table 15-10. BJT Element

Argument Name	Description	Trace Property Name
Qxxx	BJT element name. Must begin with a Q followed by any number of allowed characters.	
nc	Collector terminal node name. String of any number of allowed characters.	
nb	Base terminal node name. String of any number of allowed characters.	
ne	Emitter terminal node name. String of any number of allowed characters.	
[ns]	Optional substrate terminal node name enclosed in brackets. String of any number of allowed characters. Ignored by LVS.	
ns	Optional substrate terminal node name. String of any number of allowed characters.	
mname	Model name. String of any number of allowed characters.	
a	Optional emitter area.	a
w	Optional emitter width.	w
l	Optional emitter length.	l

Table 15-10. BJT Element (cont.)

Argument Name	Description	Trace Property Name
m	Optional multiplier factor to simulate multiple BJTs. Property m is evaluated immediately and not handed down through parameter passing. 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.	m
OFF	Ignored.	
vbe	Ignored.	
vce	Ignored.	
parnam=pval	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 allowed characters. You can specify any number of parnam=pval pairs. See “ String Parameters ” on page 640 for complete details on string parameters.	parnam
\$SUB=ns	Optional substrate terminal node name, coded as a comment. Overrides regular SPICE substrate terminal field, if present.	
\$EA=a	Optional emitter area, coded as a comment. Overrides <AREA=a> or <a>. A practical use would be to use \$EA in the source, calculate the A property in the layout, then use Trace Property Q A A 1 for LVS comparison.	a
\$W=w	Optional width, coded as comment. Overrides property W if present.	w
\$L=l	Optional length, coded as comment. Overrides property L if present.	l
\$X=x \$Y=y	Optional x,y coordinates coded as comments. Integer numbers in database units. Used in hierarchical applications only.	
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.	

LVS component type: Q

LVS component subtype: mname

LVS pin names: c (collector), b (base), e (emitter), s (optional substrate).

See “[Ambiguous Element Checks](#)” on page 645 for information about how ambiguities in BJT syntax are handled.

Resistor Element

The following strings define a resistor.

```
Rxxx n1 n2 <mname> r <tc1 <tc2 <scale <m <ac>>>>>
+ <L=l> <W=w> <parnam=pval> ... <$SUB=ns>
+ <$ [mname] | $.MODEL=mname> <$W=w> <$L=l> <$X=x> <$Y=y> <$D=d>

Rxxx n1 n2 <mname> r <TC=tc1 <tc2 <scale>>> <M=m> <AC=ac>
+ <L=l> <W=w> <parnam=pval> ... <$SUB=ns>
+ <$ [mname] | $.MODEL=mname> <$W=w> <$L=l> <$X=x> <$Y=y> <$D=d>

Rxxx n1 n2 <mname> R=r <TC1=tc1> <TC2=tc2> <SCALE=scale>
+ <M=m> <AC=ac> <L=l> <W=w> <parnam=pval> ... <$SUB=ns>
+ <$ [mname] | $.MODEL=mname> <$W=w> <$L=l> <$X=x> <$Y=y> <$D=d>
```

The mname and **r** parameters in the first two syntaxes may be interchanged. If one of these two strings cannot be evaluated numerically (for example, if it is not a numeric value or a previously defined parameter), then it is assumed to be the mname parameter.

The following are some examples:

```
R1 1 2 4 $comment
R2 n5 n6 4 TC=2 3 4 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
```

To define resistor devices with more than one substrate pin, define a primitive subcircuit as described in the section “[.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO](#)” on page 696.

Table 15-11. Resistor Element

Argument Name	Description	Trace Property Name
Rxxx	Resistor element name. Must begin with an R followed by any number of allowed characters.	
n1	Positive terminal node name. String of any number of allowed characters.	
n2	Negative terminal node name. String of any number of allowed characters.	
mname	Optional model name. String of any number of allowed characters. May not be identical to argument r. ¹	
r	Resistance in ohms. May not be identical to the mname.	r

Table 15-11. Resistor Element (cont.)

Argument Name	Description	Trace Property Name
tc1	Ignored.	
tc2	Ignored.	
scale	Optional scale factor. Multiplies resistance (r).	
m	Optional multiplier factor used to simulate multiple parallel resistors. Property m is evaluated immediately and not handed down through parameter passing. Normally, divides resistance (r), multiplies width (w). If LVS Spice Replicate Devices YES is specified in the rule file, then m parallel copies of the resistor are created instead, and m for each copy is set to 1. Defaults to 1 if not specified. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement.	m
ac	Optional AC resistance for AC analysis.	ac
l	Optional length.	l
w	Optional width.	w
parnam= pval	Optional parameter name set to a numeric or string value. Arbitrary parameter names are allowed. The parnam must begin with a letter followed by any number of allowed characters. You can specify any number of parnam=pval pairs. See “ String Parameters ” on page 640 for complete details on string parameters.	parnam
ns	Optional substrate terminal node name coded as a comment. String of any number of allowed characters.	
[\$[mname] or \$.MODEL =mname	Optional model name, coded as a comment. String of any number of allowed characters. Overrides the regular optional mname parameter.	
\$W=w	Optional width, coded as a comment. Overrides the regular optional w parameter	w
\$L=l	Optional length, coded as a comment. Overrides the regular optional l parameter	l
\$X=x \$Y=y	Optional x,y coordinates coded as comments. Integer numbers in database units. Used in hierarchical applications only.	
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.	

1. If you have mname and “r” parameters of the same name, this was allowed in version 2010.2 and earlier. To restore that behavior, set the CALIBRE_OLD_PASSIVE_MODELS environment variable to any non-null value.

LVS component type: R

LVS component subtype: mname

LVS pin names: pos (positive), neg (negative), sub (optional substrate)

Voltage Source Element

The following string defines a voltage source.

```
Vxxx nplus nminus <<DC <=> dc <M=m> <parnam=pval> ... <$X=x> <$Y=y> <$D=d>
```

The following are some examples:

```
V1 n1 n2 5
V2 n1 n2 DC 5
V3 n1 n2 DC=5 AAA=5 BBB="zz"
```

To define voltage source devices with non-standard numbers of pins, define a primitive subcircuit as described in the section “[.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO](#)” on page 696.

Table 15-12. Voltage Source Element

Argument Name	Description	Trace Property Name
Vxxx	Voltage source element name. Must begin with a V followed by any number of allowed characters.	
nplus	Positive terminal node name. String of any number of allowed characters.	
nminus	Negative terminal node name. String of any number of allowed characters.	
dc	Optional DC and transient analysis value of the source.	dc
parnam= pval	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 allowed characters. You can specify any number of parnam=pval pairs. See “ String Parameters ” on page 640 for complete details on string parameters.	parnam

Table 15-12. Voltage Source Element (cont.)

Argument Name	Description	Trace Property Name
m	Optional multiplier factor to simulate multiple voltage sources. Property m is evaluated immediately and not handed down through parameter passing. 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.	m
\$X=x \$Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in hierarchical applications only.	
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.	

LVS component type: V

LVS component subtype: none

LVS pin names: pos (positive), neg (negative).

Subcircuit Definitions

Subcircuits are written by the hierarchical SPICE netlister for hcells, regular LVS Box cells, and cells created by the hierarchical database constructor ([ICV_*](#) cells). If the tool is run with the -flatten option, then there is only one subcircuit.

If you specify a top-level subcircuit name, then this subcircuit serves as the top-level network. The pins of this subcircuit serve as design ports. If a top-level subcircuit name is not specified, then the tool 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. This includes user-defined devices and LVS Box cells.
- Parameters are passed between levels of hierarchy. Primitive subcircuits can have arbitrary parameters.

.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO

The following strings define a subcircuit.

```
.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 > ...
```

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
```

All subcircuit definitions must terminate with a .ENDS or .EOM statement.

Table 15-13. SUBCKT Statements

Argument Name	Description
subname	Subcircuit name. String of any number of allowed characters.
n1 n2 ...	Node names for external reference. Strings of any number of allowed characters. Any element nodes appearing in the subcircuit but not included in this list or in a .GLOBAL or *.GLOBAL statement are strictly local. At least one of these parameters must appear in a called subcircuit.
/ m1 m2 ...	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.
parnam = pval	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 allowed characters. See “ String Parameters ” on page 640 for complete details on string parameters.

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.

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.

When the top-level network is enclosed in a subcircuit definition, hierarchical applications use the external node names of the top-level subcircuit as design ports.

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.

Here is an example:

```
.SUBCKT primitive a b c par1=5U par2=10U
.ENDS primitive
```

Here is the general form:

LVS component type: subcircuit_name

LVS component subtype: as specified in a subcircuit call or none.

LVS pin names: any allowed name

User-defined devices (ones that do not conform to built-in criteria) appear as primitive subcircuits. These include transistors having more than four pins, capacitors and resistors having more than three pins, and so forth.

All primitive subcircuits own a special property called M, which is available for tracing. The Trace Property name is M. The value is always 1. This property is not available for tracing in non-primitive subcircuits.

Nested Subcircuits

Subcircuit definitions may be nested. 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      $$ Beginning of subcircuit aaa.  
$\n.subckt bbb a b c    $$ Local definition of bbb #1;  
c1 a b                $$ scope is aaa.  
c2 b c  
.ends bbb  
$\n.x1 d e f bbb       $$ Call to bbb #1.  
.ends aaa  
.subckt ccc d e f     $$ Beginning of subcircuit ccc.  
$\n.subckt bbb a b c    $$ Local definition of bbb #2;  
r1 a b                $$ scope is ccc.  
r2 b c  
.ends bbb  
$\n.subckt ddd a b c    $$ Call to bbb #2.  
x1 a b c bbb  
.ends ddd  
$\n.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
```

Duplicate Pins

The 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 cell1 a a  
C1 a b 100  
.ENDS  
  
.SUBCKT cell2  
X1 1 2 cell1  
.ENDS
```

In the subcircuit call X1, node 1 connects to C1 in cell1 through pin a of cell1. Node 2 does not connect to any devices in cell1. A warning is issued about duplicate definition of pin a in cell1.

Duplicate .SUBCKT Definitions

The 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.

You can tell the SPICE parser to report warnings for duplicate subcircuit definitions with different pin counts 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, as follows:

```
Warning: Duplicate subckt definition "<cell>" at line <N> in file
"<pathname>" previously defined at line <n> in file "<pathname>"
```

The SPICE parser ceases these warnings after 256 duplicates have been found.

Subcircuit Calls

Strings beginning with X, having pin names, and referencing a subcircuit definition name constitute subcircuit calls.

```
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> ... >
```

Here is an example:

```
.SUB opamp 1 2 3
...
.ENDS opamp
X1 2 4 17 opamp wn=100 ln=5 pp="abc"
```

Table 15-14. Subcircuit Calls

Argument Name	Description
Xyyy	Subcircuit call name. Must begin with an X followed by any number of allowed characters.
n1 n2 ...	Node names for external reference. Strings of any number of allowed characters. The program references the names in the order they are specified in the subcircuit definition.
/	Optional. The / characters in the node name list and in the subcircuit reference name are treated as white space, unless you specify the LVS Spice Slash Is Space NO specification statement in your rule file. See \$PINS also.
subname	Subcircuit reference name. Must appear in a corresponding .SUBCKT or .MACRO definition in the netlist. A subcircuit call can appear before or after the corresponding subcircuit definition.

Table 15-14. Subcircuit Calls (cont.)

Argument Name	Description
parnam = pval	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 allowed characters. See " String Parameters " on page 640 for complete details on string parameters.
m	Optional multiplier factor. Generates M subcircuit calls connected in parallel. Property m is evaluated immediately and not handed down through parameter passing. 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.
mname	Optional model name, coded as a comment. String of any number of allowed characters. Valid only on calls to primitive subcircuits. Used as LVS component subtype for the call.
\$T=tx ty r a	Optional layout transform consisting of x translation, y translation, reflection, and rotation-angle. Used in hierarchical applications only. The translation components tx and ty are integer numbers in database units. The reflection r is along the x-axis and is either 0 or 1 which denote the absence or presence of reflection respectively. The rotation angle a is an integer number in degrees and the only valid values are 0, 90, 180, or 270; rotation is counter-clockwise.
\$X=x \$Y=y	Optional x,y coordinates coded as comments. Integer numbers in database units.
\$D=d	Optional rule file Device statement identifier, coded as comment, where d is a non-negative integer. Used in extracted layout netlists to identify the rule file Device statement that generated the device. For example, \$D=0 means the first Device statement in the rule file was used to define the netlist element. Parsed but not used.

Table 15-14. Subcircuit Calls (cont.)

Argument Name	Description
\$PINS <pin=node> ...	<p>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 allowed characters. Embedded / characters are treated as part of the names. For example:</p> <pre>.SUBCKT F A B \$ subcircuit contents... .ENDS X1 F \$PINS B=1 A=2</pre> <p>In subcircuit call X1, pin B connects to node 1 and pin A connects to node 2.</p> <p>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:</p> <pre>.SUBCKT SSS A B C D \$ subcircuit contents... .ENDS X2 1 2 SSS X3 SSS \$PINS B=1 D=2</pre> <p>In subcircuit call X2, pins C and D are floating. In subcircuit call X3, pins A and C are floating.</p> <p>Notes:</p> <ol style="list-style-type: none"> 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. The \$PINS argument overrides the standard SPICE node specification <n1 n2 ...> if one is present. The 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. 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 does not contain any element statements or subcircuit calls. The corresponding cell name to a primitive subcircuit can appear in an LVS Box specification statement.

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 the tool 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 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 ensures 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:

```
.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 by 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 nmLVS or ICtrace and should be used with caution because it may generate incorrect connectivity.

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. Calibre applications allow floating pins in subcircuit calls.

Example:

```
.GLOBAL VCC VSS

.SUBCKT SSS A B C D
...
.ENDS

.SUBCKT XXX E VCC VSS
...
.ENDS

.SUBCKT ZZZ
X1 1 2 SSS      $ Pins C and D are floating.
X2 SSS $PINS B=1 D=2 $ Pins A and C are floating.
X3 XXX          $ Pins E, VCC and VSS are floating.
.ENDS
```

As shown in the example, in regular subcircuit calls, pin reference is positional, starting with the first pin, and floating pins are at the end of the pin list. When the \$PINS syntax is used, pin reference is by name and any other pins are floating.

In hierarchical operation, for each floating pin in a subcircuit call, if the pin name is not global, then the SPICE reader creates a respective floating net in the subcircuit that contains the call. The floating net is connected to the floating pin. The floating net name consists of the instance name (including the prefix X) followed by a slash (/) followed by the pin name. In the 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 Calibre 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](#).

Block-Level Chip Assembly Using Scoped SPICE Subcircuits

When assembling a schematic design from several block-level components, some high-level SPICE blocks may use lower-level subcircuits that conflict with block names used by other high-level blocks. Because SPICE cannot distinguish between the calls to the conflicting cells, this situation results in an incorrect instantiation of the first encountered cell for all instances of the conflicting name.

Example:

```
---- file blk1.cdl ----
.SUBCKT A P2 P1 $$ This subcircuit conflicts with SUBCKT A in blk2.cdl
C0 P1 P2 40
.ENDS

.SUBCKT BLK1 A[1] A[0] B C
XA0 A[1] C A
XA1 A[0] B A
.ENDS
---- file blk2.cdl ----
.SUBCKT A P2 P1 $$ This subcircuit conflicts with SUBCKT A in blk1.cdl
R0 P1 P2
.ENDS

.SUBCKT BLK2 A[1] A[0] B C
XA0 A[1] C A
XA1 A[0] B A
.ENDS
---- file top.cdl ----
.INCLUDE blk1.cdl
.INCLUDE blk2.cdl

.SUBCKT TOP
X0 A B C D BLK1
X1 A B C D BLK2
.ENDS
```

When this SPICE netlist is read, the following warning results:

```
LVS Netlist Compiler - Errors and Warnings for "top.cdl"
-----
Warning: Duplicate subckt definition "A" at line 1 in file "blk2.cdl"
previously defined at line 1 in file "blk1.cdl"
```

and SUBCKT A from *blk1.cdl* is used for all instantiations of A in the design.

SPICE has a mechanism for scoping that can alleviate this issue. In order to avoid the issue, a top-level SUBCKT is used to wrap the entire high-level block. That top-level SUBCKT in turn

contains the entire file along with a single-instance call to the top-level block inside. For example, one can replace *top.cdl* with the following top level netlist:

```
---- new top.cdl ----
$$ Wrapper subcircuit that protects BLK1
.SUBCKT BLK1_WRAPPER A[1] A[0] B C
.INCLUDE blk1.cdl
X0 A[1] A[0] B C BLK1
.ENDS

$$ Wrapper subcircuit that protects BLK2
.SUBCKT BLK2_WRAPPER A[1] A[0] B C
.INCLUDE blk2.cdl
X0 A[1] A[0] B C BLK2
.ENDS

.SUBCKT TOP
X0 A B C D BLK1_WRAPPER
X1 A B C D BLK2_WRAPPER
.ENDS
```

Note the new .SUBCKT entries BLK1_WRAPPER and BLK2_WRAPPER. Each contains the .INCLUDE for a high-level block, which remains unchanged from before. Each also contains a single call to the lower-level block being instantiated. All instance calls inside the scope of the wrapper cell resolve inside the wrapper cell first. Cells inside the wrapper cell are no longer visible outside the wrapper cell. Thus, the conflict between the two A cells is resolved.

V2LVS supports the -cfg option, which helps to resolve this situation in some Verilog designs. See “[Enable Subcircuit Scoping](#)” on page 754.

Multiplier (m) Parameters

Built-in SPICE elements support a multiplier factor parameter, which is called “m” by default. This parameter is evaluated immediately and is not handed down through parameter passing.

The multiplier parameter defines a multiplication factor applied to certain built-in parameters. Either the value of the multiplier parameter or its reciprocal is used, depending on the other built-in parameter. The affected built-in parameters are defined in the tables corresponding to each SPICE element discussed in the “[Element Statements](#)” section. The row that defines “m” in a table discusses which built-in parameters are scaled by the “m” value. (The [LVS Spice Multiplier Name](#) specification statement can change the multiplier parameter name.)

If you want to apply the multiplication factor parameter to user-defined parameters, then you can do either of the following:

- Use [LVS Property Map](#) to rename the user property name to a built-in name. For example:

```
LVS PROPERTY MAP MN W "WR" LAYOUT
```

specifies that for MN devices, the property W is represented as WR in the input layout database.

- Use [LVS Property Initialize](#) to compute the desired property values. (See the examples in that section for usage ideas.)

The multiplier parameter can also be used to replicate devices through the [LVS Spice Replicate Devices](#) YES or LPI keywords. The LPI option can be useful if using LVS Property Initialize as discussed previously when you want device replication to occur.

Eldo Format Y Element

The Calibre SPICE parser supports the Y element used in Eldo®. The Y element is another form of the instantiation element X (as in subcircuit calls), but with a different syntax.

The general form is this:

`Yxxx subname [GENERIC: {parnam = pval} ...] [PORT: net ...] [$ comments]`

The *subname* is subcircuit name.

Keywords PORT and GENERIC are case-insensitive and can be followed by an optional whitespace character before the colon. Hence, “PORT:” and “port :” are both allowed.

The “GENERIC: *parnam* = *pval*” specification may appear zero or more times to define all the desired parameters.

At least one “PORT: *net*” specification must appear to connect the instantiation in the circuit, although this is not required. 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.

The Y device is treated by the SPICE parser 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 in a discrepancy report, for example, the instance name starts with Y instead of X.

Chapter 16

V2LVS

The V2LVS (Verilog-to-LVS) tool translates a Verilog structural netlist into a SPICE netlist suitable for Calibre nmLVS/Calibre nmLVS-H and Calibre PERC.

This chapter covers each part of the formal Backus Naur Form specification for the Verilog language in the IEEE standard (IEEE Computer Society, IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language, IEEE Std 1364-2001). This chapter is best read in conjunction with the standard document itself or with another Verilog text (for example, D.E. Thomas and P.R. Moorby, *The Verilog Hardware Description Language*, 5th ed., Springer Science+Media, Inc., New York, 2002).

V2LVS Overview	709
Basic V2LVS Usage	712
Modules	714
Declarations	724
Primitive Instances	742
Behavioral Statements	744
Specify Block	745
Expressions	745
Compiler Directives	749
Library Files	750
Simulation Output Generation	760
Calibre xRC Source Template File	762
Collecting SPICE .INCLUDE Netlists into a Single Netlist	763
Creating LVS Box Subcircuits	765
V2LVS Tcl Interface	766
Supported Verilog Syntax	833
V2LVS Command Line Syntax	836
V2LVS Errors and Warnings	843

V2LVS Overview

V2LVS converts a structural Verilog design into an equivalent netlist in extended SPICE form.

The section “[SPICE Format](#)” on page 633 describes the output netlist format and extensions produced by the Calibre netlister. This format is consistent with netlists produced by V2LVS.

V2LVS has these primary modes of operation:

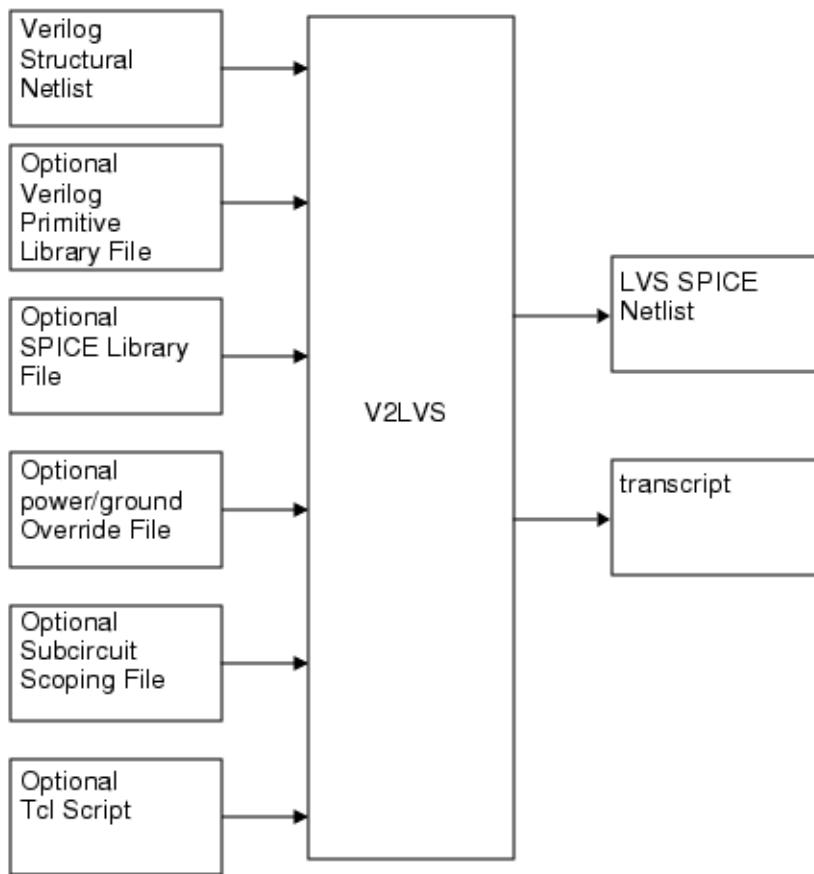
Tcl shell — Tcl shell interface commands are used. The section “[V2LVS Tcl Interface](#)” on page 766 shows the Tcl shell commands. The Tcl shell interface may be used interactively, or in batch mode with the -tcl command line option.

Command line — Command line options are used exclusively without the -tcl option. See “[V2LVS Command Line Syntax](#)” on page 836.

Command line mixed mode — Command line options are used together with the -tcl option, which references a script of Tcl shell commands.

The Tcl shell interface is the preferred mode of operation. The Tcl shell provides all of the historical command line interface capabilities within a scripting environment. In addition, the Tcl interface provides commands to bias bulk pins in a Place and Route environment, which the command line interface does not have.

[Figure 16-1](#) shows the V2LVS inputs and outputs.

Figure 16-1. Typical V2LVS Inputs and Output

The Verilog structural netlist is a required input and is checked for syntactical correctness. Multiple Verilog design files or Verilog primitive library files must be concatenated into single files prior to running V2LVS. V2LVS supports the `include compiler directive.

The Verilog primitive library file (when specified) is accessed to find the pins associated with nets shown in positional cell instances. The Verilog library file generally contains the module definitions that are already implemented in a SPICE library file.

The optional SPICE library file generally contains transistor-level details of the primitive modules. The 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.

Warning messages are issued for behavioral syntax found in the Verilog netlist, and modules containing behavioral syntax are not translated.

Complete licensing information is provided under “[Licensing: Physical Verification Products](#)” in the *Calibre Administrator’s Guide*.

Basic V2LVS Usage

An interactive V2LVS Tcl shell session is a good way to understand basic V2LVS usage and to develop scripts.

Here is a Verilog input file *design.v*.

Example 16-1. Basic V2LVS Usage

```
//////////////////////////// Verilog file //////////////////////////////
module B (.P11(P1), .Q11({Q1,Q3}), .R11({R1, R1, Q4}), .P12(P1));
    input P1;
    output Q1, R1;
    output [0:1] Q3, Q4;
endmodule

module A ( P, Q, R );
    input P;
    output [0:2] Q, R;
    wire [0:2]S;
    B Binst( P, Q, R, S );
endmodule
//////////////////// End of Verilog file /////////////////////
```

The first command in the following sequence starts the Tcl shell. The subsequent commands load the Verilog design, set the warning level, the Verilog version, get the modules under cell A, and write the translated output.

```
% v2lvs -tcl
...
% v2lvs::set_warning_level -level 1
% v2lvs::verilog_version -version 2001
% v2lvs::load_verilog -filename design.v
% v2lvs::find_module -under A
Info: Creating Design Database ...
B
% v2lvs::write_output -filename A.out
% exit
```

The *A.out* file contains this:

```
*.BUSDELIMITER [
.SUBCKT B P1 Q1 Q3[0] Q3[1] R1 R1 Q4[0] Q4[1] P1
.ENDS

.SUBCKT A P Q[0] Q[1] Q[2] R[0] R[1] R[2]
XBinst P Q[0] Q[1] Q[2] 1000 R[0] R[1] R[2] S[2] B
.ENDS
```

You can put the Tcl commands into a script and run it using this command line:

```
% v2lvs -tcl <script_name>
```

Here is a basic script that reads in a Verilog design and associated Verilog and SPICE library files:

```
v2lvs::load_verilog -filename design.v
v2lvs::load_verilog -filename lib.v -lib_mode
v2lvs::load_spice -filename stdcells.sp -range_mode
v2lvs::load_spice -filename memory.sp -range_mode
v2lvs::load_spice -filename dac.sp -range_mode
v2lvs::load_spice -filename pads.sp -range_mode
v2lvs::set_includes -filename {stdcells.sp memory.sp dac.sp pads.sp}
v2lvs::write_output -filename my_design.sp
```

This script reads in the Verilog design, includes the SPICE library files in the output netlist, and writes pins using range mode. There is another mode called pin mode that can also be used depending on how the ports are arranged.

Related Topics

[V2LVS Tcl Interface](#)

[V2LVS Used Without Verilog or SPICE Library Files](#)

[Range Mode](#)

[Pin Mode](#)

Modules

V2LVS supports modules and macro modules. Both are mapped directly into SPICE subcircuits having no exceptions. 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.

Any defparam (default parameter) declarations are ignored.

Example 16-2. Simple Verilog Module Translation to SPICE Subcircuit

In this example, the top level module B creates an instantiation of module A called inst1. Module B passes in the arguments w1, w2, and w3:

```
module A ( in1, in2, out3 );
  input in1, in2;
  output out3;
endmodule

module B ();
  wire w1, w2, w3;
  A inst1( .in1(w1), .in2(w2), .out3(w3) );
endmodule
```

and is translated into this 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.

Named Port Declarations in a Verilog Module	715
Port Selection, Bit Selection, and Array Mapping	715
Undefined Modules	716
User-Defined Primitives	718
Module Instantiations	718
Call Conventions	718
Bit Expressions	720
Unnamed Concatenation Expressions in Declarations	721
Unconnected Pins	722

Named Port Declarations in a Verilog Module

Ports declared in Verilog modules are translated to subcircuit ports.

In this example, module A declares ports INA, INB, and OUTC, which are connected to in1, in2, and out3 respectively:

```
module A ( .INA(in1), .INB(in2), .OUTC(out3) );
  input in1, in2;
  output out3;
endmodule

module B ();
  wire w1, w2, w3;
  A inst1( .INA(w1), .INB(w2), .OUTC(w3) );
endmodule
```

This is translated as follows:

```
.SUBCKT A in1 in2 out3
.ENDS

.SUBCKT B
Xinst1 A $PINS in1=w1 in2=w2 out3=w3
.ENDS
```

Port Selection, Bit Selection, and Array Mapping

Port and bit selections in port references are supported. In this example, module AA uses a port selection [2:3] on the input array TH and a bit selection [4] on the input array UH. SH is a simple array that is used intact as an output array.

```
module AA (TH[2:3], UH[4], SH) ;
  input [0:3] TH;
  input [0:7] UH;
  output [3:4] SH;
endmodule

module BB ();
  wire[0:1] w1;
  wire w2;
  wire [1:2] w3;
  AA inst1 ( .TH(w1), .UH(w2), .SH(w3) );
endmodule
```

This translates as follows:

```
.SUBCKT AA TH[2] TH[3] UH[4] SH[3] SH[4]
.ENDS

.SUBCKT BB
Xinst1 AA $PINS TH[2]=w1[0] TH[3]=w1[1] UH[4]=w2 SH[3]=w3[1] SH[4]=w3[2]
.ENDS
```

Related Topics

[Declarations](#)

Undefined Modules

By default, V2LVS processes undefined modules by attempting to translate them into correct SPICE instances. The tool processes all placements of the undefined module and makes an assumption about port widths from the context of the placements. The assumption is used when writing the SPICE output.

Example — Undefined module processing

In this example Verilog code, notice in the following Verilog code that module2 has no module definition block. There are two instances of m2_01sig_unclear. From these placements, the tool assumes the width of a pin is 1 bit.

```
module first_module ( ssig, asig, vss );
    output [7:0] ssig;
    output [15:0] asig;
    inout vss;

// NOTICE THIS INSTANCE
module2 m2_01sig_unclear (
    .spin(),
    .apin(asig[15]),
    .gnd( vss )
);
endmodule

module top_module ( ssig, asig, vss );
    output [7:0] ssig;
    output [15:0] asig;
    inout vss;
    first_module m0( ssig, asig, vss );
    module1 m1_00sig_correct (
        .spin(),
        .apin(),
        .gnd( vss )
);

module1 m1_12sig_wrong (
    .spin( ssig[7] ),
    .apin( { asig[15], asig[7] } ),
    .gnd( vss )
);
```

```
// DUPLICATE INSTANCE
module2 m2_01sig_unclear (
    .spin(),
    .apin(asig[15]),
    .gnd( VSS )
);

module2 m2_12sig_unclear (
    .spin( ssig[7] ),
    .apin( { asig[15], asig[7] } ),
    .gnd( VSS )
);

moduleR mR_12sig_reference (
    .spin( ssig[7] ),
    .apin( { asig[15], asig[7] } ),
    .gnd( VSS )
);
);
endmodule
```

By default, this is translated into SPICE as follows:

```
.SUBCKT first_module ssig[7] ssig[6] ssig[5] ssig[4] ssig[3] ssig[2]
ssig[1]
+ ssig[0] asig[15] asig[14] asig[13] asig[12] asig[11] asig[10] asig[9]
asig[8]
+ asig[7] asig[6] asig[5] asig[4] asig[3] asig[2] asig[1] asig[0] VSS
$$ NOTICE THE NEXT LINE
Xm2_01sig_unclear module2 $PINS apin[0]=asig[15] gnd=VSS
.ENDS

.SUBCKT top_module ssig[7] ssig[6] ssig[5] ssig[4] ssig[3] ssig[2] ssig[1]
+ ssig[0] asig[15] asig[14] asig[13] asig[12] asig[11] asig[10] asig[9]
asig[8]
+ asig[7] asig[6] asig[5] asig[4] asig[3] asig[2] asig[1] asig[0] VSS
Xm0 first_module $PINS ssig[7]=ssig[7] ssig[6]=ssig[6] ssig[5]=ssig[5]
+ ssig[4]=ssig[4] ssig[3]=ssig[3] ssig[2]=ssig[2] ssig[1]=ssig[1]
ssig[0]=ssig[0]
+ asig[15]=asig[15] asig[14]=asig[14] asig[13]=asig[13] asig[12]=asig[12]
+ asig[11]=asig[11] asig[10]=asig[10] asig[9]=asig[9] asig[8]=asig[8]
+ asig[7]=asig[7] asig[6]=asig[6] asig[5]=asig[5] asig[4]=asig[4]
asig[3]=asig[3]
+ asig[2]=asig[2] asig[1]=asig[1] asig[0]=asig[0] VSS=VSS
Xm1_00sig_correct module1 $PINS gnd=VSS
Xm1_12sig_wrong module1 $PINS spin=ssig[7] apin[1]=asig[15]
apin[0]=asig[7]
+ gnd=VSS
Xm2_01sig_unclear module2 $PINS apin[0]=asig[15] gnd=VSS
Xm2_12sig_unclear module2 $PINS spin=ssig[7] apin[1]=asig[15]
apin[0]=asig[7]
+ gnd=VSS
XmR_12sig_reference moduleR $PINS spin=ssig[7] apin[1]=asig[15]
apin[0]=asig[7]
+ gnd=VSS
.ENDS
```

User-Defined Primitives

UDP Declarations are treated the same way as behavioral Module declarations; they are used to specify the calling interface of the underlying module. UDPs are ignored in translation (with a warning), but they can be called from other modules.

The SPICE library is assumed to contain SPICE-level representations of any UDP that is called in the structural netlist.

Module Instantiations

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 [Figure 16-2](#) on page 714 for a simple case of a module instantiation. Verilog also supports array and concatenation constructs.

Related Topics

[Modules](#)

Call Conventions

V2LVS uses 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, Bit Selection, and Array Mapping](#)” on page 715 shows a simple case of instantiations using ranges. The “[Example — Use of supply0 and supply1 net types](#)” on page 730 has an example that shows a simple case of instantiation using bit expressions.

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.

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

In the following Verilog circuit, module B creates 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 generated module call maps the right-most pins of IN_ARR to the instance pins P:

```
.SUBCKT A P[3] P[2] P[1] P[0] Q
.ENDS

.SUBCKT B
Xinst1 A $PINS P[3]=IN_ARR[4] P[2]=IN_ARR[5] P[1]=IN_ARR[6] P[0]=IN_ARR[7]
Q=X
.ENDS
```

A warning is issued when calling pins are wider than the port 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 right-most called port. Remaining ports are mapped to arbitrary 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:

```
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 left-most P pins to sequential unconnected nets:

```
.SUBCKT A P[3] P[2] P[1] P[0] Q
.ENDS

.SUBCKT B
Xinst1 A $PINS P[1]=IN_ARR[0] P[0]=IN_ARR[1] Q=X
.ENDS
```

Bit Expressions

Bit expressions are supported by V2LVS. Special bit values of x and z are not supported and cause termination of the run.

Bit expressions take the form n'<t><val>.

- 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

By default, pin connections of value 1 are assigned to global net VDD in SPICE, and pin connections of value 0 are assigned to global net VSS in SPICE.

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:

```
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.

```
.SUBCKT A in1[0] in1[1] in1[2] in1[3] in1[4] in1[5] in1[6] in1[7] out1
.ENDS

.SUBCKT B
Xinst1 A $PINS in1[0]=a[0] in1[1]=a[1] in1[2]=VSS in1[3]=VDD in1[4]=a[0]
+ in1[5]=a[1] in1[6]=VSS in1[7]=VDD out1=b
.ENDS
.GLOBAL VDD
.GLOBAL VSS
```

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 instances of the contents of the concatenation.

The `v2lvs::override_globals` -supply0 and -supply1 options define new default values for net connections to 0 or 1. 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 -supply0 and -supply1 options override this assignment and cause the values of 0 and 1 to be connected to global ground and power nets.)

Unnamed Concatenation Expressions in Declarations

For the case of unnamed concatenation expressions in declarations, ports are unnamed and can only be connected by positional calls.

In the following code, the port is unnamed.

```
module A ( {B,C} )
```

This module can be instantiated with positional calling only. Attempts to call via named connections always leave 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.

```
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 B C
.ENDS

.SUBCKT B
Xinst1 A $PINS B=a[0] C=a[1]
.ENDS
```

Related Topics

Modules

Unconnected Pins

Unconnected pins are uncalled. Instantiations that leave pins uncalled in Verilog do not appear in the output. Calibre nmLVS handles these unconnected pins appropriately.

Example — Unconnected pins

The following Verilog circuit:

```
module A ( B, C, D );
  input B, C;
  output [1:0] D;
endmodule

module B ();
  wire bb,cc;
  A anA (.B(bb), .C(cc));
  A anA1 ( x, b );
endmodule
```

converts into the following SPICE subcircuit:

```
.SUBCKT A B C D[1] D[0]
.ENDS

.SUBCKT B
XanA A $PINS B=bb C=cc
XanA1 A $PINS B=x C=b
.ENDS
```

If `v2lvs::number_unconnected_pins` -enable (command line: -n) is used, it causes unconnected pins to be connected to sequential, undeclared, numbered nets as shown here:

```
.SUBCKT A B C D[1] D[0]
.ENDS

.SUBCKT B
XanA A $PINS B=bb C=cc D[1]=1000 D[0]=1001
XanA1 A $PINS B=x C=b D[1]=1002 D[0]=1003
.ENDS
```

The [v2lvs::generate_ordered_pins](#) -enable option (command line: -i) causes V2LVS to generate numbered, unconnected pins as shown here:

```
.SUBCKT A B C D[1] D[0]
.ENDS
```

```
.SUBCKT B
XanA bb cc 1000 1001 A
XanA1 x b 1002 1003 A
.ENDS
```

Declarations

The following declaration types are meaningful to V2LVS: input, output, inout, all net types, and parameter.

These behavioral declaration types are *not supported* by V2LVS and cause a containing module not to be translated: reg, time, integer, real, realtime, event, function, and task.

Port Declarations	724
Net Types	727
Addition of Pins	733
Management of Bulk Pin Connections	740
Net Assignment	742

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.

All styles of port declaration and initialization as specified in IEEE Std 1364-2001 are supported, including these:

- data-type (integer, etc.) with port declarations
- output port declaration including: net-type, reg, integer, time, signed
- output reg/integer/time initialization
- ANSI ports with UDP declaration

Example — ANSI port declarations

Each of the following show ANSI port declarations.

```
module ansi_ports (
    input  read, read2,
    input  write,
    input  [7:0] data_in, data_in2,
    output [7:0] data_out, data_out2,
    inout io1, io2 );

    primitive test ( output reg out, input a, sel, clk);

    module top(input wire [7:0] i, output reg [7:0] o);
    module top(input wire [7:0] i, output real o);
    module top(input .i({a[0:3]}), output o=1);
    module top(input wire [7:0] in1, output integer oi1, oi2, output time ot);
```

Example — Identical port and net names

In the following module, the .data port has the same name as the data net:

```
module test ( addr, .data ({data[3], data[2], data[1], data[0]} );
  input [0:3] addr;
  output [0:3] data;
endmodule
```

This is translated as follows:

```
.subckt test addr[0] addr[1] addr[2] add[3] data[3] data[2] data[1]
data[0]
.ends
```

Example — Port and net names forming a cycle

In the following code, module one names interior net B with port name A and interior net A with port name B:

```
module two ( p1 );
  input p1;
endmodule

module one ( .A(B), .B(A) );
  input A;
  input B;
  two b1 ( B );
  two b2 ( A );
endmodule

module top ( A, B );
  input A;
  input B;
  one t1 ( .A(B), .B(A) );
  one t2 ( .A(B), .B(A) );
endmodule
```

Because SPICE does not have a separate namespace for ports and interior nets, V2LVS outputs the following netlist. Note the port names for “one” are removed, and the interior net names are used instead. Subcircuit calls to “one” are appropriately mapped to the correct net name:

```
.SUBCKT two p1
.ENDS

.SUBCKT one B A
Xb1 two $PINS p1=B
Xb2 two $PINS p1=A
.ENDS

.SUBCKT top A B
Xt1 one $PINS B=B A=A
Xt2 one $PINS B=B A=A
.ENDS
```

Related Topics

[Modules](#)

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. Signed and unsigned properties may be used in net declarations. Initialization is supported during declaration, for example: “reg a = 0;”.

All other net types are handled as wires for the purposes of LVS. Drive strengths, charge strengths, and delays are all ignored.

Power and Ground Connections	727
Power and Ground Signal Overrides.....	731

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 to support different power and ground conventions used in different chips.

Numeric values like 1'b1 and 1'b0 are translated using the following rules:

- If local supply0 or supply1 nets are present, references to those nets are made.
- If local nets are not present, global power or ground connection to the nets VDD or VSS are made.

The [v2lvs::override_globals](#) command provides control of power and ground connectivity (command line: -s0, -s1, -sk, -sl, -sn, and -so).

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 v2lvs::override_globals -supply0 and -supply1 options alter this default translation so that all numeric values of 1 are translated to the -supply1 argument, and all numeric values of 0 are translated to the -supply0 argument.

The v2lvs::override_globals -supply_not_connected and -use_local_supply options allow the -supply0 and -supply1 options to override global signals while keeping supply0 and supply1 nets separate from the global signals. The -supply_not_connected and -use_local_supply options are exclusive and perform a similar function except that -supply_not_connected generates a .GLOBAL statement for the local supply net names while -use_local_supply does not.

The v2lvs::override_globals -default_not_connected option prevents generation of .GLOBAL SPICE declarations for global signals used in the Verilog netlist.

Connected Global Supply Nets

Certain design flows have different power and ground net names in disparate places in the design, and you want all of these power and ground nets to be connected together. This typically happens when different net naming conventions are used in various places in a design, and you want to create one logical net from several names.

You can use the [v2lvs::override_globals](#) -supply0 and -supply1 options to connect global supplies. For example, when you use the -supply1 PWR option, 1'b1 is always translated to PWR.

Use the -supply0 and -supply1 options 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 options cause all power nets to be connected and all ground nets to be connected.

Separated Supply Nets

Certain design flows, most often with mixed digital/analog circuits, have the need for separate power nets that are not connected and the same for ground nets.

The [v2lvs::override_globals](#) -supply_not_connected option causes local supply0 and supply1 declarations to continue to take precedence over -supply0 and -supply1 arguments. It also causes these power and ground nets not to be connected to one another. Use the -supply_not_connected option in conjunction with -supply0, -supply1, and local supply0 and supply1 nets to support circuits with multiple power and ground supplies that are to be kept separate.

The following table illustrates a power signal pin translation under various combinations of conditions. The G_P signal is intended as global power, and the L_P signal is intended as local power.

Table 16-1. Mapping of 1'b1 Signals

v2lvs::override_globals option combination	modules with supply1 L_P get these signals	modules with no supply1 get these signals	.GLOBAL signals	Notes
none	L_P	VDD	VDD L_P	

Table 16-1. Mapping of 1'b1 Signals (cont.)

v2lvs::override_globals option combination	modules with supply1 L_P get these signals	modules with no supply1 get these signals	.GLOBAL signals	Notes
-supply1 G_P	G_P this is added to the SPICE netlist: *.CONNECT G_P L_P	G_P	G_P	overrides all power signals
-supply1 G_P -supply_not_connected	L_P	G_P	G_P L_P	overrides global power only
-supply1 G_P -use_local_supply	L_P	G_P	G_P	overrides global power only; avoids .GLOBAL for local signals
-default_not_connected	L_P	VDD	L_P	no overrides; avoids .GLOBAL for global supply
-use_local_supply	L_P	VDD	VDD	no overrides; avoids .GLOBAL for local supply
-supply1 G_P -default_not_connected	G_P this is added to the SPICE netlist: *.CONNECT G_P L_P	G_P	none	overrides all power; avoids .GLOBAL for global power supply
-supply1 G_P -supply_not_connected -default_not_connected	L_P	G_P	L_P	overrides global power only; avoids .GLOBAL for global power supply
-supply1 G_P -use_local_supply -default_not_connected	L_P	G_P	none	overrides global power only; avoids .GLOBAL for all power supplies

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:

```
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
```

By default, 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 `v2lvs::override_globals -supply1 PWR -supply0 GND` options are used, it translates as follows:

```
.SUBCKT A in1[0] in1[1] out1[0] out1[1]
.ENDS

.SUBCKT B
*.CONNECT PWR PWR[0]
*.CONNECT PWR PWR[1]
*.CONNECT GND GND[0]
*.CONNECT GND GND[1]
Xinst1 A $PINS in1[0]=GND in1[1]=PWR out1[0]=w1[0] out1[1]=w1[1]
.ENDS
.GLOBAL PWR
.GLOBAL GND
```

Notice that the `*.CONNECT` statements connect all power nets to one another and all ground nets to one another.

If the `-supply_not_connected` option is used in addition to the `-supply0` and `-supply1` options, the translation is the same except the `*.CONNECT` statements are omitted, and all power and ground nets are made into `.GLOBAL` nets:

```
.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
.GLOBAL GND
.GLOBAL PWR[0]
.GLOBAL PWR[1]
.GLOBAL GND[0]
.GLOBAL GND[1]
```

Related Topics

[Power and Ground Signal Overrides](#)

Power and Ground Signal Overrides

The `v2lvs::override_globals -custom_override` option (command line: `-so`) allows module- and instance-specific overrides for numeric constant signals in Verilog netlists. This option governs the conversion of numeric constants like `1'b1` and `1'b0` within Verilog netlists. This is implemented through a supply override file.

The format of a line in the override file is this:

module [instance_name] supply0= spice_net supply1= spice_net

module — A required Verilog module name. This field is parsed as a Verilog identifier. Escaped identifiers are allowed and interpreted similarly to the way they are in a Verilog netlist. See “[Expressions](#)” for more information.

instance_name — An optional Verilog instance name. This field is also parsed as a Verilog name similarly to ***module***. If not present, the override functions similarly to a `supply0` or `supply1` statement in the Verilog module itself and applies to all instances in the module that do not have separately-specified instance-specific overrides. SPICE signals generated by these overrides are also issued as `.GLOBAL` statements in the output SPICE netlist unless overridden by the `-use_local_supply` option.

supply0= — A literal keyword indicating a ground supply name mapping.

supply1= — A literal keyword indicating a power supply name mapping.

spice_net — A SPICE net name specified with ***supply0=*** and ***supply1=***. This field is parsed as a SPICE name and follows SPICE conventions.

The overrides are specified one per line. The + character on the first character of a line serves as a line continuation character similar to SPICE. Comments are specified with //.

Example — Use of supply override file

Consider the following Verilog modules:

```
module A ( i, i2 );
  input i,i2;
endmodule

module B ( );
  supply0 VSS;
  supply1 VDD;
  A A1( 1'b1, 1'b0 );
  A A2( 1'b1, 1'b0 );
  A A3( 1'b1, 1'b0 );
endmodule
```

With this command sequence:

```
v2lvs::load_verilog -filename example.v
v2lvs::override_globals -use_local_supply -default_not_connected
```

the module normally creates the following output from V2LVS:

```
.SUBCKT A i i2
.ENDS

.SUBCKT B
XA1 A $PINS i=VDD i2=VSS
XA2 A $PINS i=VDD i2=VSS
XA3 A $PINS i=VDD i2=VSS
.ENDS
```

The following can be used instead:

```
v2lvs::load_verilog -filename example.v
v2lvs::override_globals -use_local_supply -default_not_connected \
-custom_override supply_overrides.txt
```

If the file *supply_overrides.txt* contains the following override entries:

```
// Global override for module B
B supply0=GND supply1=PWR
// Instance specific overrides for module B
B A1 supply0=VSS0 supply1=VDD0
B A2 supply0=VSS1 supply1=VDD1
```

this is the SPICE netlist result:

```
.SUBCKT A i i2
.ENDS

.SUBCKT B
XA1 A $PINS i=VDD0 i2=VSS0
XA2 A $PINS i=VDD1 i2=VSS1
XA3 A $PINS i=PWR i2=GND
.ENDS
```

Related Topics

[Power and Ground Connections](#)

Addition of Pins

You can add pins or ports using the v2lvs::add_pin command (command line: -addpins).

Using the following Verilog and SPICE netlists:

```
//---- file lib.v -----
// Verilog Library Module
// v2lvs::add_pin pins are added to calls to this module
module B ( P1,P2 );
  input P1;
  output P2;
endmodule

$S---- file lib.s -----
** Spice Library Module
**
** No VSS or VDD pins are defined in A0
** -addpin pins are not added to calls to this SUBCKT
.SUBCKT A0 P[3] P[2] P[1] P[0] Q
.ENDS

** VSS and VDD pins are declared here
** -addpin pins are added to calls to this SUBCKT
.SUBCKT A1 P[3] P[2] P[1] P[0] Q VSS VDD
.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
// v2lvs::add_pin pins are added to calls to undeclared modules.
UNDECL inst4 ( .P(a), .Q(a) );
// Undeclared gate-level primitive
// v2lvs::add_pin pins are added to gate instantiations in the order
// specified in the command script.
nand inst5( e, f, g );

endmodule
```

You can translate the file *ex.v* with *libraries lib.s* and *lib.v* by using the `v2lvs::add_pin` command to add VSS and VDD pins with the following command sequence:

```
v2lvs::load_verilog -filename ex.v
v2lvs::load_verilog -filename lib.v -lib_mode
v2lvs::load_spice -filename lib.s -range_mode
v2lvs::set_includes -filename lib.s
v2lvs::add_pin -pin VSS -pin VDD
v2lvs::write_output -filename ex.spi
exit
```

Warnings are generated for the UNDECL module and nand primitive because they are not defined. This has the following SPICE output:

```
.INCLUDE "lib.s"

.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
```

In Place and Route flows where a third-party provides SPICE library files for standard cells and you want to manage bulk pin connections, the `-addpins` command line option does not work. The `v2lvs::add_actual_port` and `v2lvs::add_formal_port` commands in the Tcl shell are suited to this purpose.

Example — Adding actual pins

This example shows the adding of actual pins. See “[Tcl Interface Command Encyclopedia](#)” on page 771 for the definition of “actual.”

Here are the input files. Verilog file:

```
///////////////////////////// test1.v Verilog file ///////////////////////////////
module test (a,b,x,clk,vddx,vdd_abc,vss);
    input a,b,clk,vdd_abc,vddx,vss;
    output x;
    inv u1 (.A(a), .Z(net1), .VDD(vddx), .VSS(vss));
    inv u2 (.A(b), .Z(iso_net), .VDD(vddx), .VSS(vss));
    isobf io1 (.A(net1), .Z(net2), .ISOB(iso_net), .VDD(vddx), .VDD1(vddx),
.VDD2(vdd_abc), .VSS(vss));
    inv u3 (.A(net2), .Z(x), .VDD(vdd_abc), .VSS(vss));
// PLL Macro
    pll pll_u1 (.ref_clk(net2), .en(x), .clk_out(clk), .VDD_1p8v(vdd_abc),
    .VDD(vddx), .VDBB(vdd_abc), .VSS(vss));
endmodule
//////////////////////////// End of Verilog file /////////////////////
```

SPICE library:

```
***** Included lib1.s SPICE library *****
.SUBCKT pll ref_clk en clk_out VDD_1p8v VDD VDDB VSS
.ENDS
.SUBCKT isobf VDD VSS VDD1 VDD2 A ISOB Z VDD1B VDD2B
.ENDS
.SUBCKT inv VDD VSS A Z VDBB
.ENDS
***** End of SPICE library *****
```

Command script:

```
##### v2lvs.command_1 file #####
# Connect the bulk signals to the correct supplies.
#RULE 1
v2lvs::add_actual_port -module * -inst * -if_formal_actual {VDD vddx} \
    -connect_formal_actual {VDBB vddx}
#RULE 2
v2lvs::add_actual_port -module * -inst * -if_formal_actual {VDD vdd_abc} \
    -connect_formal_actual {VDBB vdd_abc}
#RULE 3
v2lvs::add_actual_port -module * -inst * -if_formal_actual {VDD1 vddx} \
    -connect_formal_actual {VDD1B vddx}
#RULE 4
v2lvs::add_actual_port -module * -inst * -if_formal_actual {VDD2 vdd_abc} \
    -connect_formal_actual {VDD2B vdd_abc}
# Have to reconnect the pll instance bulk connection to the original net. \
# This could be solved by not using wildcard previously.
#RULE 5
v2lvs::add_actual_port -module * -inst pll_u1 \
    -connect_formal_actual {VDBB vdd_abc}
##### End of command file #####
```

Where the rules might conflict, the first rule in the file has priority.

The command sequence is this:

```
v2lvs::load_verilog -filename test1.v
v2lvs::load_spice -filename lib1.s -pin_mode
v2lvs::set_includes -filename lib1.s
v2lvs::do_source -filename v2lvs.command_1
v2lvs::write_output -filename test1.spi
exit
```

The expected SPICE output is this:

```
.INCLUDE "lib1.s"

.SUBCKT test a b x clk vddx vdd_abc vss
Xu1 inv $PINS A=a Z=net1 VDD=vddx VSS=vss VDDB=vddx $$ Due to RULE 1
Xu2 inv $PINS A=b Z=iso_net VDD=vddx VSS=vss VDDB=vddx $$ Due to RULE 1
Xio1 isobuf $PINS A=net1 Z=net2 ISOB=iso_net VDD=vddx VDD1=vddx
+ VDD2=vdd_abc VSS=vss VDD1B=vddx VDD2B=vdd_abc $$ Due to RULE 3 and 4
Xu3 inv $PINS A=net2 Z=x VDD=vdd_abc VSS=vss VDDB=vdd_abc $$ Due to RULE 5
Xpll_u1 pll $PINS ref_clk=net2 en=x clk_out=clk VDD_1p8v=vdd_abc VDD=vddx
+ VDDB=vdd_abc VSS=vss
.ENDS
```

Notice that RULE 1 and RULE 2 are conflicting rules for the instance statements Xu1 and Xu2. RULE1 has priority because it came first.

Example — Adding actual pins using a cell list

This example illustrates how to use a cell list. After grouping the subcircuits inv and isobuf in a list, all the `v2lvs::add_actual_port` rules that use the `-group` option are applicable to inv and isobuf.

Here are the input files. Verilog file:

```
///////////////////////////// test2.v Verilog file ///////////////////////
module test (a,b,x,clk,vddx,vdd_abc,vss);
    input a,b,clk,vdd_abc,vddx,vss;
    output x;
    inv u1 (.A(a), .Z(net1), .VDD(vddx), .VSS(vss));
    isobuf io1 (.A(net1), .Z(net2), .ISOB(iso_net), .VDD(vddx), .VDD1(vddx),
    .VDD2(vdd_abc), .VSS(vss));
// PLL Macro
    pll pll_u1 (.ref_clk(net2), .en(x), .clk_out(clk), .VDD_1p8v(vdd_abc),
    .VDD(vddx), .VDDB(vdd_abc), .VSS(vss));
endmodule
//////////////////////////// End of Verilog input file ///////////////////
```

SPICE library:

```
***** Included lib2.s SPICE library *****
.SUBCKT pll ref_clk en clk_out VDD_1p8v VDD VDDB VSS
.ENDS
.SUBCKT isobf VDD VSS VDD1 VDD2 A ISOB Z VDD1B VDD2B
.ENDS
.SUBCKT inv VDD VSS A Z VDDB
.ENDS
***** End of SPICE library *****
```

Command script:

```
##### v2lvs.command_2 file #####
# Define cells list
set cell_list [list inv isobf]
# Connect the bulk signals to the correct supplies for stdcells only.
#RULE 1
v2lvs::add_formal_port -module * -group $cell_list \
-if_formal_actual {VDD vddx} -connect_formal_actual {VDDB vddx}
#RULE 2
v2lvs::add_formal_port -module * -group $cell_list \
-if_formal_actual {VDD2 vdd_abc} -connect_formal_actual {VDD2B vdd_abc}
##### End of command file #####
```

The command sequence is the following:

```
v2lvs::load_verilog -filename test2.v
v2lvs::load_spice -filename lib2.s -pin_mode
v2lvs::set_includes -filename lib2.s
v2lvs::do_source -filename v2lvs.command_2
v2lvs::write_output -filename test2.spi
exit
```

The expected SPICE output is this:

```
.INCLUDE "lib2.s"
.SUBCKT test a b x clk vddx vdd_abc vss
Xu1 inv $PINS A=a Z=net1 VDD=vddx VSS=vss VDDB=vddx $$ Due to RULE 1
Xio1 isobf $PINS A=net1 Z=net2 ISOB=iso_net VDD=vddx VDD1=vddx
+ VDD2=vdd_abc VSS=vss VDD2B=vdd_abc $$ Due to RULE 2
Xpll_u1 pll $PINS ref_clk=net2 en=x clk_out=clk VDD_1p8v=vdd_abc VDD=vddx
+ VDDB=vdd_abc VSS=vss
.ENDS
```

Example — Adding actual and formal pins

This example shows the impact of adding pins across the hierarchy where pins vdd1, vdd2, and VDDB are added to different modules. It shows the use of the [v2lvs::add_formal_port](#) command. It also shows how [v2lvs::add_actual_port](#) can be used for a specific parent cell and a specific instance.

The following are the input files. Verilog file:

```
///////////////////////////// test3.v Verilog file ///////////////////////
module test (in,out1,out2);
    input in;
    output out1, out2;
    domain1 d1 (.a(in), .z(out1));
    domain2 d2 (.a(in), .z(out2));
endmodule

module domain1 (a,z);
    input a;
    output z;
    supply1 vdd1;
    supply0 vss;
    inv u1 (.A(a), .Z(net1), .VDD(vdd1), .VSS(vss));
    inv u2 (.A(net1), .Z(z), .VDD(vdd1), .VSS(vss));
endmodule

module domain2 (a,z);
    input a;
    output z;
    supply1 vdd2;
    supply0 vss;
    inv u1 (.A(a), .Z(net1), .VDD(vdd2), .VSS(vss));
    inv u2 (.A(net1), .Z(z), .VDD(vdd2), .VSS(vss));
endmodule
////////////////// End of Verilog input file /////////////////////
```

SPICE library:

```
***** Included lib3.s SPICE library *****
.SUBCKT pll ref_clk en clk_out VDD_1p8v VDD VDDB VSS
.ENDS
.SUBCKT isobf VDD VSS VDD1 VDD2 A ISOB Z VDD1B VDD2B
.ENDS
.SUBCKT inv VDD VSS A Z VDDB
.ENDS
***** End of SPICE library *****
```

Command file:

```
##### v2lvs.command_3 file #####
# This propagates the bulk signals and different power signals
# through the hierarchy.
# RULE 1
v2lvs::add_formal_port -port vdd1
# RULE 2
v2lvs::add_formal_port -port vdd2
# RULE 3
v2lvs::add_formal_port -port VDDB -under test
# Connect the bulk signals to the correct supplies based on module
# hierarchy.
# RULE 4
v2lvs::add_actual_port -module test -inst d1 \
    -connect_formal_actual {VDDB vdd1}
# RULE 5
v2lvs::add_actual_port -module test -inst d2 \
    -connect_formal_actual {VDDB vdd2}
##### End of command file #####
```

The command line is the following:

```
v2lvs::load_verilog -filename test3.v
v2lvs::load_spice -filename lib3.s -pin_mode
v2lvs::set_includes -filename lib3.s
v2lvs::do_source -filename v2lvs.command_3
v2lvs::write_output -filename test3.spi
exit
```

The expected SPICE output is this (comments added for clarity):

```
.INCLUDE "lib3.s"

.SUBCKT test in out1 out2 vdd1 vdd2
* vdd1 and vdd2 added due to RULE 1 and RULE 2
Xd1 domain1 $PINS a=in z=out1 vdd1=vdd1 VDDB=vdd1
* vdd1=vdd1 written due to RULE 1 and VDDB=vdd1 due to RULE 4
Xd2 domain2 $PINS a=in z=out2 vdd2=vdd2 VDDB=vdd2
* vdd2=vdd2 written due to RULE 2 and VDDB=vdd2 due to RULE 5
.ENDS

.SUBCKT domain1 a z vdd1 VDDB $$ VDDB added due to RULE 3
Xu1 inv $PINS A=a Z=net1 VDD=vdd1 VSS=vss VDDB=VDDB
* VDDB=VDDB written due to RULE 3
Xu2 inv $PINS A=net1 Z=z VDD=vdd1 VSS=vss VDDB=VDDB
* VDDB=VDDB written due to RULE 3
.ENDS

.SUBCKT domain2 a z vdd2 VDDB $$ VDDB added due to RULE 3
Xu1 inv $PINS A=a Z=net1 VDD=vdd2 VSS=vss VDDB=VDDB
* VDDB=VDDB written due to RULE 3
Xu2 inv $PINS A=net1 Z=z VDD=vdd2 VSS=vss VDDB=VDDB
* VDDB=VDDB written due to RULE 3
.ENDS

.GLOBAL vdd1
.GLOBAL vdd2
.GLOBAL vss
```

Management of Bulk Pin Connections

In typical Place and Route (PnR) flows, a SPICE library file for the standard cells is provided by a third-party IP vendor, another group within the production organization, or it could be a part of the design kit from the foundry. Footprints of these standard cells are used in the PnR flow to route at the top level. In the PnR flow, the standard cells are in a black-box format. The Verilog netlist is generated from the PnR tool to fill in the black boxes. Since PnR does not work at the transistor level, it does not have a pin for the bulk connection in the Verilog netlist.

Generally, the PnR environment does not manage bulk pin connections, so a means to add a pin to the Verilog module is needed. Bulk pins can then connect to the correct bulk nets. The bulk nets are also added to the supply net domains of the Verilog netlist from PnR and to instantiated module interfaces.

V2LVS has supported a -addpin option that adds a pin to subcircuit definitions and subcircuit calls in the output SPICE netlist. This is done where such pins do not exist in the input Verilog netlist or in relevant SPICE libraries.

In this example, -addpin fails to give the desired output. Net VDDB acts as a bulk pin for the buffer_cell subcircuit and the designer wants to connect VDDB to vcc2 in the test_top subcircuit.

Verilog File (*top.v*):

```
module test_top (in1, vcc1, vcc2, vss, out1);
    input in1;
    output out1;
    inout vcc1, vcc2, vss;
    wire wire_conn;
    buffer_cell \U_cell1/lsync (.INP1(in1), .OUT1(wire_conn), .VDD(vcc1),
    .VSS(vss));
    buffer_cell \U_cell2/lsync (.INP1(wire_conn), .OUT1(out1), .VDD(vcc1),
    .VSS(vss));
endmodule
```

SPICE library (*subckt_def.s*):

```
.SUBCKT buffer_cell INP1 OUT1 VDD VDBB VSS
M1 OUT1 INP1 VSS VSS nmos W=0.4e6 L=0.06e6
M2 OUT1 INP1 VDD VDBB pmos W=0.2e6 L=0.06e6
.ENDS buffer_cell
```

Desired output:

```
.INCLUDE "subckt_def.s"
.SUBCKT test_top in1 vcc1 vcc2 vss out1
XU_cell1/lsync buffer_cell $PINS INP1=in1 OUT1=wire_conn VDD=vcc1 VSS=vss
VDBB=vcc2
XU_cell2/lsync buffer_cell $PINS INP1=wire_conn OUT1=out1 VDD=vcc1 VSS=vss
VDBB=vcc2
.ENDS
```

Using the V2LVS command line interface with the **-addpin** option:

```
v2lvs -v top.v -lsp subckt_def.s -s subckt_def.s -addpin VDBB \
-o v2lvs_top.spi -log v2lvs_top.log
```

The output is this:

```
.INCLUDE "subckt_def.s"
.SUBCKT test_top in1 vcc1 vcc2 vss out1 VDBB
XU_cell1/lsync buffer_cell $PINS INP1=in1 OUT1=wire_conn VDD=vcc1 VSS=vss
+ VDBB=VDBB
XU_cell2/lsync buffer_cell $PINS INP1=wire_conn OUT1=out1 VDD=vcc1 VSS=vss
+ VDBB=VDBB
.ENDS
```

Notice that VDBB is connected to VDBB, as shown in the output. However, the designer wants to connect the VDBB pin of buffer_cell to the vcc2 port present in the cell test_top. The **-addpin** option does not do this.

The desired output can be produced by in the V2LVS Tcl interface using this command:

```
v2lvs::add_actual_port -module test_top -inst * -connect_formal_actual \
{ VDBB vcc2 }
```

In this command, the * is used as a wildcard to match all instances. The command searches for all instances in cell test_top, and in such instances, the pin VDDB (if VDDB is present in the child subcircuit) is connected to the port net vcc2 (if vcc2 is present in the cell test_top).

You may want to analyze the design and then use port/pin commands to make changes in the output. Using the V2LVS Tcl commands in a Tcl script can be useful in these cases.

For example, if you want to find all cells under a cell called TOP and store them in a variable, you can use this command in the Tcl shell:

```
set cell_list [v2lvs::find_module -under TOP]
```

Now, you can pass the cell_list variable to another command as an input:

```
v2lvs::add_formal_port -port VDDB -module $cell_list
```

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:

```
module AA ( OUT1, OUT2 );
  output OUT1, OUT2;
  assign OUT1=OUT2;
endmodule
```

is converted to the following SPICE netlist:

```
.SUBCKT AA OUT1 OUT2
*.CONNECT OUT1 OUT2
.ENDS
```

Related Topics

[Net Types](#)

Primitive Instances

IEEE Std 1364-2001 describes the use of primitive instances.

V2LVS recognizes the following gate-level primitives in Verilog:

Table 16-2. Gate-Level Primitives

and	nand	not
or	nor	notif0

Table 16-2. Gate-Level Primitives (cont.)

xor	xnor	notif1
buf	bufif0	bufif1
pulldown	pullup	rnmos
nmos	pmos	rpmos
cmos	rcmos	tran
rtran	tranif0	tranif1
rtranif0	rtranif1	

Gates are handled similarly to [Modules](#).

Ranges of gates are not supported (such as, 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 in the Verilog netlist must be present in the SPICE library in order for calls to the gates in the output SPICE netlist to resolve correctly. The corresponding Verilog and SPICE library circuits must have *identical pin ordering*.

Primitive gates may be declared like this:

```
and ( in, out, control );
```

Warnings are always issued by V2LVS for instantiation of gate-level primitives since these are not usually part of a purely structural netlist. The correct subcircuit call is nevertheless written to the output.

V2LVS generates 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).

Example — Use of Verilog primitive gates

The following Verilog code:

```
module A () ;
  wire a, b, c, d, e, f;
  and i1 ( a, b, c );
  xor (d, e, f );
endmodule
```

is converted into the following SPICE circuit:

```
.SUBCKT A
X11 a b c and
X0 d e f xor
.ENDS
```

Avoid Collisions with Gate-Level Primitives

You can use v2lvs::set_prefix (command line: -p) to add a prefix to all gate-level primitive calls.

The following Verilog code:

```
module A ();
  wire a, b, c, d, e, f;
  and i1 ( a, b, c );
  xor (d, e, f );
endmodule
```

is converted into the following SPICE circuit if you use “[v2lvs::set_prefix -prefix v2lvs_](#)”.

```
.SUBCKT A
Xi1 a b c v2lvs_and
X0 d e f v2lvs_xor
.ENDS
```

Use this option to avoid name collisions between Verilog gate level primitives and incompatible SPICE library subcircuits that have the same name.

Behavioral Statements

All behavioral statements cause the containing module to be skipped in translation, except for the assign statement, which applies continuous assignment. 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 following module 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;
  ...
endmodule
```

translates to the SPICE circuit:

```
.SUBCKT B a[0] a[1]
* .CONNECT x[0] b[0]
* .CONNECT x[1] b[1]
* .CONNECT w[0] a[0]
* .CONNECT w[1] a[1]
* .CONNECT d c[0]
* .CONNECT e c[1]
...
.ENDS
```

Other behavioral statements, including those in [Table 16-3](#), cause the module they are in to be skipped during translation.

Table 16-3. Behavioral Statements

initial	always	deassign
force	release	repeat
posedge	negedge	if-else-ifnone
case-endcase	casex	casez
default	forever	while
for	wait	disable
begin-end	fork-join	

Related Topics

[Modules](#)

Specify Block

A Specify block is used to specify timing information for paths across a Verilog module. V2LVS processes \$recrem system timing checks in Specify blocks.

Expressions

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 selection concatenations; and multiple concatenations.

- **Identifiers** — Regular and escaped identifiers are supported 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 because they can present problems throughout the flow as each tool handles 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 ([v2lvs::set_array_delimiter](#) applies). 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 to the same escaped identifier (for example, A is the same as \A).

To provide for handling identifiers in V2LVS conversion, the [v2lvs::preserve_back_slash](#)-enable option is useful. This command causes the leading backslash (\) character to be retained on escaped identifiers that are not permitted Verilog identifiers. This has the effect as ensuring bit 3 of A is not the same as \A[3].

Example — Handling escaped identifiers

This Verilog module:

```
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 v2lvs::preserve_back_slash -disable (the default) is used:

```
.SUBCKT DDD b[2] b[1] b[0] sum[2] sum[1] sum[0] ci
*.CONNECT b[2] b[1]
*.CONNECT b[1] b[0]
*.CONNECT sum[2] b[2]
*.CONNECT sum[0] b[1]
XA1 A $PINS i=b[2] zn=sum[1] ci=ci
.ENDS

.SUBCKT A i zn ci
.ENDS
```

It translates as follows when v2lvs::preserve_back_slash -enable is used:

```
.SUBCKT DDD b[2] b[1] b[0] sum[2] sum[1] sum[0] ci
*.CONNECT \b[2] b[1]
*.CONNECT \b[1] b[0]
*.CONNECT sum[2] \b[2]
*.CONNECT sum[0] \b[1]
XA1 A $PINS i=\b[2] zn=sum[1] ci=ci
.ENDS

.SUBCKT A i zn ci
.ENDS
```

In V2LVS, most escaped identifiers are mapped directly to SPICE identifiers since SPICE supports most ASCII characters in its identifiers. However, some characters cannot be translated directly to SPICE. Leading dollar sign (\$), comma (,), equals sign (=), and forward slash (/) are escaped identifier characters which, under certain circumstances, are disallowed in SPICE. The first three characters in this list are replaced by the hash (#) character by default. Additional hashes can be added to make generated names unique, as necessary. However, if a generated name collides with an existing (non-generated) name, this causes an error.

Any / characters are left as-is in V2LVS by default. However, some Verilog escaped identifier names can produce undesired effects. The [v2lvs::substitute_chars](#) command (command line: -c) mitigates the issue.

- **Unsupported expressions** — Function calls and real numbers are not supported. Ternary expressions (?:) are not supported in any structural application.
- **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. Concatenation of constant expressions, module path expressions, and nets are also respectively supported.
- **Parameters in expressions** — Parameters may be referenced in expressions, provided they return a constant value that can be evaluated in a structural context.

For example, the following module:

```
module A ( inA );
parameter width=8;
input [width-1:0] inA;
endmodule
```

is translated into the following subcircuit:

```
.SUBCKT A inA[7]  inA[6]  inA[5]  inA[4]  inA[3]  inA[2]  inA[1]
+ inA[0]
.ENDS
```

The following uses of parameters are supported:

- Parameter and local parameter declarations.
- Signed property with local parameter.
- List of parameter declarations in the module declaration line.

Example:

```
parameter integer width = 8;
parameter signed [5:0] p3 = 3;
module SUB #(parameter p = 2) (out1, in1, in2);
module top #(parameter signed num1 = 24, parameter real num2 = num1)
( input wire [7:0] in, output [5:0] out1);
```

- Explicit parameter support in module instantiation. For example:

```
vdff #(.size(8),.delay(12)) mod_c(out_c, in_c, clk);
```

Unary and Binary Expressions

Unary and binary expressions are often used for behavioral syntax and cause the module they are contained in to be skipped in translation with a warning.

Some binary expressions may be used in port declarations to specify range values. The supported binary expressions include + - * ** / % < =< > >= == != && ||.

The + and - operators may be used in range expressions, such as the following code:

```
assign b1[1+:4] = a[1];
sub i1[1:0] (.a(tp1[11:-8]), .b_i({tp2[2-:3], 1'b0}));
```

Some binary expressions may be used with bit expressions. The supported binary expressions for use with bit expressions include + - < =< > >= == != && ||.

The signed shift operators >>> and <<< are treated as >> and <<, respectively.

Miscellaneous Expression Handling

These expressions are all supported.

- Event triggers (with -> operator) can take a hierarchical identifier. For example:

```
if ( in1[4:1] == 4'b1010 )
    -> top.a.trig1
```

- Event expressions can take hierarchical identifiers.
- Variable assignments can be performed on hierarchical paths.
- Expressions with constant-range (as in assignment to a slice).
- Initialization during declaration, for example: reg a = 0;.
- UDP instantiation in module items and generate items.
- Arrayed identifiers and escaped arrayed identifiers.
- Comma-separated sensitivity list.
- @*, @(*) event control statements.

Compiler Directives

V2LVS supports a limited set of compiler directives.

`define	`else	`elsif
`endif	`ifdef	`ifndef
`include	`line	`undef

All other compiler directives are ignored to the end of the line where they appear.

V2LVS does not support the nested form `define (`define a(b,c) ...). `define directives have global scope.

Arguments to `include statements that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment. SPICE files referenced in `include statements behave similarly, as well as any SPICE netlists referenced recursively from the included SPICE netlists in Verilog file.

Library Files

V2LVS can use a Verilog library file to specify declarations to represent leaf modules that are defined in SPICE. V2LVS reads the Verilog library file to gather the port interface names that are used during instantiation. The Verilog library file may contain full Verilog modules, including user-defined primitives and behavioral syntax.

Library files are loaded using the [v2lvs::load_verilog -lib_mode](#) command (command line: `-l`).

It is possible to use V2LVS without a Verilog library file as the information contained in the Verilog library file is not strictly necessary for mapping Verilog to SPICE. This is discussed in “[V2LVS Used Without Verilog or SPICE Library Files](#)” on page 758.

V2LVS Used With SPICE Library Files [**751**](#)

V2LVS Used Without Verilog or SPICE Library Files [**758**](#)

V2LVS Used With SPICE Library Files

SPICE library files may be used to define lower-level modules. V2LVS reads SPICE library files in order to deduce Verilog module interfaces, and to establish appropriate connectivity to SPICE library subcircuits from the V2LVS output netlist.

The SPICE library flow 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.

This flow can be used in conjunction with the Verilog library modules ([v2lvs::load_verilog-lib_mode](#)).

Consider the following SPICE subcircuit and corresponding Verilog module:

```
.SUBCKT EX1 A B C
.ENDS

module EX1 ( A, C, B );
endmodule
```

These descriptions have an inconsistent pin order (pins B and C are swapped). This Verilog call:

```
EX1 e( w1, w2, w3 );
```

produces different results based on whether the Verilog library or the SPICE library is used to define EX1. The SPICE library generates the following SPICE call in the V2LVS output by default:

```
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 made correctly as long as [v2lvs::generate_ordered_pins -enable](#) is not used. This is because the \$PINS connection creates a named connection in SPICE. (The [v2lvs::generate_ordered_pins -enable](#) command does not output the \$PINS construct, which is not recognized by many tools outside of the Calibre line.)

Note

 The pin order in SPICE library files versus Verilog modules is important to consider when positional Verilog calls are made, and when [v2lvs::generate_ordered_pins -enable](#) is used, because V2LVS connects pins in Verilog order in this configuration.

Range Mode	752
Pin Mode.....	753

Enable Subcircuit Scoping	754
*.BUSDELIMITER Statement Handling	756

Range Mode

The v2lvs::load_spice -range_mode option (command line: -lsr) causes the SPICE library 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. Any .INCLUDE statements are processed. Multiple instances of commands using -range_mode are allowed.

Example — Range mode

This SPICE library subcircuit:

```
.SUBCKT BLK_A A[0] A[1] A[2] A[3] B
...
.ENDS
```

is interpreted as a Verilog module with this signature when using [v2lvs::load_spice -range_mode](#):

```
module BLK_A ( A, B );
  inout [0:3]A;
  inout B ;
  ...
endmodule
```

Use -range_mode if Verilog module calls to BLK_A treat the port A as a port range. An example of a call that treats them as a port range is this:

```
module test;
  wire [0:3] w1;
  wire w2;
  // wire bundle w1 is passed as a unit to port A.
  BLK_A instA ( .A( w1 ), .B( w2 ) );
endmodule
```

With -range_mode, the output would be like this:

```
.SUBCKT test
XinstA BLK_A $PINS A[0]=w1[0] A[1]=w1[1] A[2]=w1[2] A[3]=w1[3] B=w2
.ENDS
```

The -range_mode option is sensitive to [v2lvs::set_array_delimiter](#) characters.

The v2lvs::load_spice -detect_bus_delimiter option can be useful with -range_mode when the bus delimiter character in the library files is not the default ([]). The -detect_bus_delimiter option automatically detects the bus delimiter characters and writes the appropriate *.BUSDELIMITER statements into the output netlist.

The [v2lvs::combine_interface_info](#) command is useful in combining the bus delimiter character from a SPICE library subcircuit that matches a Verilog library module with the pin order from the corresponding Verilog module.

Pin Mode

The `v2lvs::load_spice -pin_mode` option (command line: `-lsp`) 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. Any `.INCLUDE` statements are processed. Multiple instances of commands using `-pin_mode` are allowed.

Example — Pin mode

This SPICE library subcircuit:

```
.SUBCKT BLK_A A[0] A[1] A[2] A[3] B
...
.ENDS
```

is interpreted as a Verilog module with this signature when using `v2lvs::load_spice -pin_mode`:

```
module BLK_A ( \A[0] , \A[1] , \A[2] , \A[3] , B );
  inout \A[0] ;
  inout \A[1] ;
  inout \A[2] ;
  inout \A[3] ;
  inout B ;
endmodule
```

Use `-pin_mode` if Verilog module calls to `BLK_A` treat the `A` ports as individual pins. An example of a call that treats them as individual pins is this:

```
module test;
  wire [0:3] w1;
  wire w2;
  // wire bundle w1 is passed using pin select notation to individual
  // pins of A.
  BLK_A instA (
    .\A[0] ( w1[0] ),
    .\A[1] ( w1[1] ),
    .\A[2] ( w1[2] ),
    .\A[3] ( w1[3] ),
    .B( w2 )
  );
endmodule
```

With `-pin_mode`, the output would be like this:

```
.SUBCKT test
XinstA BLK_A $PINS A[0]=w1[0] A[1]=w1[1] A[2]=w1[2] A[3]=w1[3] B=w2
.ENDS
```

Enable Subcircuit Scoping

V2LVS supports a configuration file that allows subcircuit scoping of lower-level SPICE blocks that are incorporated into a top-level Verilog design.

This uses the SPICE scoping method described under “[Block-Level Chip Assembly Using Scoped SPICE Subcircuits](#)” on page 705. V2LVS implements this scoping method through the `v2lvs::set_config_file` command (command line: `-cfg`).

The configuration file has the following format:

```
<custom_cell_filename> <top_subckt> <wrapper_subckt> {PIN | RANGE}
```

The entries in the file are defined as follows:

custom_cell_filename — The SPICE file that includes the block-level subcircuit definition that V2LVS uses for the scoped block.

top_subckt — The name of the block-level subcircuit definition that exists in the ***custom_cell_filename*** file. This is the top-level cell that is being incorporated into the Verilog file.

wrapper_subckt — The wrapper name for the top-level cell that will be represented as a Verilog signature with the top-level cell instantiated inside it. The wrapper name can be same as the ***top_subckt*** name, although this prevents use of this name as an hcell.

PIN — Pin mode is used, similar to `v2lvs::load_spice -pin_mode`.

RANGE — Range mode is used, similar to `v2lvs::load_spice -range_mode`.

For example, consider the following top-level Verilog netlist:

```
//----- scoped.v -----
module TOP ( port_A, port_B, port_C );
inout [1:0] port_A;
inout port_B, port_C;

// instances of BLK1 - port_A is passed as an array (RANGE MODE)
BLK1 B1_0 ( port_A, port_B, port_C );
BLK1 B1_1 ( .A(port_A), .B(port_B), .C(port_C) );

// instances of BLK2 - port_A is passed as individual pins (PIN MODE)
BLK2 B2_0 ( port_A[1], port_A[0], port_B, port_C );
BLK2 B2_1 (
    .A[1] (port_A[1]),
    .A[0] (port_A[0]),
    .B(port_B),
    .C(port_C)
);
endmodule
```

The following configuration file could be used to incorporate these three separate blocks in V2LVS:

```
blk1.cdl BLK1 BLK1_WRAPPER RANGE
blk2.cdl BLK2 BLK2_WRAPPER PIN
```

Assume the following SPICE library files:

```
* blk1.cdl
.SUBCKT A P2 P1
C0 P1 P2 40
.ENDS

.SUBCKT BLK1 A[1] A[0] B C
XA0 A[1] C A
XA1 A[0] B A
.ENDS

* blk2.cdl
.SUBCKT A P2 P1
R0 P1 P2
.ENDS

.SUBCKT BLK2 A[1] A[0] B C
XA0 A[1] C A
XA1 A[0] B A
.ENDS
```

The following V2LVS command sequence is used:

```
v2lvs::set_config_file -filename scoped.cfg
v2lvs::load_verilog -filename scoped.v
v2lvs::write_output -filename scoped.out
```

This generates the following SPICE netlist:

```
$$---- scoped.out -----
$$ Spice netlist generated by v2lvs

.SUBCKT TOP port_A[1] port_A[0] port_B port_C
XB1_0 BLK1_WRAPPER $PINS A[1]=port_A[1] A[0]=port_A[0] B=port_B C=port_C
XB1_1 BLK1_WRAPPER $PINS A[1]=port_A[1] A[0]=port_A[0] B=port_B C=port_C
XB2_0 BLK2_WRAPPER $PINS A[1]=port_A[1] A[0]=port_A[0] B=port_B C=port_C
XB2_1 BLK2_WRAPPER $PINS A[1]=port_A[1] A[0]=port_A[0] B=port_B C=port_C
.ENDS

.SUBCKT BLK1_WRAPPER A[1] A[0] B C
.INCLUDE blk1.cdl
X0 A[1] A[0] B C BLK1
.ENDS

.SUBCKT BLK2_WRAPPER A[1] A[0] B C
.INCLUDE blk2.cdl
X0 A[1] A[0] B C BLK2
.ENDS
```

*.BUSDELIMITER Statement Handling

The *.BUSDELIMITER SPICE statement specifies the character that identifies the index portion of SPICE pin names. The indexed pins indicate the SPICE equivalent of a Verilog port specified with a range expression.

Consider the following SPICE library:

```
* .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:

```
module TOP;
wire [3:0] w1;
wire [3:0] w2;
A A1 (.I1(w1));
B B1 (.I2(w2));
endmodule
```

When a SPICE library is specified with `v2lvs::load_spice -range_mode`, 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. Using `-range_mode` creates the following Verilog interface interpretation of the previous SPICE library:

```
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
```

The next examples show the use of the DECREASING and INCREASING keywords.

The following SPICE library:

```
* .BUSDELIMITER < DECREASING
.SUBCKT TOP1 I1<2> I1<3> I1<1> I1<0>
.ENDS
```

is equivalent to the following Verilog module:

```
module TOP1 ( I1 );
inout [3:0] I1;
endmodule
```

The following SPICE library:

```
*.BUSDELIMITER < INCREASING
.SUBCKT TOP2 I1<2> I1<3> I1<1> I1<0>
.ENDS
```

is equivalent to the following Verilog module:

```
module TOP2 ( I1 );
inout [0:3] I1;
endmodule
```

The v2lvs::load_spice -detect_bus_delimiter option finds the bus delimiter character automatically in a SPICE library subcircuit and causes a *.BUSDELIMITER statement that covers the included SPICE netlist in the output. The [v2lvs::combine_interface_info](#) can also be useful in such applications.

V2LVS Used 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.....	758
Instances of Undeclared SPICE Primitive Modules with Ordered Ports.....	759
Potential Errors When Using V2LVS Without Verilog Libraries	759

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

This Verilog code:

```
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

This Verilog code:

```
wire [3:0] in1;
wire in2, in3;
spice_module instance_aa ( .A(in1), .B(in2), .C(in3) );
```

is mapped to the SPICE instance:

```
Xinstance_aa spice_module $PINS A[3]=in1[3] A[2]=in[2] A[1]=in[1]
+ A[0]=in[0] B=in2 C=in3
```

Instances of Undeclared SPICE Primitive Modules with Ordered Ports

When a module is instantiated with ordered ports, the order in the Verilog instance is assumed to be the same order as the SPICE module definition.

Example — Positional call to undeclared primitive module

This Verilog code:

```
spice_module instance_aa ( in1, in2, in3 );
```

is mapped to the SPICE instance:

```
Xinstance_aa in1 in2 in3 spice_module
```

As with any undeclared module, a warning is issued during the run.

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. Calibre 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).

Potential Errors When Using V2LVS Without Verilog Libraries

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.

Related Topics

[V2LVS Used Without Verilog or SPICE Library Files](#)

Simulation Output Generation

By default, V2LVS utilizes a Calibre nmLVS comment-coded extension (\$PINS) to make pin connections. Such connections are independent of port order in the SPICE. However, the \$PINS construct is unique to Calibre flows, and some tools may not recognize it.

The [v2lvs::generate_ordered_pins](#) -enable option (command line: -i) generates standard SPICE output that is acceptable to many SPICE simulators. This output may also be used in conjunction with Calibre xRC generated netlists to perform detailed simulations of critical nets using SPICE-based simulators.

Note

 V2LVS is intended to translate Verilog netlists for use with Calibre nmLVS. The v2lvs::generate_ordered_pins command 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.

Since named connections are not possible in SPICE without use of an extension like \$PINS, a SPICE library (using [v2lvs::load_spice](#) -range_mode or -pin_mode) is useful when v2lvs::generate_ordered_pins -enable is used. This is so V2LVS can determine the pin correspondence between the Verilog netlist and the SPICE library.

V2LVS does not read the SPICE library file specified with [v2lvs::set_includes](#) (command line: -s). The v2lvs::set_includes command merely instructs V2LVS to issue a .INCLUDE statement at the start of its SPICE output. The v2lvs::load_spice -range_mode or -pin_mode options are used to read SPICE library files to determine pin format and sequence.

Inconsistencies in pin configuration between the Verilog and SPICE libraries are not detected by V2LVS. (Pin inconsistency is detected by LVS.) Inconsistencies can be addressed when using [v2lvs::combine_interface_info](#) -enable, but this uses the \$PINS construct, which may not be acceptable to circuit simulators.

Example — Generate ordered pins for standard SPICE output

File *src.v*:

```
// Verilog source
module top ();
  wire w1, w2, w3;
  A inst1( .in1(w1), .in2(w2), .out3(w3) );
endmodule
```

File *lib.v*:

```
// Verilog library
module A ( in1, in2, out3 );
  input in1, in2;
  output out3;
endmodule
```

File *lib.spi*:

```
$ SPICE library with pin order that does not match the Verilog library:
.SUBCKT A out3 in2 in1
.ENDS
```

Run V2LVS with default (\$PINS) output using this script:

```
v2lvs::load_verilog -filename src.v
v2lvs::load_verilog -filename lib.v -lib_mode
v2lvs::set_includes -filename lib.spi
v2lvs::write_output -filename src_unordered_pins.spi
exit
```

and with v2lvs::generate_ordered_pins -enable (standard SPICE pins) with this script:

```
# NOTE THE FOLLOWING LINE IS COMMENTED OUT
# v2lvs::load_spice -filename lib.spi -pin_mode
v2lvs::generate_ordered_pins -enable
v2lvs::load_verilog -filename src.v
v2lvs::load_verilog -filename lib.v -lib_mode
v2lvs::set_includes -filename lib.spi
v2lvs::write_output -filename src_ordered_pins.spi
exit
```

The respective output netlists are as follows:

```
$$ unordered pins
.INCLUDE "lib.spi"

.SUBCKT top
Xinst1 A $PINS in1=w1 in2=w2 out3=w3
.ENDS

$$ ordered pins
.INCLUDE "lib.spi"

.SUBCKT top
Xinst1 w1 w2 w3 A      $$ w1 <--> out3 w3 <--> in1 in lib.spi
.ENDS
```

Recall this line from *lib.spi*:

```
.SUBCKT A out3 in2 in1
```

Rule file:

```
// This LVS rule file compares $PINS output with
// v2lvs::generate_ordered_pins output
SOURCE PATH src_unordered_pins.spi
SOURCE PRIMARY top
SOURCE SYSTEM SPICE

LAYOUT PATH src_ordered_pins.spi
LAYOUT PRIMARY top
LAYOUT SYSTEM SPICE

LVS REPORT lvs.rep
```

Execute:

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

The LVS status is INCORRECT due to the pin order mismatch in Xinst1.

Uncommenting the v2lvs::load_spice command in the second script causes the LVS to be CORRECT in this case because the corresponding port order in *lib.spi* is matched in the output.

Calibre xRC Source Template File

The v2lvs::create_pex_template command (command line: -t) may be used to generate a file which Calibre xRC can use to determine directions of ports declared in the Verilog netlist.

When [v2lvs::create_pex_template](#) is used, V2LVS generates a file named *<dir>/template/%source.stl*. The *<dir>* is configured as a Mask SVDB Directory. Use the XRC option for that statement.

This file contains a single starting line:

```
%%SOURCE
```

followed by a .stl format template for each module in the Verilog netlist. The module template consists of a module name declaration line:

```
% <module_name> <module_name>
```

followed by one line per interface pin:

```
<pin_name> <pin_name> 0 <io_spec>
```

Where *<pin_name>* is the expanded pin name used in the SPICE netlist output (that is, busses are expanded and *<io_spec>* is one of i (input), o (output), io (inout).

For example, this Verilog netlist:

```
module A (X, Y, Z);
  input X;
  output Y;
  inout [0:1] Z;
endmodule

module B(V, W);
  output V;
  input W;
endmodule
```

produces this source template file:

```
%%SOURCE
% B B
V V 0 o
W W 0 i
% A A
X X 0 i
Y Y 0 o
Z[0] Z[0] 0 io
Z[1] Z[1] 0 io
```

Calibre xRC can read the *%source.stl* file in the same way that it reads individual *layout.stl* files when they are available. See the [Calibre xRC User's Manual](#) for more information regarding templates.

Collecting SPICE .INCLUDE Netlists into a Single Netlist

In certain cases, a SPICE library may have numerous .INCLUDE statements that encompass all the netlists of the design. It may be desirable to collect all of these netlists into a single library netlist. The following script performs this function. V2LVS does not need to be part of the tool flow for this script to be useful.

Procedure

- Enter the following script into a text editor:

```

# v2lvs Tcl script that collects SPICE files from .INCLUDE
# statements and deposits them into a single top-level file.
#
package require fileutil

# <spice_input_file> is the top-level SPICE file.
# <primary_subcircuit> is the top-level subcircuit name.
# <spice_output_file> is the name of the output netlist.
set usage "usage: v2lvs -tcl collect_spice.tcl -- \
<spice_input_file> <primary_subcircuit> <spice_output_file>

# there must be three arguments to the command.
if { $argc != 3 } {
    error $usage
}

# set up variable array.
set params(file) [ lindex $argv 0 ]
set params(primary) [ lindex $argv 1 ]
set params(out) [ lindex $argv 2 ]
# unset these for debugging purposes.
# puts "Parameters file: $params(file) "
# puts "Parameters primary: $params(primary) "
# puts "Parameters out: $params(out) "

set my_pid [ pid ]
set fake_v_filename [ ::fileutil::tempfile "collect_spice." ]
set fake_v [ open $fake_v_filename "w" ]

puts $fake_v "module fake_calibre_v2lvs_top_level_module ();"
puts $fake_v "$params(primary) fake1 ( );"
puts $fake_v "endmodule"
close $fake_v

# collect the SPICE and write the aggregate netlist
v2lvs::load_spice -filename $params(file) -pin_mode
v2lvs::load_verilog -filename $fake_v_filename
v2lvs::write_output -filename "/dev/null" -outslib $params(out)
file delete $fake_v_filename

exit 0

```

- Save the script as *collect_spice.tcl*.
- Run the script as follows, where *top.spi* is the top-level netlist name and TOP is the primary subcircuit name.

```
v2lvs -tcl collect_spice.tcl -- top.spi TOP top_collected.spi
```

Results

The output is an aggregate SPICE netlist containing all library netlists. The output netlist will contain a *.BUSDELIMITER statement that defines the bus delimiter character.

Creating LVS Box Subcircuits

The v2lvs::generate_empty_subckts -enable option (command line: -e) generates 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.

Prerequisites

- Verilog structural netlist.
- Verilog primitive library file.

Procedure

1. Generate a SPICE netlist using the Verilog design and primitive library file:

```
v2lvs::load_verilog -filename model.v
v2lvs::load_verilog -filename lib.vlib -lib_mode
v2lvs::write_output -filename model.spi
```

2. Generate a SPICE netlist from the Verilog library file:

```
v2lvs::load_verilog -filename lib.vlib
v2lvs::generate_empty_subckts -enable
v2lvs::write_output -filename lib.spi
```

3. Specify the **LVS Box** statement for the layout cells that correspond to the subcircuits in the Verilog library file (*lib.vlib* and *lib.spi*).

4. Include *lib.spi* from *model.spi*:

```
$$ model.spi file
.include lib.spi
```

Results

The *model.spi* file is a SPICE netlist with all the pin interfaces to the library modules being provided, but not the translations of the library modules themselves. The *lib.spi* file has empty subcircuit definitions for library primitives. These empty subcircuits can be treated as LVS Box cells in a hierarchical LVS run.

V2LVS Tcl Interface

The V2LVS Tcl interface supports programmability of the Verilog-to-SPICE translation process through standard Tcl language features. The interface has a Tcl pre-processor similar to other Calibre tools.

All of the historical command line interface capabilities are present in the Tcl interface. In addition, the Tcl interface provides commands to bias bulk pins in a Place and Route (PnR) environment.

The V2LVS Tcl commands can be used in one of two primary modes: interactive or script.

To use the interactive Tcl interface, use the `-tcl` option as follows:

```
v2lvs [ <options> ] -tcl
```

This starts a Tcl shell session in which you can enter V2LVS Tcl commands interactively. You do not have to specify any options other than `-tcl` to enter this mode. The command line options can be any of the historical V2LVS options, and these are executed when the shell starts.

Executing the `v2lvs::do_source` command in the shell runs a Tcl script specified by that command. The `v2lvs::do_source` command is recommended instead of the Tcl `source` command because `v2lvs::do_source` provides error output that is easier to debug.

To use a Tcl script upon invocation of the tool, use the `-tcl` option with the script name as an argument, as follows:

```
v2lvs [ <options> ] -tcl <script_name>
```

Using this command, any command line options and the specified Tcl script are processed and executed in a single step.

Conflicts between any specified command line options and commands executed in the Tcl shell generate warning or error messages. For options that can only be specified only once, like `-o`, `-s1`, and so forth, those specifications cannot be changed in the Tcl shell. For options that can be specified more than once, like `-v`, additional related commands can be issued in the Tcl shell.

Tcl Interface and Command Line Option Correspondence	766
Tcl Basics	770
Tcl Interface Command Encyclopedia	771

Tcl Interface and Command Line Option Correspondence

Most of the Tcl interface commands correspond with command line options from the command line interface.

Command line options are discussed under “[V2LVS Command Line Syntax](#)” on page 836.

Table 16-4. Tcl Command Correspondence

Command Line Option	Corresponding Tcl Command	Description
-a	v2lvs::set_array_delimiter	Specifies array delimiters.
-addpin	v2lvs::add_pin	Adds pins to subcircuit calls and definitions where such pins do not exist in the Verilog or SPICE inputs.
-b	v2lvs::preserve_back_slash	Controls how backslash () characters are handled in escaped identifiers.
-c	v2lvs::substitute_chars	Specifies character substitutions.
-cb	None.	Uses Calibre Cell/Block license.
-cfg	v2lvs::set_config_file	Specifies a configuration file that allows scoping of lower-level SPICE blocks incorporated into the top-level Verilog design.
-e	v2lvs::generate_empty_subckts	Controls empty .SUBCKT output.
-h	v2lvs::help	Shows the command usage description.
-i	v2lvs::generate_ordered_pins	Specifies whether pin netlisting using traditional SPICE or \$PINS constructs.
-ictrace	None.	Uses an ICtrace license.
-incvdir	v2lvs::include_dir	Specifies a directory for `include files.
-l	v2lvs::load_verilog -lib_mode	Specifies a Verilog library file.
-log	None.	Specifies a run transcript file.
-lsp	v2lvs::load_spice -pin_mode	Specifies a SPICE library file with translation of SPICE pins as Verilog ports without assembling arrays.
-lsr	v2lvs::load_spice -range_mode	Specifies a SPICE library file with translation of SPICE pins as Verilog ranged ports or arrays.
-n	v2lvs::number_unconnected_pins	Specifies whether unconnected pins receive numbered connections.

Table 16-4. Tcl Command Correspondence (cont.)

Command Line Option	Corresponding Tcl Command	Description
-o	v2lvs::write_output	Specifies to output a translated SPICE netlist of the input Verilog design. Optionally specifies the name of a consolidated SPICE library file.
-p	v2lvs::set_prefix	Specifies that gate-level primitives receive a prefix.
-rnt	v2lvs::write_output -renametext	Specifies to output a renamed text file that is included with the Calibre rule set. The rules in the renamed text file avoid text shorts in layout feedthrough cells.
-s	v2lvs::set_includes	Specifies a SPICE netlist to be referenced in a .INCLUDE statement in the output.
-s0	v2lvs::override_globals -supply0	Specifies the default global ground name.
-s1	v2lvs::override_globals -supply1	Specifies the default global power name.
-sk	v2lvs::override_globals -supply1 -supply_not_connected	Specifies supply0 and supply1 nets are not connected to global power and ground.
-sl	v2lvs::override_globals -supply1 -use_local_supply	Specifies to use non-global net names for supply0 and supply1 nets.
-sn	v2lvs::override_globals -supply1 -default_not_connected	Specifies default power and ground nets are not connected to global nets.
-so	v2lvs::override_globals -supply1 -custom_override	Specifies a filename containing supply net overrides.
-t	v2lvs::create_pex_template	Writes a Calibre xRC source template file.
-tcl	None.	Specifies to open a Tcl shell or to execute a Tcl command script.
-u	v2lvs::set_unnamed_pin_prefix	Specifies a prefix to add to various unnamed instantiations.
-v	v2lvs::load_verilog	Specifies Verilog netlist. This is a required option or command.
-V1995	v2lvs::verilog_version -version 1995	Specifies to use Verilog 1995 syntax.

Table 16-4. Tcl Command Correspondence (cont.)

Command Line Option	Corresponding Tcl Command	Description
-V2001	v2lvs::verilog_version -version 2001	Specifies to use Verilog 2001 syntax.
-w	v2lvs::set_warning_level	Specifies the verbosity of warning messages issued by the tool.
-werror	v2lvs::warning_as_error	Specifies to treat warnings as errors.

The following Tcl commands have no corresponding command line options:

Table 16-5. Tcl Commands With No Corresponding Command Line Option

Tcl Command	Description
exit	Quits the Tcl shell. Outputs files from V2LVS are written when this command is executed.
v2lvs::add_actual_port	Specifies to add an <i>actual</i> port/pin.
v2lvs::add_formal_port	Specifies to add a <i>formal</i> port/pin.
v2lvs::combine_interface_info	Specifies whether to prefer Verilog library pin order over the SPICE library for the same macro.
v2lvs::do_source	Specifies to read and execute a Tcl script.
v2lvs::connect_missing_ports	Specifies whether to write the connection for missing ports.
v2lvs::find_module	Creates an list of Verilog modules.
v2lvs::get_includes	Returns a list of Verilog `include filenames.
v2lvs::get_ports	Returns a list of port names for a module.
v2lvs::insert_port_instance	Inserts an instance inside a module and connects adds a pin.
v2lvs::override_assign	Overrides an assign statement in a Verilog module with a specified SPICE subcircuit.
v2lvs::rename_duplicate_cells	Resolves duplicate subcircuit names in SPICE libraries.
v2lvs::set_verbose_level	Specifies the verbosity of debug messages issued by the tool.
v2lvs::show_properties	Specifies to write instance properties.
v2lvs::write_file_info	Specifies whether to write informational comments to the output SPICE netlist.
v2lvs::write_pin_info	Specifies whether to write *.PININFO comments to the output netlist.

Tcl Basics

Tcl has a number of foundational ideas that govern all programs.

- **Tcl is case sensitive.**

The core set of Tcl commands are all lower case, as are the Calibre V2LVS extensions.

- **Order matters.**

Statements are executed sequentially within a procedure in the order they appear from top to bottom in the file.

- **Many characters have special meanings in Tcl.**

The following are the special characters used in this document:

Table 16-6. Tcl Special Characters

Character	Meaning
;	The semicolon terminates the previous command, allowing you to place more than one command on the same line.
\	Causes backslash substitution. Normally this is used to remove the special meaning of a metacharacter. Be careful not to have whitespace on the same line after a backslash that terminates a line, as this can cause Tcl interpretation errors.
\n	The backslash with the letter “n” creates a new line.
\$	The dollar sign in front of a variable name instructs the Tcl interpreter to access the value stored in the variable.
[]	Brackets cause command substitution, instructing the Tcl interpreter to treat everything within the brackets as the result of a command. Command substitution can be nested within words.
{ }	Braces instruct the Tcl interpreter to treat the enclosed words as a single string. The Tcl interpreter accepts the string as-is, without performing any variable substitution or evaluation.
“ ”	Quotes instruct the Tcl interpreter to treat the enclosed words as a single string. However, when the Tcl interpreter encounters variables or commands within string in quotes, it evaluates the variables and commands to generate the string.
#	The hash character denotes a comment. It must be the first non-white space character at the start of a command and it must be followed by white space. Comments are parsed as commands, so if they follow a command, they must be preceded by a semicolon (;).

- **The command to terminate a program is “exit”.**

Tcl Interface Command Encyclopedia

Throughout the command encyclopedia, the brace characters “{ }” indicate choices between required options. The bracket characters “[]” indicate non-required options. Both of these characters have special meaning in Tcl so it is important not to include these markup characters as part of the literal command syntax in a script.

These are two important definitions:

actual port (or actual name) — port used in the instantiation of a Verilog module.

formal port (or formal name) — name used in the declaration of a Verilog module.

These constructs appear as follows in Verilog:

formal port(actual port)

Suppose you have this Verilog file:

```
module test (in,out1,out2);
    input in;
    output out1, out2;
    domain inst (.a(in), .z(out1));
endmodule

module domain (a,z);
    input a;
    output z;
    supply1 vdd1;
    supply0 vss;
    inv u1 (.A(a), .Z(net1), .VDD(vdd1), .VSS(vss));
    inv u2 (.A(net1), .Z(z), .VDD(vdd1), .VSS(vss));
endmodule
```

For the instance “inst”, which is of type “domain”, “in” and “out1” are *actual* ports. The ports “a” and “z” specified in the definition of module “domain” are *formal* ports.

v2lvs::add_actual_port

Netlist configuration command.

Specifies to add an actual port/pin.

Usage

```
v2lvs::add_actual_port { -module module_name -inst instance_name \
    [-if_formal_actual '{'formal_name actual_name'}'] \
    -connect_formal_actual '{'formal_name actual_name'}'
} | \
{ -module module_name -group cell_list \
    [-if_formal_actual '{'formal_name actual_name'}'] \
    -connect_formal_actual '{'formal_name actual_name'}' }
[-force]
[-h]
```

Arguments

- **-module *module_name***

Required argument set that specifies the parent cell inside which connections are to be written. The wildcard character “*” can be specified as the **module_name**, which implies that this command is applicable for all the cells present in the design based upon other conditions specified in the command.

- **-connect_formal_actual '{'formal_name actual_name'}**

Required argument set that specifies the formal port and actual port for which a connection is to be made in the output netlist. The **formal_name** and **actual_name** arguments must be specified in a Tcl list. In the case where the specified formal port or actual port is not present in the design, then the command is ignored.

- **-inst *instance_name***

Argument set that specifies the instance name for which the formal port and actual port connection is to be made in the output netlist. The wildcard character “*” can be specified as the **instance_name**, which causes this command to apply for all the instances present in the cell mentioned with the **-module** argument, and based on further conditions specified in the command. Either this argument set or the **-group** argument set must be specified. “[Example — Adding actual pins](#)” on page 735 and the example shown under [v2lvs::add_formal_port](#) demonstrate this option.

- **-group *cell_list***

Argument set that specifies the cell names for which a net and port connection is to be made in the output netlist. The **cell_list** must be a Tcl list. Either this or the **-inst** argument set must be specified. “[Example — Adding actual pins using a cell list](#)” on page 736 demonstrates this option.

- **-if_formal_actual** ‘{’*formal_name actual_name*‘}’

Optional argument set that specifies a condition on port connections to test. Success of the test causes the command to be applied. If the net specified as a ***formal_name*** port is connected to the ***actual_name*** port, only then the connection for the formal port and actual port specified with **-connect_formal_actual** is written to the output netlist.

Generally, if this option is used, the *actual_name* is the same as the corresponding argument in the **-connect_formal_actual** argument set. The idea is, the ***formal_name*** arguments mentioned with **-connect_formal_actual** and **-if_formal_actual** are connected to the same net: the ***actual_name*** mentioned with both argument sets. However, the command also supports the case where the *actual_name* is not the same as **-connect_formal_actual** and **-if_formal_actual**.

- **-force**

Optional argument that adds a named port to the specified instances or cells, where this port does not exist in the Verilog structural netlist or in the SPICE library. The formal port in the command where **-force** appears is generated by a prior v2lvs::add_formal_port command in the flow. If the specified instances or cells have an instantiated SPICE module definition included in the run that has an actual port of the same name as the one to be added, then a pin connection is made in the output. A top-level port is also added to the SPICE output matching the specified pin name. See the Examples section for further discussion of this option.

- **-h**

Optional argument that shows the command usage message.

Description

This command makes connections between the ***formal_name*** and the ***actual_name*** nets specified with the **-connect_formal_actual** argument based upon the **-module**, **-inst**, **-group** and **-if_formal_actual** arguments.

The definitions of formal and actual names are found under “[Tcl Interface Command Encyclopedia](#)” on page 771. In the SPICE output, when a \$PINS construct is written, the port assignment is of the form ***formal_name=actual_name***.

If this command specifies connections for ports/pins that already have connectivity, the existing connectivity is not changed.

If there is a conflict between v2lvs::add_actual_port and [v2lvs::add_pin](#), the v2lvs::add_actual_port command prevails.

If there are multiple v2lvs::add_actual_port commands that have conflicting connections specified, such as here:

```
v2lvs::add_actual_port -module * -inst * -if_formal_actual {VDD vddx}\n  -connect_formal_actual {VDBB vddx}\n  v2lvs::add_actual_port -module * -inst * -if_formal_actual {VDD vdd_abc}\n  -connect_formal_actual {VDBB vdd_abc}
```

where either command might apply to an instance, the first command is used and the second is ignored.

Command Line Equivalence

None.

Examples

See “[Example — Adding actual pins](#)” on page 735 for a primary use case.

For the following case, the intent is to introduce a port named VDD0 in the SPICE representation of Verilog module top_vi that connects to the VDDSUB port in the SPICE library subcircuit block1.

```
///////////////////////////// Verilog file top2.v ///////////////////\nmodule top_vi (I,Z,VDDI,VSS);\n  input I;\n  output Z;\n  inout VDDI;\n  inout VSS;\n  block1 inst1 (.I(I), .Z(Z), .VDD(VDDI), .VSS(VSS));\nendmodule\n\nmodule top (I,Z,VDDI,VDDO,VSS,TM,TZ);\n  input I;\n  input [3:0] TM;\n  output Z;\n  inout VDDI, VDDO, VSS;\n  output [3:0] TZ;\n  wire net1;\n  block1 inst2 (.I(I), .Z(net1), .VDD(VDDO), .VSS(VSS));\n  top_vi top_vi_inst( .I(net1), .Z(Z), .VDDI(VDDI), .VSS(VSS));\nendmodule\n////////////////// End of Verilog file ///////////////////
```

Here is the SPICE library:

```
***** SPICE library file block1.spi *****\n.SUBCKT block1 I Z VDD VSS VDDSUB\n.ENDS\n***** End of SPICE library *****
```

Here is the command file. The -force option causes the VDD0 port to be added to top_vi.

```
#####
# V2LVS Tcl command file #####
v2lvs::load_spice -range_mode -filename block1.spi -detect_bus_delimiter
v2lvs::load_verilog -filename top2.v
v2lvs::add_formal_port -port VDDSUB -under block1
### connect formal port VDDSUB to actual port VDD0 in block1
v2lvs::add_actual_port -module * -group block1 \
-connect_formal_actual {VDDSUB VDD0}
### add formal port VDD0 to top_vi
v2lvs::add_formal_port -port VDD0 -under top_vi
### connect formal and actual VDD0 ports, and add VDD0 port to top_vi
v2lvs::add_actual_port -module * -group top_vi \
-connect_formal_actual {VDD0 VDD0} -force
v2lvs::write_output -filename top.spi
exit
#####
End of file #####
```

Here is the output netlist *top.spi*. The VDD0 port in *top_vi* is due to the v2lvs::add_actual_port -force option. The connections to this net are then made throughout the output hierarchy, as specified by the command script.

```
.SUBCKT top_vi I Z VDDI VSS VDDO
Xinst1 block1 $PINS I=I Z=Z VDD=VDDI VSS=VSS VDDSUB=VDDO
.ENDS

.SUBCKT top I Z VDDI VDDO VSS TM[3] TM[2] TM[1] TM[0] TZ[3] TZ[2] TZ[1]
+ TZ[0]
Xinst2 block1 $PINS I=I Z=net1 VDD=VDDO VSS=VSS VDDSUB=VDDO
Xtop_vi_inst top_vi $PINS I=net1 Z=Z VDDI=VDDI VSS=VSS VDDO=VDDO
.ENDS
```

v2lvs::add_formal_port

Netlist configuration command.

Specifies to add a formal port/pin.

Usage

v2lvs::add_formal_port -port *formal_name* [-under *cell_name*] [-h]

Arguments

- **-port *formal_name***

Required argument set that specifies the name of a formal port.

- **-under *cell_name***

Optional argument set that specifies a cell above which the *formal_name* is not propagated as a formal port.

- **-h**

Optional argument that shows the command usage message.

Description

This command finds nets at lower levels of hierarchy and propagates them upward as ports in the design using the specified *formal_name*.

The -under option specifies *cell_name*, which acts as a hierarchical ceiling beyond which the net is not propagated.

The definitions of formal and actual names are found in the section “[Tcl Interface Command Encyclopedia](#)” on page 771. In the SPICE output, when a \$PINS construct is written, the port assignment is of the form *formal_name=actual_name*.

Command Line Equivalence

None.

Examples

In the following example, formal ports vdd1 and VBBD are added to the modules test and domain1 respectively.

Port vdd1 is added to module test (based on Rule 1) because the module domain1, which is down the hierarchy, has a net vdd1. Similarly, port VDDB is added to the module domain1 because inv is down the hierarchy and it contains a VDDB port. However, VDDB is not added to module test because test is specified using the -under option in the RULE 2 of the rule file.

```
////////// Verilog file test.v ///////////////
module test (in, out1, out2);
    input in;
    output out1, out2;
    domain1 d1 (.a(in), .z(out1));
endmodule

module domain1 (a,z);
    input a;
    output z;
    supply1 vdd1;
    supply0 vss;
    inv u1 (.A(a), .Z(net1), .VDD(vdd1), .VSS(vss));
    inv u2 (.A(net1), .Z(z), .VDD(vdd1), .VSS(vss));
endmodule
////////// End of Verilog file ///////////
```

Here is the SPICE library:

```
***** SPICE library file lib.spi *****
.SUBCKT inv VDD VSS A Z VDDB
MPA Z A VDD VDDB P W=1e+06 L=1e+06
MNA Z A VSS VSS N W=1e+06 L=1e+06
.ENDS
***** End of SPICE library *****
```

Here is the command file:

```
##### V2LVS Tcl command file #####
v2lvs::load_verilog -filename test.v
v2lvs::load_spice -filename lib.spi -pin_mode
# This propagates the bulk signals and different power signals
# through the hierarchy.
# RULE 1
v2lvs::add_formal_port -port vdd1

# RULE 2
v2lvs::add_formal_port -port VDDB -under test
# Connect the bulk signals to the correct supplies based on module
# hierarchy.
v2lvs::add_actual_port -module test -inst d1 \
-connect_formal_actual {VDDB vdd1}
v2lvs::write_output -filename test.spi
exit
##### End of file #####
```

Here is the output:

```
.SUBCKT test in out1 out2 vdd1
Xd1 domain1 $PINS a=in z=out1 vdd1=vdd1 VDDB=vdd1
.ENDS

.SUBCKT domain1 a z vdd1 VDDB
Xu1 inv $PINS A=a Z=net1 VDD=vdd1 VSS=vss VDDB=VDDB
Xu2 inv $PINS A=net1 Z=z VDD=vdd1 VSS=vss VDDB=VDDB
.ENDS
.GLOBAL vdd1
.GLOBAL vss
```

For instance Xd1, formal port vdd1 is added and is connected to the actual port of the same name. Formal port VDDB is created and connected to actual port vdd1.

For the instances Xu1 and Xu2 in the subcircuit domain1 VDDB=VDDB assignment is written since the formal port VDDB is added to the module domain1 through the v2lvs::add_formal_port command. The formal port VDDB is connected to the actual port VDDB in the inv subcircuit.

See also “[Example — Adding actual and formal pins](#)” on page 737.

v2lvs::add_pin

Netlist configuration command.

Adds pins to subcircuit calls and definitions where such pins do not exist in the Verilog or SPICE inputs.

Usage

```
v2lvs::add_pin -pin pin1 [-pin pin2 ...] [-h]
```

Arguments

- **-pin *pin1***

Required argument set that specifies to add a pin to the output subcircuit definitions. More than one set may be specified.

- **-h**

Optional argument that shows the command usage message.

Description

Adds one or more pins specified with the **-pin** argument set to subcircuit definitions and subcircuit calls in the output SPICE netlist where such pins did not exist in the input Verilog or in relevant SPICE libraries.

This command should be used with care because it adds pins that are not explicitly present in the original Verilog netlist. Generally, SPICE .GLOBAL declarations are better suited for this purpose. Additionally, the rule file specification statement [LVS Spice Override Globals](#) can provide flexibility for designs with multiple supply nets.

The following things occur when using this command:

Each specified pin is added to .SUBCKT statements in the output SPICE netlist if the original Verilog module definition did not already have a pin with that name.

The assignment *pinN=pinN* is added to subcircuit calls in the output SPICE netlist if the original Verilog instance did not already have a connection to *pinN*.

Exception: If the subcircuit being called is defined in a SPICE netlist read with the [v2lvs::load_spice](#) command, and that subcircuit does not have a pin called *pinN*, then *pinN* is not added to the subcircuit call.

This command is incompatible with the [v2lvs::generate_ordered_pins](#) command and the **-i** option.

See “[Addition of Pins](#)” on page 733 for an example.

Command Line Equivalence

-addpin *pin_name*

v2lvs::combine_interface_info

Netlist configuration command.

Specifies whether to prefer Verilog library pin order over the SPICE library when both refer to the same macro.

Usage

```
v2lvs::combine_interface_info { -disable | -enable} [-h]
```

Arguments

- **-disable**

Causes the pin order to be taken from a SPICE library instead of a Verilog library when both refer to the same macro. This is the default.

- **-enable**

Causes the pin order to be taken from a Verilog library instead of a SPICE library when both refer to the same macro.

- **-h**

Optional argument that shows the command usage message.

Description

This command is used in conjunction with [v2lvs::load_spice](#) for determining output pin order. In cases where there is a Verilog library module that is read by [v2lvs::load_verilog](#) -lib_mode that matches the subcircuit in the SPICE library (that is, they are duplicates), the Verilog module's pin order is preferred for output by specifying **-enable**. By default, pin order for output subcircuits is taken from the SPICE library referenced by [v2lvs::load_spice](#).

Command Line Equivalence

None.

Examples

For this example, assume the following files.

Verilog primary netlist *src.v*:

```
module top;
MACRO_A I_MACRO_A (
    .A({A_3_,
        A_2_,
        A_1_,
        A_0_}),
    .X(n_X));
endmodule
```

Verilog library netlist *lib.v*:

```
module MACRO_A (A, X);
  input [0:3] A;
  output X;
endmodule
```

SPICE netlist *lib.spi*:

```
* port range has differing order from lib.v except for X
* this has a different bus delimiter than lib.v and src.v
*.BUSDELIMITER <
.SUBCKT MACRO_A A<3> A<1> A<2> A<0> X
.ENDS
```

Assume this V2LVS script:

```
v2lvs::load_verilog -filename src.v
v2lvs::load_verilog -filename lib.v -lib_mode
v2lvs::load_spice -filename lib.spi -range_mode
v2lvs::set_includes -filename lib.spi
v2lvs::write_output -filename out.sp
exit
```

Upon execution of the script, these warnings are issued because *lib.spi* has a non-sequential port range:

```
Warning: Encountered non-sequential pin: A<1> at lib.spi:4
Port range values may not be consistent.
Warning: Encountered non-sequential pin: A<2> at lib.spi:4
Port range values may not be consistent.
```

This warning is also given because of a duplicate macro definition:

```
Warning: Multiple declarations of module MACRO_A in lib.v:2
```

This is the output netlist, which represents the default behavior:

```
.INCLUDE "lib.spi"
.SUBCKT top
XI_MACRO_A MACRO_A $PINS A<3>=A_3_ A<2>=A_2_ A<1>=A_1_ A<0>=A_0_ X=n_X
.ENDS
```

Notice that the pin order is descending starting with A<3> and shows the bus delimiter from *lib.spi*.

Now assume this command is used in the same script:

```
v2lvs::combine_interface_info -enable
```

The warning about duplicate macro definitions is not given because the interface information is combined.

This is the output:

```
.INCLUDE "lib.spi"
.SUBCKT top
XI_MACRO_A MACRO_A $PINS A<0>=A_3_ A<1>=A_2_ A<2>=A_1_ A<3>=A_0_ X=n_X
.ENDS
```

Notice the pin order is now ascending, starting with A<0>. The pin order is taken from *lib.v* in this case.

v2lvs::connect_missing_ports

Input command.

Specifies whether to write the connection for missing ports.

Usage

`v2lvs::connect_missing_ports {-enable | -disable | -status} [-h]`

Arguments

- **-enable**

Argument that specifies to write connections for missing ports. This is the default.

- **-disable**

Argument that specifies not to write connections for missing ports.

- **-status**

Argument that specifies to indicate the setting of this command. This option returns a 1 if **-enable** is currently selected and a 0 if **-disable** is currently selected.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies whether to write connections for missing ports. At least one of the three required arguments must be specified.

Command Line Equivalence

None.

Examples

Given the following Verilog file:

```
module top (net1, net2, net3, VDD, VSS);  
    CELL1 inst1( .in1(net1), .in2(net2), .out3(net3), .VDD(VDD), .VSS(VSS));  
endmodule
```

the following SPICE library:

```
.global VDD  
.global VSS  
.SUBCKT CELL1 in1 in2 out3  
.ENDS
```

and this Tcl script:

```
v2lvs::load_verilog -filename design.v
v2lvs::load_spice -filename lib.s -pin_mode
v2lvs::connect_missing_ports -enable
v2lvs::write_output -filename design.out
exit
```

The following warnings are generated when the script is run:

```
Warning: top/inst1 calls port VDD not declared in module CELL1
Warning: top/inst1 calls port VSS not declared in module CELL1
```

This is created when v2lvs::connect_missing_ports is enabled:

```
.SUBCKT top net1 net2 net3 VDD VSS
Xinst1 CELL1 $PINS in1=net1 in2=net2 out3=net3 VDD=VDD VSS=VSS
.ENDS
```

When v2lvs::connect_missing_ports is disabled, v2lvs does not write the connection for the missing ports:

```
.SUBCKT top net1 net2 net3 VDD VSS
Xinst1 CELL1 $PINS in1=net1 in2=net2 out3=net3
.ENDS
```

v2lvs::create_pex_template

Output command.

Writes a Calibre xRC source template file.

Usage

v2lvs::create_pex_template -dirname *svdb_dir* [-h]

Arguments

- **-dirname *svdb_dir***

Required argument set that specifies a [Mask SVDB Directory](#) path to which a Calibre xRC source template file is written.

- **-h**

Optional argument that shows the command usage message.

Description

Causes a Calibre xRC source template file to be written to the specified [Mask SVDB Directory](#).

See “[Calibre xRC Source Template File](#)” on page 762.

Command Line Equivalence

-t *svdb_dir*

v2lvs::do_source

Input command.

Specifies to read and execute a Tcl script.

Usage

v2lvs::do_source -filename *tcl_script* [-h]

Arguments

- **-filename *tcl_script***

Required argument set that specifies a Tcl script to execute during the run.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies to read and execute the *tcl_script*. This command is recommended instead of the Tcl source command because v2lvs::do_source provides the line numbers where the *tcl_script* fails to parse correctly.

Command Line Equivalence

None.

Examples

See “[Example — Adding actual pins](#)” on page 735 for a use case.

v2lvs::find_module

List command.

Creates a list of module names from the input Verilog netlist. When used without any optional argument sets, this command returns a list of all module names.

Usage

v2lvs::find_module [-under *cell_name*] [-match *expression*] [-undef_mod] [-def_mod] [-h]

Arguments

- **-under *cell_name***
Optional argument set specifying that only module names in the underlying hierarchy of the *cell_name* are selected.
- **-match *expression***
Optional argument set specifying that only module names matching the specified *expression* are chosen. The *expression* may contain valid Verilog characters for module names in addition to the “*” wildcard. The * matches 0 or more characters.
- **-undef_mod**
Optional argument that when specified exclusively shows a list of undefined modules that have been used in the design.
- **-def_mod**
Optional argument that when specified exclusively shows a list of defined modules that have been used in the design.
- **-h**
Optional argument that shows the command usage message.

Command Line Equivalence

None.

Examples

```
# match all modules below cellA
set below_A [ v2lvs::find_module -under cellA ]
```

The v2lvs::find_module command initiates the parsing of the input Verilog files if it has not already occurred. Parsing occurs only once.

v2lvs::generate_empty_subckts

Netlist configuration command.

Controls empty .SUBCKT output.

Usage

```
v2lvs::generate_empty_subckts {-disable | -enable | -status} [-h]
```

Arguments

- **-disable**

Argument that specifies to write only non-empty subcircuits. This is the default.

- **-enable**

Argument that specifies to write empty subcircuits for all modules.

- **-status**

Argument that specifies to indicate the setting of this command. This option returns a 1 if **-enable** is currently selected and a 0 if **-disable** is currently selected.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies whether to write empty subcircuits for all modules. At least one of the three required arguments must be specified. The **-enable** option can be useful for generating “black box” circuits from library files.

Command Line Equivalence

```
-e
```

Examples

See “[Creating LVS Box Subcircuits](#)” on page 765.

v2lvs::generate_ordered_pins

Netlist configuration command.

Specifies whether pin netlisting uses traditional SPICE rather than \$PINS constructs.

Usage

```
v2lvs::generate_ordered_pins {-disable | -enable | -status} [-h]
```

Arguments

- **-disable**

Argument that specifies to write \$PINS constructs in subcircuit calls. This is the default.

- **-enable**

Argument that specifies to write ordered pins in subcircuit calls.

- **-status**

Argument that specifies to indicate the setting of this command. This option returns a 1 if **-enable** is currently selected and a 0 if **-disable** is currently selected.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies whether to write subcircuit calls having \$PINS constructs or ordered pins as in traditional SPICE. At least one of the three required arguments must be specified.

The **-enable** option causes all pins to be listed without \$PINS constructs, which is required by certain downstream tools.

The **-status** option is for checking the status of this command in a script.

See “[Simulation Output Generation](#)” on page 760 for a use case.

Command Line Equivalence

`-i`

Examples

Here is the Verilog structural netlist. Notice the port range order is numerically descending.

```

module top;
wire [3:0] w1;
wire w2;
BLK_A instA ( .I( w1 ), .Y( w2 ) );
endmodule

module BLK_A ( I, Y );
inout [3:0] I;
inout Y;
wire [3:0] A;
wire B;
BLK_B instB ( .A( I ), .B( Y ) );
endmodule

```

Here is the SPICE library. Notice the port range order is the reverse of the Verilog.

```

* reversed port order compared to structural netlist
.SUBCKT BLK_B A[0] A[1] A[2] A[3] B
.ENDS

```

This script generates the default \$PINS output with this script:

```

v2lvs::load_verilog -filename top.v
v2lvs::load_spice -filename spice_lib -range_mode
v2lvs::set_includes -filename spice_lib
v2lvs::generate_ordered_pins -disable
v2lvs::write_output -filename out.sp
exit

```

Output:

```

.INCLUDE "spice_lib"

.SUBCKT top
XinstA BLK_A $PINS I[3]=w1[3] I[2]=w1[2] I[1]=w1[1] I[0]=w1[0] Y=w2
.ENDS

.SUBCKT BLK_A I[3] I[2] I[1] I[0] Y
XinstB BLK_B $PINS A[0]=I[3] A[1]=I[2] A[2]=I[1] A[3]=I[0] B=Y
.ENDS

```

In particular, notice how the A and I port ranges have been mapped. The A range order comes from the SPICE library. The I range order comes from the Verilog.

If this command is used:

```
v2lvs::generate_ordered_pins -enable
```

then the output is as follows:

```
.INCLUDE "spice_lib"

.SUBCKT top
XinstA w1[3] w1[2] w1[1] w1[0] w2 BLK_A
.ENDS

.SUBCKT BLK_A I[3] I[2] I[1] I[0] Y
XinstB I[3] I[2] I[1] I[0] Y BLK_B
.ENDS
```

The I range order correctly corresponds to the SPICE library port range order.

v2lvs::get_includes

Iterator creation command.

Returns a list of Verilog `include filenames.

Usage

`v2lvs::get_includes [-h]`

Arguments

- `-h`

Optional argument that shows the command usage message.

Description

Returns a Tcl list of all `include files. For example:

```
set include_files [v2lvs::get_includes]
```

See also [v2lvs::include_dir](#).

Command Line Equivalence

None.

v2lvs::get_ports

List command.

Returns a list of port names for a module.

Usage

v2lvs::get_ports -module *module_name* [-h]

Arguments

- **-module *module_name***

Required argument set that specifies the module from which port names are to be written.

- **-h**

Optional argument that shows the command usage message.

Description

Returns Tcl list of all port names in a module.

Command Line Equivalence

None.

Examples

In this example, port_list creates a list of ports in the mux module:

```
set port_list [v2lvs::get_ports -module mux]
```

The port_list could be used in conjunction with commands like [v2lvs::add_actual_port](#), [v2lvs::add_formal_port](#), and [v2lvs::connect_missing_ports](#).

v2lvs::help

Usage command.

Shows the command usage description.

Usage

v2lvs::help

Arguments

None.

Command Line Equivalence

-h

v2lvs::include_dir

Input command.

Specifies a directory for `include files.

Usage

v2lvs::include_dir -dirname *directory_name* [-h]

Arguments

- **-dirname *directory_name***

Required argument set that specifies where `include directive files are located.

- **-h**

Optional argument that shows the command usage message.

Command Line Equivalence

-incvdir *dir*

v2lvs::insert_port_instance

Output command.

Inserts a new instance inside of a given module and assigns an added pin to the instance.

Usage

```
v2lvs::insert_port_instance -module module_name -inst_type instance_name
    -port port_name [-h]
```

Arguments

- **-module *module_name***
Required argument set that specifies the module in which an instance is to be written.
- **-inst_type *instance_name***
Required argument set that specifies a name for the added instance.
- **-port *port_name***
Required argument set that specifies an instance pin name or number.
- **-h**
Optional argument that shows the command usage message.

Command Line Equivalence

None.

Examples

Given the following Verilog file:

```
module mux(f,a,sel);
  input [3:0] a;
  input sel;
  output f;
  wire f1, f2;
  not(nsel,sel);
  and(f1,a[1],nsel);
  and(f2,a[0],sel);
  or (f,f1,f2);
endmodule
```

using the following command:

```
v2lvs::insert_port_instance -module mux -inst_type new_1 -port 1
```

creates the following output SPICE:

```
.SUBCKT mux f a[3] a[2] a[1] a[0] sel
x0 1 1000 new_1

X1 f1 a[1] nsel and
X2 f2 a[0] sel and
X3 f f1 f2 or
X4 nsel sel not
.ENDS
```

v2lvs::load_spice

Input command.

Specifies a SPICE library file with translation mode for SPICE pins.

Usage

```
v2lvs::load_spice -filename spice_library
    {-pin_mode | {-range_mode [-detect_bus_delimiter]} } [-libname library_name] [-h]
```

Arguments

- **-filename *spice_library***

Required argument set that specifies the path to a SPICE library file.

- **-pin_mode**

Argument set that specifies to translate SPICE pins with no assembly into arrays.

- **-range_mode**

Argument set that specifies to translate SPICE pins with array delimiter characters into arrays or ranged ports.

- **-detect_bus_delimiter**

An optional argument specified with **-range_mode** that automatically detects the bus delimiter character in the *spice_library*. A *.BUSDELIMITER statement with the appropriate character is added before the .INCLUDE directive in the output netlist when **v2lvs::set_includes** is used.

- **-libname *library_name***

The name of an output SPICE library file. This option must be specified in order for **v2lvs::rename_duplicate_cells** to have an effect. The *library_name* file contains all SPICE subcircuits in the *spice_library*, including any renamed duplicates.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies a SPICE library file and a method for generating pins. Either the **-pin_mode** or **-range_mode** option must be specified. Input files to this option that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment. This command may be specified more than once to process multiple SPICE library files.

When the **-pin_mode** option is specified, the *spice_library* is parsed for interface configurations. This option translates SPICE pins directly from ports with no assembly into arrays. Pins with pin select ([]) by default) annotation are kept as individual pins using escaped identifiers. See “[Pin Mode](#)” on page 753 for details.

When the **-range_mode** option is specified, the *spice_library* is parsed for interface configurations. This option translates SPICE pins with array delimiter characters into arrays or ranged ports. Pins with pin select ([] by default) annotation are assembled into Verilog ranges. See “[Range Mode](#)” on page 752 for details.

The **-detect_bus_delimiter** option is useful when SPICE libraries contain non-default bus delimiter characters, or when their bus delimiters differ from the Verilog circuit. When this option is used, *.BUSDELIMITER statements are written for all v2lvs::set_includes SPICE netlists that are included in the output. Additionally, there is a *.BUSDELIMITER statement that governs the top-level SPICE subcircuit written to the output.

Contents of SPICE .INCLUDE and .LIB statements are not output by this command. To output such content, specify [v2lvs::write_output](#) with the **-outslib** option.

The following output occurs when a non-existent *spice_library* is specified:

```
File "spice_library" does not exist
```

See also [v2lvs::load_verilog](#) and [v2lvs::set_array_delimiter](#).

Command Line Equivalence

-lsp *spice_library_file*: **-pin_mode**

-lsr *spice_library_file*: **-range_mode**

Examples

The following files are assumed for the examples.

Top-level Verilog netlist *src.v*:

```
module top;
MACRO_A I_MACRO_A (
    .A({A_3_,
        A_2_,
        A_1_,
        A_0_}),
    .X(n_X));
endmodule
```

SPICE library *lib.spi*:

```
* port range is out of order.
* different bus delimiter than src.v.
.SUBCKT MACRO_A A<3> A<1> A<2> A<0> X
.ENDS
```

Example 1

This script:

```
v2lvs::load_verilog -filename src.v
v2lvs::load_spice -filename lib.spi -range_mode
v2lvs::set_includes -filename lib.spi
v2lvs::write_output -filename ex1.sp
exit
```

generates these warnings due to a non-sequential port range in the SPICE library:

```
Warning: Encountered non-sequential pin: A<1> at lib.spi:4
Port range values may not be consistent.
Warning: Encountered non-sequential pin: A<2> at lib.spi:4
Port range values may not be consistent.
```

This is the output netlist:

```
.INCLUDE "lib.spi"
.SUBCKT top
XI_MACRO_A MACRO_A $PINS A<3>=A_3_ A<2>=A_2_ A<1>=A_1_ A<0>=A_0_ X=n_X
.ENDS
```

Notice that the pin range is written in descending sequential order. The \$PINS construct's mapped pin names are taken from *lib.spi* using the bus delimiter found there and the pin names from *src.v*.

Example 2

If the -detect_bus_delimiter option is added to the Example 1 script, this is the output:

```
*.BUSDELIMITER <
.INCLUDE "lib.spi"
*.BUSDELIMITER [
.SUBCKT top
XI_MACRO_A MACRO_A $PINS A<3>=A_3_ A<2>=A_2_ A<1>=A_1_ A<0>=A_0_ X=n_X
.ENDS
```

The first *.BUSDELIMITER statement governs *lib.spi*. The second *.BUSDELIMITER statement governs the top subcircuit, which has no ports in this case.

v2lvs::load_verilog

Input command.

Specifies a Verilog structural netlist or library file.

Usage

v2lvs::load_verilog -filename *verilog_file* [-lib_mode] [-h]

Arguments

- **-filename *verilog_file***

Required argument set that specifies a Verilog structural netlist.

- **-lib_mode**

Optional argument that specifies the *verilog_file* is a library file.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies the input Verilog structural netlist. When -lib_mode is not used, the *verilog_file* is the primitive Verilog design file. Input files that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment.

When -lib_mode is used, this command defines the Verilog library filename only. The Verilog library is parsed for interface pin configurations. The library is not translated to SPICE.

When calls are made to SPICE modules, a description of the SPICE interface can be provided in Verilog syntax using the *verilog_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.

If an attempt to open a non-existent file occurs, the following message is given:

```
File "verilog_file" does not exist
```

See [v2lvs::combine_interface_info](#) for information about preferring Verilog library macro pin order in the output netlist.

Command Line Equivalence

-v *verilog_design_file*: -lib_mode is not specified.

-l *verilog_lib_file*: -lib_mode

Examples

See “[Basic V2LVS Usage](#)” on page 712 for examples.

v2lvs::number_unconnected_pins

Netlist configuration command.

Specifies whether unconnected pins receive numbered connections.

Usage

`v2lvs::number_unconnected_pins {-disable | -enable | -status} [-h]`

Arguments

- **-disable**

Argument that specifies unconnected pins are not netlisted. This is the default.

- **-enable**

Argument that specifies unconnected pins are netlisted.

- **-status**

Argument that specifies to indicate the setting of this command. This option returns a 1 if **-enable** is currently selected and a 0 if **-disable** is currently selected.

- **-h**

Optional argument that shows the command usage message.

Description

Controls handling of unconnected pins. One of the three required arguments must be specified. The **-enable** option causes unconnected pins to receive numbers starting with 1000; therefore, explicit connections are made to unconnected pins. Calibre nmLVS interprets missing pins as unconnected pins.

See also “[Unconnected Pins](#)” on page 722.

Command Line Equivalence

`-n`

v2lvs::override_assign

Netlist configuration command.

Overrides an assign statement in a Verilog module with a specified SPICE subcircuit.

Usage

```
v2lvs::override_assign -spice_subckt circuit_name -module cell_list [-h]
```

Arguments

- **-spice_subckt *circuit_name***

Required argument set that specifies a SPICE subcircuit name.

- **-module *cell_list***

Required argument set that specifies a Tcl list of cell names.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies that the *circuit_name* replace “assign” statements in the Verilog modules specified in the *cell_list*. The *circuit_name* is a subcircuit name in a SPICE library file. The circuit may have only two ports, otherwise a warning message is given, and the assign operation is not overridden. The *cell_list* is a Tcl list comprised of Verilog module names.

This command is useful for through-silicon-via (TSV) interposer designs.

If the same **-module** cell name is specified in two v2lvs::override_assign commands, the second specification is used and a warning is given. For example, suppose these two commands are used:

```
v2lvs::override_assign -spice_subckt dummy1 -module TOP
v2lvs::override_assign -spice_subckt dummy -module TOP
```

This warning is given:

```
WARNING: For the cell TOP override assign cell already set to dummy1.
Resetting it to dummy.
```

Command Line Equivalence

None.

Examples

Assume you have this Verilog file:

```
// Verilog file si_interposer.v
module si_interposer(A, B, C);
    inout A;
    inout C;
    inout B;
    assign A=B;
    assign C=B;
endmodule
```

Assume this is in a SPICE library file:

```
$$ SPICE library dummy.spi
.subckt dummy 1 2
    R0 1 2
.ends
```

Assume these commands are in your V2LVS Tcl command file:

```
# v2lvs.cmd file
v2lvs::load_verilog -filename si_interposer.v
v2lvs::load_spice -filename dummy.spi -pin_mode
v2lvs::override_assign -spice_subckt dummy -module si_interposer
v2lvs::write_output -filename out.spi
exit
```

Then given this command:

```
v2lvs -tcl v2lvs.cmd
```

This is the output:

```
.SUBCKT si_interposer A B C
X0 A 1000 dummy
X1 B 1000 dummy
X2 C 1000 dummy
.ENDS
```

v2lvs::override_globals

Netlist configuration command.

Specifies handling of global nets, including supply nets.

Usage

```
v2lvs::override_globals [-supply0 ground] [-supply1 power] [-supply_not_connected]  
[-use_local_supply] [-default_not_connected] [-custom_override filename] [-h]
```

Arguments

- **-supply0 *ground***
Optional argument set that specifies a .GLOBAL ground name.
- **-supply1 *power***
Optional argument set that specifies a .GLOBAL power name.
- **-supply_not_connected**
Optional argument that specifies that supply0 and supply1 nets are not connected to global ground and power, respectively. This option generates .GLOBAL statements for local power and ground.
- **-use_local_supply**
Optional argument that specifies to use local supply0 and supply1 nets that are not connected to global ground or power.
- **-default_not_connected**
Optional argument that specifies that default power and ground nets are not .GLOBAL nets. This option can be used together with the -use_local_supply option to suppress all .GLOBAL signal generation by V2LVS.
- **-custom_override *filename***
Optional argument set that specifies a filename that defines instance- or module-specific power or ground overrides. See “[Power and Ground Signal Overrides](#)” on page 731.
- **-h**
Optional argument that shows the command usage message.

Description

Controls configuration of .GLOBAL signals in the output netlist. At least one of the optional arguments must be specified.

This command is not affected by the [v2lvs::add_actual_port](#) and [v2lvs::add_formal_port](#) commands. The v2lvs::override_globals options apply only when net connections to ports are specified in the instance statements. The v2lvs::add_actual_port command is used for unconnected ports.

Command Line Equivalence

-s0 *groundnet*: -supply0
-s1 *powernet*: -supply1
-sk: -supply_not_connected
-sl: -use_local_supply
-sn: -default_not_connected
-so: -custom_override

Examples

Use of the options is discussed in detail under “[Power and Ground Connections](#)” on page 727.

v2lvs::preserve_back_slash

Netlist configuration command.

Specifies how backslash (\) characters are handled in escaped identifiers.

Usage

```
v2lvs::preserve_back_slash {-disable | -enable | -status} [-h]
```

Arguments

- **-disable**

Argument that specifies leading backslash characters (\) are not preserved. This is the default.

- **-enable**

Argument that specifies leading backslashes are preserved in escaped identifiers.

- **-status**

Argument that specifies to indicate the setting of this command. This option returns a 1 if **-status** is set and a 0 if **-disable** is set.

- **-h**

Optional argument that shows the command usage message.

Description

Controls output of the leading backslash (\) character in escaped identifiers. One of the three required arguments must be specified.

Verilog allows special identifiers called escaped identifiers to contain any non-white-space character. These escaped identifiers normally map to the same character string in SPICE (when characters used are legal in SPICE) without the leading backslash.

Command Line Equivalence

-b

Examples

See “[Example — Handling escaped identifiers](#)” on page 746 for details.

v2lvs::rename_duplicate_cells

Netlist configuration command.

Resolves duplicate subcircuit names in SPICE libraries.

Usage

```
v2lvs::rename_duplicate_cells {-disable} | {-enable [-case {0 | 1}]} | -status}
```

Arguments

- **-disable**

Argument that specifies duplicate subcircuit names are not resolved. Instead, duplicate-named subcircuits are simply passed through to the output. This is the default behavior.

- **-enable**

Argument that specifies duplicate subcircuit names are resolved. When this option is specified, the [v2lvs::load_spice](#) -libname option must also be used in order to obtain a netlist with resolved duplicate names.

- **-case {0 | 1}**

Optional argument used with **-enable** that controls the case-sensitivity of duplicate name matching. A 0 means text case is not used when matching names, and this is the default. A 1 means text case must match between names in order to treat them as duplicates.

- **-status**

A required argument that returns whether the **-enable** option is currently active. The command returns a 1 if duplicate name resolution is enabled and 0 otherwise.

Description

SPICE libraries referenced by the v2lvs::load_spice command may have naming conflicts in subcircuits originating from multiple design sources. The v2lvs::rename_duplicate_cells command controls how duplicate subcircuit names in SPICE library files are handled.

In Calibre verification runs, the [LVS Spice Allow Duplicate Subcircuit Names](#) rule file statement manages how duplicate subcircuits are handled. Regardless of the setting of that statement, subcircuit definitions that have the same names but different pin/port counts are not considered duplicates by the Calibre SPICE parser. However, duplicate subcircuit definition detection in V2LVS is performed based upon names only, not port counts.

By default, or when **-disable** is specified, V2LVS does not resolve duplicate subcircuits in SPICE libraries. “X” calls to any subcircuits having the same name reference the common name.

When **-enable** is specified, then the following occurs:

- Verilog is always translated directly according to the first encountered subcircuit definition corresponding to an instance call (similar to what occurs in the SPICE parser).

- If duplicate subcircuit definition names are found in SPICE library files, then the first duplicate subcircuit name receives the suffix “_dup_1”, the second “_dup_2”, and so forth.
- If `v2lvs::add_actual_port` is used, pin/port count correspondence is taken into account for (possibly renamed) output SPICE subcircuit calls after any additional ports have been added.
- If a subcircuit call originating in a SPICE library file is mapped to an original subcircuit definition name and port count, then the call references that definition.
- If a subcircuit call originating in a SPICE library file can be mapped to multiple renamed subcircuit definitions by port count, then the call references the name of the last renamed definition.
- Renamed duplicate subcircuits are written to the `v2lvs::load_spice -libname` output regardless of whether they are called.

The netlist with resolved duplicate names is generated only when `v2lvs::load_spice -libname` is used.

Examples

Example 1

Consider the following SPICE library netlists:

```
***** SPICE library file a.sp *****
.SUBCKT BLOCK1 VDD VSS
M0 7 A 3 VSS N
M1 VSS B 7 VSS N
M2 Y 3 VSS VSS N
M3 3 A VDD VDD P
M4 VDD B 3 VDD P
M5 Y 3 VDD VDD P
.ENDS
***** End of SPICE file *****

***** SPICE library file b.sp *****
.SUBCKT BLOCK1 VSS VDD A Y
M0 Y A VSS VSS N
M1 Y A VDD VDD P
.ENDS

.SUBCKT BLOCK2 VSS VDD
M0 Y B VSS VSS N
M1 VSS A Y VSS N
M2 6 B VDD VDD P
M3 Y A 6 VDD P
X4 VSS VDD BLOCK1
.ENDS
***** End of SPICE file *****
```

Notice that BLOCK1 is a duplicate subcircuit name. By default, this warning is given by V2LVS when these netlists are processed:

Warning: Duplicate declaration "BLOCK1" at line 2 in "b.sp".

Now consider the following Verilog netlist and Tcl command script:

```
///////////////////////////// Verilog file top.v ///////////////////////////////
module TOP;
wire A, B;
BLOCK1 X0 (.VDD(B), .VSS(A));
BLOCK2 X1 (.VSS(A), .VDD(B));
endmodule
//////////////////////////// End of Verilog file //////////////////////////////

#####
# V2LVS Tcl command file #####
v2lvs::load_spice -filename a.sp -pin_mode -libname new_a.sp
v2lvs::load_spice -filename b.sp -pin_mode -libname new_b.sp
v2lvs::load_verilog -filename top.v
v2lvs::rename_duplicate_cells -enable
v2lvs::set_includes -filename {new_a.sp new_b.sp}
v2lvs::write_output -filename out.sp
exit
#####
# End of file #####
```

The output library netlists from v2lvs::load_spice -libname are as follows. Note the renaming of the duplicate BLOCK1 subcircuit definition.

```
***** SPICE library file a.sp *****
.SUBCKT BLOCK1 VDD VSS
M0 7 A 3 VSS N
M1 VSS B 7 VSS N
M2 Y 3 VSS VSS N
M3 3 A VDD VDD P
M4 VDD B 3 VDD P
M5 Y 3 VDD VDD P
.ENDS
***** End of SPICE file *****

***** SPICE library file b.sp *****
.SUBCKT BLOCK1_dup_1 VSS VDD
M0 Y A VSS VSS N
M1 Y A VDD VDD P
.ENDS

.SUBCKT BLOCK2 VSS VDD
M0 Y B VSS VSS N
M1 VSS A Y VSS N
M2 6 B VDD VDD P
M3 Y A 6 VDD P
X4 VSS VDD BLOCK1
.ENDS
***** End of SPICE file *****
```

Subcircuit call X4 references BLOCK1 instead of BLOCK_dup_1 because the call's pin count matches the BLOCK1 port count. Renamed subcircuit definitions are written whether or not they are called.

Example 2

This example shows the interaction between v2lvs::rename_duplicate_cells and v2lvs::add_actual_port.

Notice in *top.v* that instance u1 has four pins. In *lib.sp*, the first subcircuit definition has four ports, but the second has three. These are duplicates.

```
////////////////// Verilog file top.v ///////////////////
module top (a,b,x,clk,vddx,vdd_abc,vss);
  input a,b,clk,vdd_abc,vddx,vss;
  output x;
  inv u1 (.A(a), .Z(x), .VDD(vddx), .VSS(vss));
endmodule
//////////////// End of Verilog Input file //////////////////

***** SPICE library file lib.sp *****
.SUBCKT inv VDD VSS A Z VDDB
.ENDS.SUBCKT inv VDD VSS A Z
.ENDS
***** End of SPICE file *****
```

In the following Tcl script, v2lvs::add_actual_port adds a VDDB port connected to vddx in inv.

```
##### V2LVS Tcl command file #####
v2lvs::load_spice -filename lib.sp -pin_mode -libname new_lib.sp
v2lvs::load_verilog -filename top.v
v2lvs::add_actual_port -module top -group inv \
-if_formal_actual {VDD vddx} -connect_formal_actual {VDDB vddx}
v2lvs::rename_duplicate_cells -enable
v2lvs::set_includes -filename new_lib.sp
v2lvs::write_output -filename out.sp
exit
##### End of file #####
```

The following are the output netlists. The first is *out.sp*. Notice the VDDB port has been added to Xu1, and this matches subcircuit inv. The second is *new_lib.sp*. The duplicate subcircuit is renamed.

```
$ Spice netlist generated by v2lvs
.INCLUDE "lib.sp"
*.BUSDELIMITER [
.SUBCKT top a b x clk vddx vdd_abc vss
Xu1 inv $PINS A=a Z=x VDD=vddx VSS=vss VDDB=vddx
.ENDS
```

```
***** SPICE library file lib.sp *****
.SUBCKT inv VDD VSS A Z VDDB
.ENDS

.SUBCKT inv_dup_1 VDD VSS A Z
.ENDS
***** End of SPICE file *****
```

v2lvs::set_array_delimiter

Netlist configuration command.

Specifies array delimiters.

Usage

```
v2lvs::set_array_delimiter -delimiter char1 [char2] [-h]
```

Arguments

- **-delimiter *char1***

Required argument set that specifies a character to replace the leading '[' in an array delimiter. The character should be {, <, or (to conform with usual conventions.

- *char2*

Optional argument that specifies a character to replace the closing ']' in an array delimiter. The character should be }, >, or) to conform with usual conventions.

- -h

Optional argument that shows the command usage message.

Description

Specifies the array delimiter characters. By default, brackets ([]) are used. The *char1* argument is required when this command is used and replaces the initial [. The *char2* argument is optional and replaces the closing]. The *char1* and *char2* arguments are not checked for SPICE compatibility, so you must ensure proper characters are selected.

You use this option to change characters used to describe Verilog net selections to the specified characters in the SPICE output. Verilog supports bus type nets and ports with a bracket [] syntax. SPICE only represents simple nets.

As an example, the Verilog net selection bus_a[2] translates into the SPICE net bus_a[2]. The following command:

```
v2lvs::set_array_delimiter -delimiter < >
```

maps bus_a[2] to the SPICE net bus_a<2>.

See “[Expressions](#)” on page 745 for further details on this topic.

Command Line Equivalence

`-a delimiter1 [delimiter2]`

v2lvs::set_config_file

Netlist configuration command.

Specifies a configuration file that allows scoping of lower-level SPICE blocks incorporated into the top-level Verilog design.

Usage

v2lvs::set_config_file -filename *filename*[-h]

Arguments

- **-filename *filename***
Required argument set that specifies a configuration file.
- **-h**
Optional argument that shows the command usage message.

Description

Specifies a configuration file that allows subcircuit scoping of lower level SPICE blocks that are incorporated into a top-level Verilog design.

Command Line Equivalence

`-cfg config_file`

Examples

See “[Enable Subcircuit Scoping](#)” on page 754.

v2lvs::set_includes

Run configuration command.

Specifies a SPICE netlist to be referenced in a .INCLUDE statement in the output.

Usage

v2lvs::set_includes -filename *list_of_SPICE_libraries* [-h]

Arguments

- **-filename *list_of_SPICE_libraries***

Required argument set that specifies a Tcl list of SPICE library names.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies to add .INCLUDE statements that reference the SPICE library files in the *list_of_SPICE_libraries*. The specified SPICE files are not parsed. They are simply referenced by .INCLUDE statements.

See [v2lvs::load_spice](#) for a command that reads SPICE netlists.

Command Line Equivalence

-s *spice_library_file*

Examples

This combination of commands is frequently used:

```
# reads the SPICE library for interface connections
v2lvs::load_spice -filename lib.spi -pin_mode
# generates a .INCLUDE statement in the output for specified netlists
v2lvs::set_includes -filename lib.spi
```

v2lvs::set_prefix

Netlist configuration command.

Specifies that gate-level primitives receive a prefix.

Usage

`v2lvs::set_prefix -prefix string [-h]`

Arguments

- **-prefix *string***

Required argument set that specifies a prefix for gate-level primitive cell names.

- **-h**

Optional argument that shows the command usage message.

Description

Adds a prefix ***string*** to Verilog gate-level primitive cells. This allows construction of SPICE subcircuits that correspond to the Verilog gates used. This option can help to resolve naming conflicts in certain circumstances.

See “[Primitive Instances](#)” on page 742 for details.

Command Line Equivalence

`-p prefix`

v2lvs::set_unnamed_pin_prefix

Netlist configuration command.

Specifies a prefix to add to unnamed pin connections in module instantiations, and to unnamed primitive and gate instantiations.

Usage

```
v2lvs::set_unnamed_pin_prefix -prefix string [-h]
```

Arguments

- **-prefix *string***
Required argument set that specifies a prefix string for unnamed pin connections.
- **-h**
Optional argument that shows the command usage message.

Command Line Equivalence

```
-u unnamed_pin_prefix
```

v2lvs::set_verbose_level

Run configuration command.

Specifies the verbosity of debug messages issued by the tool.

Usage

v2lvs::set_verbose_level -level {0 | 1 | 2 | 3} [-h]

Arguments

- **-level {0 | 1 | 2 | 3}**

Required argument set that specifies a verbosity level. The default is 0.

- **-h**

Optional argument that shows the command usage message.

Description

Sets the level of runtime debug messaging detail. The default is 0, which means no debugging messages are issued. The level of verbosity increases as the **-level** setting increases. See “[v2lvs::set_verbose Debugging Messages](#).”

Command Line Equivalence

None.

v2lvs::set_warning_level

Run configuration command.

Specifies the verbosity of warning messages issued by the tool.

Usage

```
v2lvs::set_warning_level -level {0 | 1 | 2 | 3 | 4} [-h]
```

Arguments

- **-level {0| 1 | 2 | 3| 4}**

Required argument set that specifies a verbosity level for warnings. The default is 2.

0 — Output no warning messages.

1 — Output warning messages for skipped blocks and modules only.

2 — Output level 1 messages and calls to undeclared modules and pin arrays with widths wider than ports.

3 — Output level 2 messages, called port array mismatches, and unsupported compiler directives.

4 — Output level 3 messages, plus all ignored constructs (as in delays and charges).

- **-h**

Optional argument that shows the command usage message.

Command Line Equivalence

`-w warning_level`

v2lvs::show_properties

Netlist configuration command.

Controls output of instance properties. One of the three required arguments must be specified.

Usage

```
v2lvs::show_properties {-disable} | -enable | -status [-h]
```

Arguments

- **-disable**

Argument that specifies instance properties are not output to the netlist. This is the default.

- **-enable**

Argument that specifies instance properties are output to the netlist.

- **-status**

Argument that specifies to indicate the setting of this command. This option returns 1 if **-enable** is currently set and 0 if **-disable** is currently set.

- **-h**

Optional argument that shows the command usage message.

Command Line Equivalence

None.

v2lvs::substitute_chars

Netlist configuration command.

Specifies character substitutions.

Usage

```
v2lvs::substitute_chars -from find_char -to replace_char [-h]
```

Arguments

- **-from *find_char***

Required argument set that specifies a character to replace in the output netlist.

- **-to *replace_char***

Required argument set that specifies a substitution character.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies to substitute the *replace_char* for the *find_char* in the output netlist. Substitution is allowed only for the characters equals (=), forward slash (/), leading dollar sign (\$), and comma (.). By default, equals (=), leading dollar sign (\$), and comma (.) are replaced by a hash symbol (#).

When using this command, ensure that the leading \$ character in the Verilog is preceded by an escaped identifier so that V2LVS can read it correctly.

See the discussion of escaped identifiers under “[Expressions](#)” on page 745 for related details and an example.

Command Line Equivalence

```
-c char1[char2]
```

v2lvs::verilog_version

Run configuration command.

Specifies the Verilog version in use.

Usage

v2lvs::verilog_version -version {2001 | 1995} [-h]

Arguments

- **-version {2001 | 1995}**

Required argument set that specifies the Verilog version to use. The default is 2001.

- **-h**

Optional argument that shows the command usage message.

Description

The **1995** option should be used when there are Verilog 1995 modules that use Verilog 2001 keywords as identifiers.

See “[Supported Verilog Syntax](#)” on page 833 for details.

Command Line Equivalence

-V1995: 1995

-V2001: 2001

v2lvs::warning_as_error

Run configuration command.

Specifies whether to treat warnings as errors.

Usage

```
v2lvs::warning_as_error {-disable | -enable | -status} [-h]
```

Arguments

- **-disable**

Argument that specifies warnings are not fatal errors. This is the default.

- **-enable**

Argument that specifies warnings are fatal errors.

- **-status**

Argument that specifies to indicate the setting of this command. This option returns a 1 if **-enable** is currently set and a 0 if **-disable** is currently set.

- **-h**

Optional argument that shows the command usage message.

Description

Controls handling of warning severity. One of the three required arguments must be specified. Verbosity is controlled using the [v2lvs::set_warning_level](#) option.

Command Line Equivalence

-werror

v2lvs::write_file_info

Output command.

Specifies whether to write informational comments to the output SPICE netlist.

Usage

`v2lvs::write_file_info {-disable | -enable | -status} [-h]`

Arguments

- **-disable**

Argument that specifies not to write runtime information to the output. This is the default.

- **-enable**

Argument that specifies to write runtime information to the output.

- **-status**

Argument that specifies to indicate the setting of this command. This option returns a 1 if **-status** is currently set and a 0 if **-disable** is currently set.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies to write runtime information to the output SPICE netlist. One of the three required arguments must be specified.

The **-enable** option causes the following comments to be written to the SPICE netlist:

`*#SP_INC:(newline) + path` — Pathname of an included SPICE file. Comment spans two lines. Appears when .INCLUDE statements are written to the output.

`*#VLG_MOD: LINE#: line_number PATH: path` — Pathname of Verilog file and line number of the module definition in that file. Appears when Verilog library files are loaded in addition to the top-level netlist.

`*#PIN_TEMP: { SP | VLG | UNDEF_MOD } LINE#: line_number PIN_TEMP_PATH: path` — This comment is written below instance statements in the output SPICE netlist. It provides the pathname of the file from which the pin template for the instance is found along with the line number of the module definition. SP indicates a SPICE library file. VLG indicates a Verilog file. UNDEF_MOD indicates an undefined module.

Command Line Equivalence

None.

v2lvs::write_output

Output command.

Specifies to output a translated SPICE netlist of the input Verilog design. Optionally specifies the name of a consolidated SPICE library file.

Usage

```
v2lvs::write_output [-filename output_netlist] [-outslib library_netlist]  
[-renametext rename_text_file] [-h]
```

Arguments

- **-filename *output_netlist***
Optional argument set that specifies the path to an output SPICE file containing the translated Verilog.
- **-outslib *library_netlist***
An optional argument set that specifies the path to an output SPICE library file. This option consolidates the contents of all of the required SPICE library files and necessary .SUBCKT structures into a single file. Unneeded library files or subcircuits are not included in the output.
- **-renametext *renamed_text_file***
Specifies to output a renamed text file that is included with the Calibre rule set. The rules in the renamed text file avoid text shorts in layout feedthrough cells.
- **-h**
Optional argument that shows the command usage message.

Description

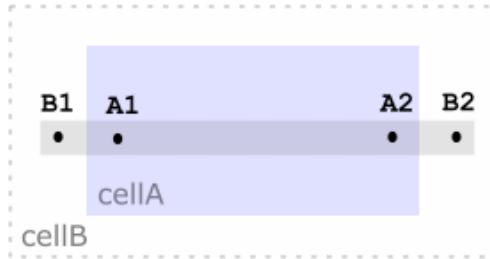
The output netlist from the -filename option uses the Calibre SPICE dialect. When -filename is not specified, the output is written to STDOUT. When -filename is specified, the netlist is written to the specified *output_netlist*. If this command is not used in the V2LVS session, and the -o option also is not used, then the output is written to STDOUT upon exiting the Tcl shell.

The -outslib option is useful if you want to generate a single file that contains only the SPICE library structures needed by the input Verilog design. Any files referenced by .INCLUDE statements in the global scope of SPICE library files are scanned to produce the consolidated output. .INCLUDE statements inside of .SUBCKT blocks are only transcribed in the output; the contents of files referenced by such .INCLUDE statements are not included in the output. .LIB statements are handled in a similar way to the latter .INCLUDE statements.

This command may be specified any number of times in a script. This command triggers a parsing of the Verilog netlists. This is indicated by the “Info: Creating Design Database ...” message.

-renametext Details

Place and route systems can use text markers to specify feedthrough locations in layout cells. The feedthrough markers typically come in pairs, with different text marking each side of the feedthrough. For example:



`cellA` contains a feedthrough that goes from label `A1` to `A2`, and in `cellB` from label `B1` to `B2`. Verilog uses the “assign” command and an instance call to make these connections. The Verilog for this situation is as follows:

```
module cellA( A1, A2 );
  input A1;
  output A2;
  assign A1 = A2;
endmodule

module cellB();
  wire B1, B2;
  cellA cellA_inst( B1, B2 );
endmodule
```

which would be translated by V2LVS to SPICE:

```
.SUBCKT cellA A1 A2
*.CONNECT A1 A2
.ENDS

.SUBCKT cellB
XcellA_inst cellA $PINS A1=B1 A2=B2
.ENDS
```

Nets `A1` and `A2` in `cellA` are shorted by virtue of the `*.CONNECT` directive. One level up in the hierarchy, nets `B1` and `B2` are shorted together through nets `A1` and `A2` by virtue of the `$PINS` comment.

However, in the layout, text labels as shown previously cause a short during circuit extraction. To avoid this, the `-renametext` option is used to output rules that can be included in the Calibre rule set to rename the shorted text objects. So assuming the previous Verilog netlist is loaded and this command is used in V2LVS:

```
v2lvs::write_output -filename out.sp -renametext renamed_text_file
```

then *renamed_text_file* would contain these rules:

```
LAYOUT CELL LIST RenameText_List_cellB cellB
LAYOUT RENAME TEXT
"=^B1$=B2="
CELL LIST RenameText_List_cellB

LAYOUT CELL LIST RenameText_List_cellA cellA
LAYOUT RENAME TEXT
"=^A1$=A2="
CELL LIST RenameText_List_cellA
```

This shows the text label mapping of B1 to B2 and A1 to A2 in cellB and cellA, respectively. Including these rules in the Calibre rule set avoids layout text shorts involving the feedthrough labels.

Command Line Equivalence

- filename: -o *output_spice_file*
- renametext: -rnt *renamed_text_file*

Examples

Example 1

Here is a Verilog netlist *top.v*:

```
module top();
  wire DATA1;
  wire OUT1;
  Cell1 U1 ( .OUT1(OUT1), .IN1(DATA1) );
  Cell2 U2 ( .OUT1(OUT1), .IN1(DATA1) );
endmodule
```

Here are the SPICE library files:

```
$$ Cell1.cdl
*.BUSDELIMITER [
.SUBCKT Cell1 OUT1 IN1
R0 OUT1 IN1 50
.ENDS

.SUBCKT Cell3
R0 OUT1 IN1 50
.ENDS

$$ Cell2.cdl
*.BUSDELIMITER [
.SUBCKT Cell2 OUT1 IN1
R0 OUT1 IN1 50
.ENDS
```

Here is the V2LVS script:

```
v2lvs::load_verilog -filename top.v
v2lvs::load_spice -filename Cell1.cdl -pin_mode
v2lvs::load_spice -filename Cell2.cdl -pin_mode
v2lvs::set_includes -filename v2lvs_lib.cdl
v2lvs::write_output -filename v2lvs_top.cdl -outslib v2lvs_lib.cdl
exit
```

This is the -outslib file *v2lvs_lib.cdl*:

```
*.BUSDELIMITER [
.SUBCKT Cell1 OUT1 IN1
R0 OUT1 IN1 50
.ENDS
.SUBCKT Cell2 OUT1 IN1
R0 OUT1 IN1 50
.ENDS
```

Notice that .SUBCKT Cell3 is omitted because it is not needed by the top-level circuit.

Example 2

See “[Collecting SPICE .INCLUDE Netlists into a Single Netlist](#)” on page 763.

v2lvs::write_pin_info

Netlist configuration command.

Specifies whether to write *.PININFO comments to the output netlist.

Usage

```
v2lvs::write_pin_info {-disable | -enable | -status} [-h]
```

Arguments

- **-disable**

Argument that specifies not to write *.PININFO comments to the output. This is the default.

- **-enable**

Argument that specifies to write *.PININFO comments to the output.

- **-enable**

Argument that specifies to indicate the setting of this command. This option returns a 1 if **-enable** is currently set and a 0 if **-disable** is currently set.

- **-h**

Optional argument that shows the command usage message.

Description

Specifies whether *.PININFO comments are written to the output. These comments indicate the direction of a signal at a pin and are used in certain third-party tools. Long comment lines are continued on new lines using the “*+” continuation sequence.

Array pins are written in ascending alphanumeric order in the *.PININFO line, so the comment line can have a different port order than the .SUBCKT line. For example, if the Verilog netlist has this:

```
module X (a);
  input [2:1] a ;
endmodule
```

then the output is this:

```
.SUBCKT X a[2] a[1]
*.PININFO a[1]:I a[2]:I
.ENDS
```

If the Verilog module has no ports, then no *.PININFO comment is written.

Examples

Here is the Verilog input:

```
module D ( h, g, M);
  input [2:6] h;
  output g;
  inout M;
endmodule
```

This is the output when -enable is used:

```
.SUBCKT D h[2] h[3] h[4] h[5] h[6] g M
*.PININFO h[2]:I h[3]:I h[4]:I h[5]:I h[6]:I g:O M:B
.ENDS
```

Error Handling for the V2LVS Tcl Interface

The V2LVS command line interface performs two levels of parsing before generating an output SPICE file. In the first pass, a database is created and only interfaces for the Verilog modules are parsed. In the second pass, the complete module structure is parsed and converted from Verilog to SPICE.

When using the [V2LVS Tcl Interface](#), the first pass is only done when required. The only commands that require a first pass are [v2lvs::find_module](#) and [exit](#), since both of these commands require database creation. The [exit](#) command only generates output if the [v2lvs::write_output](#) is used before it.

If an error occurs while creating a database or generating output when using the V2LVS Tcl Interface, then the application might need to be terminated and restarted. V2LVS only creates a database once per invocation. Once a database is loaded, all other load commands are ignored.

During a V2LVS run, there can be errors at different stages:

Error before the creation of a database — You can continue working with the current session, and there is no need to restart the application.

Errors during the creation of a database — In these instances, it is advisable to terminate the current session and restart the application.

Errors after the creation of a database — You can continue working with the current session, and there is no need to restart. Once a database is created you cannot update or change the database without restarting the application.

Errors during the generation of output — In these instances, fixing the error and restarting the application is the best course of action.

When a Tcl command is given to V2LVS there can be two types of errors:

Usage Error — If there is a syntax or usage issue in the Tcl command, then V2LVS gives an error explaining the correct usage of that command.

Run Time Processing Error — These errors occur during the creation of a database or when generating output. In these instances, terminating the application and restarting the session is advisable.

When sourcing a Tcl file, an error may occur. If a database is created before an erroneous command, then fixing the error and restarting the application is the best solution. If a syntax or usage error occurs before the creation of the database, then fixing the error and sourcing the file again is acceptable.

Supported Verilog Syntax

V2LVS supports Verilog-2001 keywords and constructs. Such items may be parsed but are not translated. When this is the case, it is indicated in the table as a parsing support issue.

Table 16-7. Verilog-2001 Constructs

Construct	Description of supported elements
ANSI style ports	<p>All styles of port declaration and initialization as specified in the IEEE Std 1364-2001 are supported, including:</p> <ul style="list-style-type: none"> • data-type (integer, etc.) with port declarations • output port declaration including: net-type, reg, integer, time, signed • output reg/integer/time initialization • ANSI ports with UDP declaration <p>Example:</p> <pre>module ansi_ports (input read, read2, input write, input [7:0] data_in, data_in2, output [7:0] data_out, data_out2, output data_out, inout io1, io2); primitive test (output reg out, input a, b, sel, clk); module top(input wire [7:0] i, output reg [7:0] o); module top(input wire [7:0] i, output real o); module top(input .i({a[0:3]}), output o=1); module top(input wire [7:0] in1, output integer o1, o2, output time ot);</pre>

Table 16-7. Verilog-2001 Constructs (cont.)

Construct	Description of supported elements
`begin_keywords	“1364-1995” and “1364-2001”
Compiler directives	`line (parsing only), `ifndef, `elsif
Config declaration	Library parsing support for config ... endconfig
Expressions	<p>Parsing support of the following elements:</p> <ul style="list-style-type: none"> • >>> and <<< signed shift operators V2LVS treats these as >> and << respectively • event trigger (with -> operator) can take a hierarchical identifier. <p>Example:</p> <pre>if (in1[4:1] == 4'b1010) -> top.a.trig1</pre> <ul style="list-style-type: none"> • event expression can take hierarchical identifier • concatenation of constant-expressions, module-path expressions, and nets, respectively • +: and -: in range specification <p>Example:</p> <pre>assign b1[1+:4] = a[1]; sub i1[1:0] (.a(tp1[11-:8]), .b_i({tp2[2-:3], 1'b0}));</pre> <ul style="list-style-type: none"> • variable assignments can be done to hierarchical paths • constant-range expression (as in assignment to a slice) • power operator (***) (fully supported) • signed numbers in tick notation (2'sb01) • initialization during declaration. Example: reg a = 0; • UDP instantiation in module items and generate items • arrayed identifiers and escaped arrayed identifiers • comma-separated sensitivity list • @*, @(*) event control statements
Library declaration	Library parsing support for cell, design, instance, liblist, and use keywords.
Multidimensional arrays	<p>Parsing support of the following constructs:</p> <ul style="list-style-type: none"> • library parsing for multidimensional arrays. Note that multidimensional ports and nets are not translated to SPICE • multidimensional array declaration (as reg [7:0] a[0:1][3:0]) • multidimensional array usage (as wire b=a[0][0][0])

Table 16-7. Verilog-2001 Constructs (cont.)

Construct	Description of supported elements
Net declaration	Parsing support for signed/unsigned property with net declaration. Parsing support for initialization during declaration. Example: <pre>reg a = 0;</pre>
New keywords	Parsing support of these keywords: automatic, cell, config, design, endconfig, -incdir, instance, liblist, library, realtime, signed, unsigned, use.
Parameters	The following parameter types are supported: <ul style="list-style-type: none"> • parameter and local parameter declarations • parsing support of signed property with local parameter • list of parameter declarations in the module declaration line Examples: <pre>parameter integer width = 8; parameter signed [5:0] p3 = 3; module SUB #(parameter p = 2) (out1, in1, in2); module top #(parameter signed num1= 24, parameter real num2= num1) (input wire [7:0] in, output [5:0] out1);</pre> <ul style="list-style-type: none"> • explicit parameter support in module instantiation Example: <pre>vdff #(.size(8),.delay(12)) mod_c(out_c, in_c, clk);</pre>
specparam	Parsing support of the following elements: <ul style="list-style-type: none"> • optional range specification with specparam declaration for constructs related to module name and interface information, like ANSI ports • pulse_control_specparam in specparam assignment
Timing checks	Parsing support of the following system calls: <ul style="list-style-type: none"> • \$removal, \$timeskew, \$fullskew system calls • notify_reg in all the timing_check systems calls

The -V1995 command line option supports Verilog-1995 syntax. For example, if you have the following code:

```
module generate(input endgenerate);
endmodule
```

The generate and endgenerate words are keywords in Verilog-2001, but they are normal identifiers in Verilog-1995. If you want to interpret them as normal identifiers, use the -V1995 command line option.

V2LVS Command Line Syntax

“[Tcl Interface and Command Line Option Correspondence](#)” shows Tcl shell equivalents to command line options.

V2LVS command line.

Note

 As of 2012.1, enhancements to the V2LVS command line interface have stopped. All further enhancements will be made in the [V2LVS Tcl Interface](#). This being the case, the command line interface should be considered deprecated.

Usage

```
v2lvs -v verilog_design_file
      [-a array_delimiters]
      [-addpin pin_name]
      [-b]
      [-c <char1><string>]
      [-cb]
      [-cfg config_file]
      [-e]
      [-h]
      [-i]
      [-ictrace]
      [-incvdir dir]
      [-l verilog_lib_file]
      [-log pathname]
      [-lsp spice_library_file]
      [-lsr spice_library_file]
      [-n]
      [-o output_spice_file]
      [-p prefix]
      [-rnt renamed_text_file]
      [-s spice_library_file]
      [-so groundnet]
      [-s1 powernet]
      [-sk] [-s1] [-sn]
      [-so filename]
      [-t svdb_dir]
      [-tcl [cmd_file]]
      [-u unnamed_pin_prefix]
      [-V1995] [-V2001]
      [-w warning_level]
      [-werror]
      [-- tcl_arg0 [tcl_arg1 ...]]
```

Arguments

- **-v verilog_design_file**

Specifies the filename of the input Verilog structural netlist. This option is required and may be specified multiple times if there is more than one input netlist. Input files to this option

that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment.

- **-a delimiter1[delimiter2]**

Changes the array delimiter characters to the ones you specify. The delimiters must be acceptable in SPICE. The default delimiters are brackets: "[" and "]". 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. For example, the Verilog net selection bus_a[2] translates into the SPICE net bus_a[2]. The -a "<>" option maps bus_a[2] to the SPICE net bus_a<2>. Also see “[*.BUSDELIMITER Statement Handling](#)” on page 756 and the -lsr option.

- **-addpin pin_name**

Note

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 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 supply nets. Adding pins with the [V2LVS Tcl Interface](#) is the preferred method.

This option 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:

- 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.
- The assignment *pin_name=pin_name* is added to subcircuit calls in the output SPICE netlist if the original Verilog instance did not already have a connection to pin *pin_name*.

Exception: If the subcircuit being called is defined in a SPICE netlist read with the -lsp or -lsp options, and that subcircuit does not have a pin called *pin_name*, then *pin_name* is not added to the subcircuit call.

This option is incompatible with the v2lvs -i option. See “[Addition of Pins](#)” on page 733 for an example.

- **-b**

Retains the leading backslash for escaped identifiers. Verilog allows special identifiers called *escaped identifiers* to contain any non-white space character. These escaped identifiers normally map to the same character string in SPICE (when characters used are allowed in SPICE). The leading backslash (\) character that indicates an escaped identifier is normally removed. This option causes it to be retained. See “[Example — Handling escaped identifiers](#)” on page 746 for details.

- `-c 'char1[char2]'`

Specifies the substitution of characters in Verilog (that are disallowed in SPICE) for characters that are permitted in SPICE. The *char1* argument must be specified. The *char2* character must be valid in SPICE when specified. There is no space between the *charN* arguments if two are specified. The entire string of arguments must appear in quotation marks. Single quotes are best to protect a \$ character from the shell.

By default, # is substituted in the SPICE for comma (,), equals (=), and leading dollar (\$) in the Verilog.

If just *char1* is specified, then that character is removed in the output. All the other default substitutions are still made. If both *char1* and *char2* are specified, then *char1* in the Verilog is substituted for *char2* in the SPICE. Again, all other default substitutions are still made.

When using this option, ensure that the leading \$ character in the Verilog is preceded by an escaped identifier so that V2LVS can read it correctly.

See the discussion of escaped identifiers under “[Expressions](#)” on page 745 for more details.

- `-cb`

Specifies to use a Calibre CB license.

- `-cfg config_file`

Specifies a configuration file that allows subcircuit scoping of lower level SPICE blocks that are incorporated into a top-level Verilog design. See “[Enable Subcircuit Scoping](#)” on page 754.

- `-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 “[Creating LVS Box Subcircuits](#)” on page 765.

- `-h`

Prints a help message.

- `-i`

Specifies that calls to subcircuits with pins be done in order, according to traditional SPICE rather than with \$PINS construct. Calibre nmLVS has a special SPICE extension for named pin connections in the \$PINS construct. This option is for non-Calibre applications only. See “[Simulation Output Generation](#)” on page 760 for details.

- `-ictrace`

Specifies to use an ICtrace license.

- `-incvmdir dir`

Specifies a directory containing files that are to be referenced by the ‘include directive in Verilog.

For example, consider the following Verilog file:

```
// top.v
`include "local_inc.v"
`include "project_inc.v"
module top (); /* top */
// ...
endmodule
```

where *local_inc.v* is in the directory *./inc_v* and *project_inc.v* is in the directory */project/verilog*. You can use the following V2LVS command line to include these files without putting their complete pathname into the Verilog source file:

```
v2lvs -v top.v -o top.spi -incvdir ./inc_v/local_inc.v \
-incvdir /project/verilog
```

- **-l verilog_lib_file**

Specifies the pathname 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 simulation. It is essential, however, that the Verilog modules match the SPICE modules on details like pin ordering and so forth.

Input files to this option that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment.

- **-log pathname**

Specifies the output pathname of the runtime log file.

- **-lsp spice_library_file**

Specifies a SPICE library filename using pin mode. The SPICE file is parsed for interface configurations. Pins with pin select ([]) annotation are kept as individual pins using escaped identifiers. This option may be specified multiple times. See “[Pin Mode](#)” on page 753 for details. See also the -l and -lsr arguments for this command.

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.

Input files to this option that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment. SPICE .INCLUDE statements are not processed. See [v2lvs::write_output](#) for that capability.

- **-lsr spice_library_file**

Specifies SPICE library filename using range mode. The SPICE file is parsed for interface configurations. Pins with pin select ([]) annotation are assembled into Verilog ranges. This option may be specified multiple times. See “[Range Mode](#)” on page 752 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 under “[V2LVS Used With SPICE Library Files](#)” on page 751.

Input files to this option that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment. SPICE .INCLUDE statements are not processed. See [v2lvs::write_output](#) for that capability.

- **-n**
Specifies unconnected pins to receive numbered connections, starting with 1000.
By default, unconnected pins are not specified in the SPICE netlist because Calibre nmLVS interprets missing pin connections as unconnected pins. This option causes V2LVS to generate explicit connections to unconnected nets. See “[Unconnected Pins](#)” on page 722 for details.
- **-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 output channel is STDOUT. Appending the *output_spice_file* with either .gz or .Z causes the file to be compressed with gzip or compress, respectively, if they are available in your environment.
- **-p *prefix***
Specifies that a prefix is added to the Verilog gate-level primitive cells. This allows construction of specific SPICE subcircuits that correspond to the Verilog gates used. This option can help to resolve naming conflicts in certain circumstances. See “[Primitive Instances](#)” on page 742 for details.
- **-rnt *renamed_text_file***
Specifies to output a renamed text file that is included with the Calibre rule set. The rules in the renamed text file avoid text shorts in layout feedthrough cells. See [v2lvs::write_output -renametext](#) for further details.
- **-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. This option may be specified multiple times. 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, -sk, -sl, -sn, and -so options can be useful in handling various supply net conventions in different design flows.
The supply net options are discussed further under “[Power and Ground Connections](#)” on page 727.

- **-s1 *powernet***
Specifies that the default global power is changed to *powernet*. The *powernet* is declared as a .GLOBAL net. See also -s0 option.
The supply net options are discussed further under “[Power and Ground Connections](#)” on page 727.
- **-sk**
Specifies that Verilog supply0 and supply1 nets are not connected to the global power and ground nets. This option generates .GLOBAL statements for local power and ground. See the -s0 and -s1 options.
The supply net options are discussed further under “[Power and Ground Connections](#)” on page 727.
- **-sl**
Specifies to create local supply0 and supply1 nets that are not connected to global power and ground. See the -s0 and -s1 options.
The supply net options are discussed further under “[Power and Ground Connections](#)” on page 727.
- **-sn**
Specifies that default power and ground signals should not be generated as .GLOBAL signals in the V2LVS output SPICE netlist. This option can be used in conjunction with the -sl option (no global signals for supply0 or supply1 signals) to suppress all global signal generation by V2LVS.
The supply net options are discussed further under “[Power and Ground Connections](#)” on page 727.
- **-so *filename***
Specifies a filename that provides instance- or module-specific power and ground overrides. See the -s0 and -s1 options. See “[Power and Ground Signal Overrides](#)” on page 731.
- **-t *svdb_dir***
Specifies that source template file information is added to the SVDB database. This is used in Calibre xRC. See “[Calibre xRC Source Template File](#)” on page 762.
- **-tcl [*cmd_file*]**
Specifies to run the Tcl interface. If -tcl is specified, then all other command line options are processed (if any) and the Tcl shell is started in interactive mode. If the *cmd_file* is provided, then all command line options are processed as before, and the *cmd_file* is executed in batch mode. See “[V2LVS Tcl Interface](#)” on page 766.
- **-u *unnamed_pin_prefix***
Specifies a prefix to add to unnamed pin connections in module instantiations, and to unnamed primitive and gate instantiations.

- `-undef_mod`

This option is deprecated. Specifies that undefined modules do not get special handling during processing. By default, the tool attempts to produce correct instance statements for undefined modules. The `-undef_mod` option disables this special processing. The option can reduce processing time, but the drawback is undefined modules may not receive the proper instance definitions in the output. See “[Undefined Modules](#)” on page 716.

- `-V1995`

Enables backward compatibility with Verilog-1995. This keyword should be used when there are Verilog-1995 modules that use Verilog-2001 keywords as identifiers. See “[Supported Verilog Syntax](#)” on page 833.

- `-V2001`

Enables use of Verilog-2001 syntax. This is the default mode. See “[Supported Verilog Syntax](#)” on page 833.

- `-w warning_level`

Controls the amount of warning message output. Possible level choices are these:

- 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).

- `-werror`

Treats warnings as errors. Verbosity is controlled by `-w` option.

- `[-- tcl_arg0 [tcl_arg1 ...]]`

Specifies arguments accessible from the \$argv list in a Tcl script when the `-tcl cmd_file` option is used. When specified, “`-- tcl_arg0 ...`” must be the last set of arguments on the command line. The `--` is literal, indicating the end of the command line options list and the start of arguments to be passed to the Tcl script.

When accessed within the `cmd_file`, the arguments passed from the command line can be referenced using code similar to this:

```
set arg0 [lindex $argv 0]
set arg1 [lindex $argv 1]
```

and so forth.

Description

V2LVS is used to translate Verilog to SPICE. V2LVS has a command line interface, which has been used historically. It is superseded by a Tcl interface discussed under “[V2LVS Tcl Interface](#)” on page 766.

Discussions of how to use the various V2LVS features are summarized under “[V2LVS](#)” on page 709.

V2LVS Errors and Warnings

The following table is arranged alphabetically by the first non-variable word that appears after the word ERROR: in the error message.

Table 16-8. V2LVS Errors

Error message	Resolution
Argument for the -a switch was missing.	Specify the array delimiter character with the -a option.
Argument for the -c switch was missing.	Specify the substitution characters for escaped identifier characters disallowed in SPICE. These follow the -c option.
Argument for the -l switch is missing.	Specify the path to the Verilog library file with the -l option.
Argument for the -lsp switch is missing.	Specify a filename with the -lsp option.
Argument for the -lsr switch is missing.	Specify a filename with the -lsr option.
Argument for the -u switch was missing.	Specify a prefix for unnamed pin connections with the -u option.
Argument for the -v switch is missing.	Specify the Verilog netlist filename with the -v option.
attribute_instance not allowed in Verilog-95	Attribute instances are new features in Verilog-2001.
Bounds of part-select into: <port name> are reversed.	The declared range direction does not match with the selection range direction. Example: <pre>module A(in[3:0]); input [0:3] in; endmodule</pre>
Can initialize only output port.	Input or inout type ports cannot be initialized in ANSI port declarations.
Cannot create the MASK SVDB DIRECTORY \<template_directory>	Mask SVDB directory specified with -t option is not writable.

Table 16-8. V2LVS Errors (cont.)

Error message	Resolution
End of source encountered before closing all `endif directives.	Ensure all `ifdefs are balanced with `endifs.
Error: Can only specify 1, 2 or 4 character string with -c.	Three characters are specified with -c switch. There must be one, two, or four.
Error: Invalid ANSI port in <i>filename</i> <line number>.	Message is preceded by the reason.
Error: Invalid input/output/input port declaration in <filename> <line number>.	Message is preceded by the reason.
Error processing include statement at <filename>:<line_number>.	Ensure the included file is present.
Failed to open include file <filename>.	Ensure the included file specified with the -s option can be opened.
File name with the -o switch was missing.	Specify the pathname of the desired output netlist with the -o option.
File name with the -t switch was missing.	Specify an existing SVDB directory with the -t option.
Identifier is too long.	Shorten identifier to one within the supported V2LVS limit (2048).
Identifiers in port expression must all be the same mode: <identifier>	Specifying a port expression like the following module A causes this error: <pre>module(.A({p1,p2})); input p1; output p2;</pre> <p>p1 and p2 must have the same mode (input or output).</p>
<name> instantiates named port <port>.	Ensure that <port> is in the declaration for the module being called.
<name> instantiates new port <port>.	Ensure that <port> is in the declaration for the module being called.
Integer expression required in port or bit select: <string>	Port select or bit select expression must contain an integer value.
Input Verilog design not specified. No output generated.	An input Verilog design must be specified in order for output to be generated.
Invalid Inout Declaration in <filename> : <line_number>	Fix Inout declaration to conform to Verilog standard.
Invalid Input Declaration in <filename> : <line_number>	Fix Input declaration to conform to Verilog standard.

Table 16-8. V2LVS Errors (cont.)

Error message	Resolution
Invalid Net Declaration in <filename> : <line_number>	Fix declaration to conform to Verilog standard.
Invalid or unsupported Parameter Declaration in <filename> : <line_number>	Fix declaration to conform to Verilog standard.
Invalid Output Declaration in <filename> : <line_number>	Fix declaration to conform to Verilog standard.
Invalid Port References in <filename> : <line_number>	Fix declaration to conform to Verilog standard.
Line continuation character at end of file.	A line continuation character inside a macro definition occurs at the end of a file. Fix the macro definition.
-lsl requires consistent use of array delimiters for pin indices: <port_name> at <filename> : <line_number>	SPICE file contains pin definitions that V2LVS cannot convert into a port range. Use a Verilog library, or fix SPICE pin definitions.
-lsl requires numeric use of bit-select values: <port_name> at <filename> : <line_number>	SPICE file contains pin definitions that V2LVS cannot convert into a port range. Use a Verilog library, or fix SPICE pin definitions.
Module instantiation <instance> has too many pins <pin_list>.	Correct the pins in the instance.
Multiple -s0 switches supplied—only one is allowed.	Specify one -s0 option.
Multiple -s1 switches supplied—only one is allowed.	Specify one -s1 option.
Multiple declarations of net: <net>	Specify only one declaration of the net.
Multiple declarations of port: <port>	Specify only one declaration of the port.
Multiple references to the same named port: <port name>.	Specify only one reference to the port.
No module declaration for module <module> first encountered in module <module>. (-i switch requires Verilog declarations for all modules.)	Specify a module declaration in the Verilog module or do not use the -i switch.
Only one -o switch is allowed.	Specify one -o option.
Parameter <parameter> redefined in module <module>.	Remove one of the parameter definitions.

Table 16-8. V2LVS Errors (cont.)

Error message	Resolution
Part-select into: <port name> is out of bounds.	The declared size of a port is less than selection size. Example: <pre>module A(in[3:0]); input [2:0] in; endmodule</pre>
Port Declaration not present in module port declaration: <string>.	Specify a port declaration in the Verilog module.
Range Values for port declaration and net declaration must be the same: <value>	Make the range values consistent between the port and net.
Reference to undefined variable <variable>.	Fix the variable reference.
Substitution for the spice character "<char>" is not allowed. Allowed characters are "\$" "," "=" and "\".	The -c option only allows replacement of the four characters shown in the error message.
Spice Parse error: <filename> : <line_number>	Fix SPICE syntax in file.
Structural Reference to unsupported expression <expression>	V2LVS does not support conversion of the encountered expression.
-t pathname too long.	Use a shorter pathname for the output file.
Trouble writing to template file.	The SVDB file specified with -t option has incorrect permissions or format. Fix the SVDB file.
Unexpected end of file \n.	Complete the file with valid Verilog syntax.
Unknown switch or invalid argument <argument>.	Specify a valid argument.
Unsupported expression in port connection <expression>.	Remove the unsupported expression.
Unsupported expression in port range declaration <expression>.	Remove the unsupported expression.
Unterminated comment block starting at <string>	Properly terminate the comment block.
User Defined Primitive <primitive> not translated	UDPs are not translated by V2LVS. Remove the UDP.
Warning level following the -w switch was missing.	Specify a warning level with the -w option.
Warning level following the -w switch was incorrect.	Specify a warning level from integers 0 through 4.

Table 16-8. V2LVS Errors (cont.)

Error message	Resolution
Warnings being treated as errors.	Resolve all warnings or remove -werror switch.

Table 16-9 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 16-9. V2LVS Warnings

Warning message	Description
ANSI port declaration not allowed in Verilog-95.	ANSI port declarations is Verilog-2001 feature. The filename and line number are reported.
Behavioral Construct in <filename> : <line_number>.	Only purely structural modules are translated. (warning level 1)
Behavioral Construct near <token>.	Behavioral Constructs needs to be calculated on runtime. As V2LVS is a static translator, it does not support any behavioral construct.
<module> / <instance> calls array of unknown dimension <port> in undeclared module <called_module>.	Provide port information for called module using Verilog library (-l) or SPICE library (-lsp or -lsr). (warning level 1)
<module> / <instance> calls port <port> in undeclared module <called_module>	Provide port information for called module using Verilog library (-l) or SPICE library (-lsp or -lsr). (warning level 1)
Duplicate instance name <name1> found in module <module name> while doing case-insensitive lookup.	Verilog is case sensitive whereas SPICE is not. V2LVS warns when it catches instance names that differ only in their case.
Duplicate port/net name <name1> found in module <module name> while doing case-insensitive lookup.	Verilog is case sensitive where as SPICE is not. V2LVS warns when it catches port or net names which differ only in their case.
Encountered non-sequential numeric use of bit-select values: <port> at <filename>:<line>. Port range values may not be consistent.	Provide SPICE file for -lsr option that has consistently ordered port bit select values. For example: A[0] A[1] A[2], not A[1], A[0] A[2]. (warning level 1)
Identifier <identifier> renamed to <identifier> (use -b to avoid stripping '\').	By default, V2LVS strips the '\' character from escaped identifiers. -b avoids this. (warning level 3)
<specify parameter declaration with range> ignored.	V2LVS ignores constructs not necessary for conversion to SPICE.
<string> in <filename> : <line_number> ignored.	<line_number> contains a Verilog construct that is not supported in a purely structural context. (warning level 4)

Table 16-9. V2LVS Warnings (cont.)

Warning message	Description
<token> ignored.	The <token> is something that V2LVS ignores. Correct or remove the <object> as necessary.
Ignoring unsupported variable identifier in variable list at <string>.	Variable is a selected name (A.B.C) not supported by V2LVS. (warning level 1)
Ignoring unsupported variable identifier in variable port list.	V2LVS ignores the initialization values with ANSI output ports.
<module> / <instance> instantiates new port <port> in undeclared module <called_module>	Provide port information for called module using Verilog library (-l) or SPICE library (-lsp or -lsr). (warning level 2)
Instantiating call to gate level primitive: <primitive>	Verilog calls a gate level primitive. (warning level 2)
local parameter with range ignored. parameter with range ignored.	V2LVS does not support parameters or local parameters with ranges. The range is ignored.
Macros with arguments not supported by V2LVS: <macro> at <filename> : <line_number>	V2LVS does not support `define macros with arguments. (warning level 1)
Module instantiation <string> / <string> has pin mismatches with module <string>	Port ranges on the call site do not match ranges on the port of the called module. (warning level 3).
Multiple declarations of module <name> in <filename>.	Module is declared multiple times. (warning level 1)
No module declaration for module <name> first encountered in module <name>.	A call was made to an undeclared module. (warning level 2).
output_net_type not allowed in Verilog-95.	Output port net type (wire, tri, tri1, etc.) specification is Verilog-2001 feature. The filename and line number are reported.
output reg not allowed in Verilog-95.	Declaring output port as a reg is new feature in Verilog-2001. The filename and line number are reported.
output_variable_type not allowed in Verilog-95.	Declaring output port as a integer or time is new feature in Verilog-2001. The filename and line number are reported.

Table 16-9. V2LVS Warnings (cont.)

Warning message	Description
Port <port_name> is mentioned in the non-contiguous order in the spice subckt <subckt_name> in the file <spice_file>. Treating non-contiguous references as separate ports.	<p>A SPICE subcircuit has a non-contiguous port order, such as the following code:</p> <pre>.subckt TOP A[3] A[1] X A[2] A[0]</pre> <p>Here it can be seen that port A is mentioned in a non-contiguous order. If this SPICE file is parsed in range mode then V2LVS treats non-contiguous ports as different ports. As a result two different ports are written: A[3:1] and A[2:0].</p>
Positional call to undeclared module <module> pin order will match Verilog call.	A call was made to an undeclared module using a positional instance call. (warning level 1)
range is not allowed for specify parameter declaration in Verilog-95.	Range specification after the specparam keyword is used in Verilog-2001. The filename and line number are reported.
Redefinition of defined value: <value> at <filename> : <line_number>	A `define macro was redefined. (warning level 1)
The -s switch is missing the file name for the SPICE Include file.	An argument is missing.
The -s0 switch is missing the name for SUPPLY0.	An argument is missing.
The -s1 switch is missing the name for SUPPLY1.	An argument is missing.
Unsupported compiler directive <directive> ignored.	Compiler directive is not supported by V2LVS and is being ignored. (warning level 3).
Unknown compiler directive or undefined macro <string> ignored.	Compiler directive or macro is not supported by V2LVS and is being ignored. (warning level 3).
V2LVS does not support instance ranges <string>.	Unsupported Verilog syntax. (warning level 1)
Verilog-95 syntax error: @* Verilog-95 syntax error: @(*)	Event control using @* or @(*) is used in Verilog-2001. The filename and line number are reported.
Verilog-95 syntax error: '='.	Initializing variables during declaration is used in Verilog-2001. The filename and line number are reported.
Verilog-95 syntax error: automatic.	“automatic” is used in Verilog-2001. The filename and line number are reported.

Table 16-9. V2LVS Warnings (cont.)

Warning message	Description
Verilog-95 syntax error: multi-dimensions.	Multi-dimensional variables are used in Verilog-2001. The filename and line number are reported.
Verilog-95 syntax error: net_type.	Input and inout ports can have net type (wire, tri, etc.) in Verilog-2001. The filename and line number are reported.
Verilog-95 syntax error: signed	“signed” is used in Verilog-2001. The filename and line number are reported.
Verilog-95 syntax error: unsigned	“unsigned” is used in Verilog-2001. The filename and line number are reported.
*.CALIBRE v2lvs warning: in <verilog_file> line <N>, Verilog assignment <instance_pin> is missing the port <net> in the subcircuit <name>.	This is a netlist warning that appears when there is a missing port connection in the Verilog netlist that references a SPICE library subcircuit.

V2LVS Tcl Shell Messages

This section contains runtime messages issued by the V2LVS Tcl shell commands.

The general form of a syntax error message is this:

wrong # args: should be *command_name options*

This error is issued when an incorrect number of arguments is used.

The following are general error and warning messages for the Tcl shell.

Table 16-10. V2LVS Tcl Shell Errors and Warnings

Message	Description
Error messages	
ERROR: Output spice already generated. Cannot generate output more than once.	The v2lvs::write_output command can only be used once in a command sequence.
ERROR: Substitution for the spice character “<character>” is not allowed. Allowed characters are '\$' ',' '=' and '/'.	The specified character in a v2lvs::substitute_chars command is not allowed.
Warning messages	
WARNING: Configuration file already set to <file_name>. Resetting it to <file_name>.	The v2lvs::set_config_file command has been called more than once. The latest invocation is used.

Table 16-10. V2LVS Tcl Shell Errors and Warnings (cont.)

Message	Description
WARNING: First array delimiter already set to "<character>". Resetting it to new value "<character>".	The -a option is specified on the command line and the v2lvs::set_array_delimiter command is used. The first delimiter is renamed.
WARNING: For the cell <cell_name> override assign cell already set to <old_subckt_name>. Resetting it to <new_subckt_name>.	If two v2lvs::override_assign commands specify the same -module cell name, the second command is used for the module name replacement.
WARNING: Module <module_name> not found. Ignoring v2lvs::add_formal_port command.	If v2lvs::add_formal_port is used and a module specified by -under is undefined the command is ignored.
WARNING: Output file already set to <file_name>. Resetting it to <file_name>.	The -o option is specified on the command line and the v2lvs::write_output command is used. The output file is renamed.
WARNING: Override assign module <module_name> should have two ports. Writing .CONNECT statement.	The SPICE definition for the circuit defined with the -spice_subckt parameter must have two ports only. If not, V2LVS does not override the assign operation.
WARNING: PEX template already set to <template_name>. Resetting it to <template_name>.	The -t option is specified on the command line and the v2lvs::create_pex_template command is used. The template file is renamed.
WARNING: Primitive prefix already set to <prefix_name>. Resetting it to <prefix_name>.	The -p option is specified on the command line and the v2lvs::set_prefix command is used. The prefix is renamed.
WARNING: Second array delimiter already set to "<character>". Resetting it to new value "<character>".	The -a option is specified on the command line with a second delimiter and the v2lvs::set_array_delimiter command is used with a second delimiter. The second delimiter is renamed.
WARNING: Spice definition for the override assign cell <cell_name> not found. Writing .CONNECT statement.	A SPICE definition for the circuit defined with -spice_subckt parameter was not found. In this case, V2LVS does not override the assignment operation.
WARNING: Substitution for character "<character>" already set to "<string>". Resetting it to new value "<string>".	The -c option is specified on the command line and the v2lvs::substitute_chars command is used. The prefix is renamed.

The following are debugging messages associated with the [v2lvs::set_verbose_level](#) command. The level corresponds to the verbosity of the debug messages. A given level value causes all messages that correspond to that value, and all values less than it, to be issued.

Table 16-11. v2lvs::set_verbose Debugging Messages

Level	Message
1	ADDPINET_DEBUG_MSG: For the instance “<instance_name>” in the module “<module_name>”, net “<net_name>” is connected to the added formal port “port_name”. This port is added through v2lvs::add_formal_port rule.
1	ADDPINET_DEBUG_MSG: For the instance “<instance_name>” in the module “<module_name>”, Net “<net_name>” is connected to the formal port “<port_name>” due to v2lvs::add_actual_port rule.
1	ADDPINET_DEBUG_MSG: Formal port “<port_name>” could not be added to module “<module_name>” as it is already present.
1	ADDPINET_DEBUG_MSG: Net “<net_name>” found in the module “<module_name>” down the hierarchy. Formal port “<port_name>” added to the module “<module_name>” due to v2lvs::add_formal_port rule.
1	ADDPINET_DEBUG_MSG: Net “<net_name>” found in the module “<module_name>”. Formal port “<port_name>” added to the module “<module_name>” due to v2lvs::add_formal_port rule.
1	ADDPINET_DEBUG_MSG: Port “<port_name>” not found in the module “<module_name>”. For the instance “<instance_name>” in the module “<module_name>” v2lvs::add_actual_port rule could not be applied.
2	ADDPINET_DEBUG_MSG: Net “<net_name>” specified in the -connect_formal_actual switch, not found in the module “<module_name>”.
3	ADDPINET_DEBUG_MSG: For the instance “<instance_name>” in the module “<module_name>”, no v2lvs::add_actual_port rule is applied to formal_port “<port_name>” of the module “<module>”.

Chapter 17

E2LVS

The EDIF-to-LVS (E2LVS) program translates an EDIF (Electronic Design Interchange Format) netlist into a SPICE netlist suitable for Calibre nmLVS/Calibre nmLVS-H comparison against a layout.

E2LVS Overview	853
E2LVS Command Line Syntax	855
Differences Between EDIF and SPICE	858
Translated EDIF Syntax Elements.....	863
Netlist Example	869

E2LVS Overview

E2LVS translates a structural EDIF 2 0 0 design into an equivalent SPICE netlist for use as input to Calibre nmLVS/Calibre nmLVS-H.

The equivalent netlist is an extended form of traditional SPICE. The section “[SPICE Syntax Conventions](#)” on page 634 describes the netlist format and extensions used in an Calibre 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.

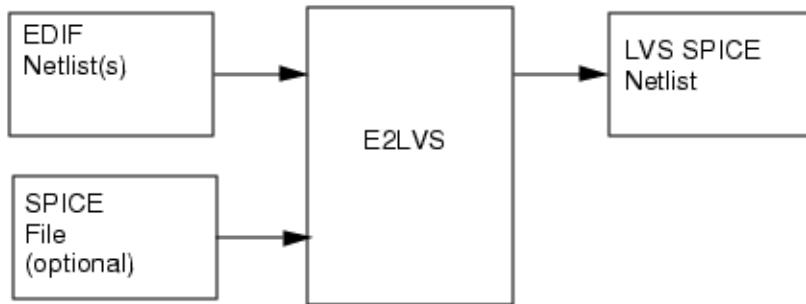
The correct EDIF netlist file order is necessary because a cell definition must be parsed before it is instantiated in another cell. Thus, if a cell in EDIF fileA uses a cell defined in EDIF fileB, then fileB must be listed before fileA in the list of EDIF files. If the files are out of sequence, E2LVS 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. This figure illustrates the E2LVS flow:

Figure 17-1. 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 861 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.

To use the output SPICE file as input into LVS, you must include certain specification statements in the rule file. These statements are identified in the section “[Rule File](#)” on page 32.

Complete licensing information is provided under “[Licensing: Physical Verification Products](#)” in the *Calibre Administrator’s Guide*.

E2LVS Command Line Syntax

E2LVS is an EDIF to SPICE translator.

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]
       [-h]
```

Arguments

Entering e2lvs -h prints a help line.

- **-e *edif_input_file***

Specifies the location of an EDIF input file. Input files to this option that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment.

- **-l *input_list_file***

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” on page 863 for more information. Any filenames listed within the *input_list_file* that have either .gz or .Z extensions are read automatically if gzip or uncompress are in your environment. The *input_list_file* itself may not be compressed.

- **-o *output_file***

A required argument set that specifies the location of the translated output SPICE netlist file. Appending the *output_file* with either .gz or .Z causes the file to be compressed with gzip or compress, respectively, if they are available in your environment.

- **-s *spice_input_file***

An optional argument set that specifies the location of an optional input SPICE file.

- **-sb [cell_file]**

An optional argument that specifies to translate EDIF cell names in the *cell_file* to SPICE black boxes. If you do not use the *cell_file* parameter, then all empty EDIF cells are translated into SPICE black boxes, that is, empty subcircuits. Otherwise, the *cell_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 in the file specified using the -s option. EDIF cells are considered empty when the contents parameter in the view statement is undefined.

Input files to these options that have either .gz or .Z extensions are read automatically if gzip or uncompress are available in your environment.

- **-ss [cell_file]**

An optional argument that specifies to translate EDIF cell names in the *cell_file* to SPICE subcircuit calls. Any cells from the *cell_file* that are empty are translated as primitive subcircuits.

If you use the -ss option without the *cell_file* parameter, then any empty cells are assumed to be defined as SPICE subcircuits in the -s option file, and the SPICE subcircuit calls are generated. This is the default behavior.

Input files to these options that have either .gz or .Z extensions are read automatically if gzip or uncompress are available in your environment.

- **-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 [].

- **-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 in disallowed 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 the hash symbol (#). It is substituted in place of the disallowed SPICE characters comma (,), equals sign (=), and dollar sign (\$). The second character delimiter is optional. When specified, it substitutes for the slash (/) character, which can be a disallowed 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 brackets ([]), angle brackets (<>), braces ({ }), and parentheses (). Refer to section “[Arrays and Bundles](#)” on page 859 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 862 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 are found during translation.
- **-p**
Specifies to pass EDIF properties on instances as SPICE properties. Only numeric 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 sub-properties. 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 these:
 - 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 license.
- **-ictrace**
Specifies to use an ICtrace license.

Examples

1. This example specifies an input EDIF file and an output SPICE file:

```
e2lvs -e design.edif -o design.spi
```

2. This example specifies 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 delimiters, a SPICE input file, the substitution character in disallowed 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
```

Differences Between EDIF and SPICE

EDIF format is different from SPICE in fundamental ways. It is important to understand EDIF syntax and how this is mapped to SPICE.

Identifiers	858
Name Scope	858
Arrays and Bundles.....	859
Untranslated EDIF Syntax	860
EDIF Cell Names Versus SPICE Subcircuit Names.....	861
Rename and Name Conflict	862

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 dollar sign (\$), equals sign (=), comma (,), and forward slash (/) characters. The / restriction is relaxed for \$PINS strings.

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 option.

Related Topics

[Differences Between EDIF and SPICE](#)

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. Objects are not allowed 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 861 for more information.

Arrays and Bundles

EDIF arrays describe a number of objects (net, port, or instance) of the same type with the same name.

For example, this statement:

```
(port (array inbus 32) (direction input))
```

creates 32 input ports named inbus that are indexed from 0 through 31.

The member statement allows access to each inbus object. For example:

```
(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 statement. A netBundle cannot contain other net bundles.

Arrays and bundles are flattened during translation. That is, given an array, the equivalent number of SPICE objects are created and named to represent the individual members of the array. This is applicable to all EDIF net, port, and instance arrays.

In the following example, translating the EDIF array inbus to SPICE involves appending [i], where i is the array index and [] are the default array delimiters:

EDIF

```
(port (array inbus 32) (direction input))
```

SPICE

```
inbus[0] inbus[1] ... inbus[31]
```

By default, an underscore (_) is inserted between the bundle name and the listed items when bundled names are flattened.

The following example shows how a portBundle is translated into SPICE:

EDIF

```
(portBundle pbExample
  (listOfPorts (port a)
    (port b)
    (port (array c 3))))
```

SPICE

```
pbExample_a pbExample_b pbExample_c[0] pbExample_c[1] pbExample_c[2]
```

You can change default array and bundle delimiters by invoking E2LVS with the -a and -b command line options, respectively. In addition, you can recognize special bus delimiters in the case of renamed arrays with the -r command line option. For example, the statement:

```
(port (array (rename A_8_TO_5 "A<8:5>") 4))
```

would expand to A[8], A[7], A[6], A[5] by using the -r “[]” command line option.

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.

Untranslated EDIF Syntax

Many aspects of EDIF syntax are not translated into SPICE because they are irrelevant in SPICE and to Calibre nmLVS.

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, and technology statements.
- portInstance statement.
- Multidimensional arrays.
- Keyword mappings.

Related Topics

[Differences Between EDIF and SPICE](#)

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 option.

You can resolve the first issue by mapping the EDIF cell name to a SPICE subcircuit name. Using the following EDIF input, the cell name fcell:

```
(library cmoslib
  (cell fcell
    (view my_netlistview (viewType NETLIST)
    ...
  )))
```

results in the following SPICE output:

```
.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 option.

The second issue, how to correlate SPICE subcircuit names when invoking E2LVS with the -ss or -sb option, is addressed by mapping the EDIF cell names directly to SPICE subcircuit 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](#)” on page 863 for more information about EDIF cell definitions and implementations.

Related Topics

[Differences Between EDIF and SPICE](#)

Rename and Name Conflict

The rename statement extends an EDIF name by enabling you to associate a given string with it.

For example, in the statement:

```
(rename busA_0 "busA(0)")
```

the string busA(0) refers to EDIF name busA_0, which is an allowed name in SPICE.

When a renamed name includes a disallowed SPICE character, such as \$, E2LVS generates its equivalent SPICE name by substituting a # in place of the disallowed characters. For example:

```
(rename dollarbusB_31 "$busB_31")
```

The translated SPICE name for \$busB_31 is #busB_31.

In some cases, such as in SPICE pin names, the slash character (/) is valid and may be present to show hierarchy. Thus, the translator retains the / characters unless you specify otherwise with the -c command line option.

All EDIF rename statements are used in the generated SPICE netlist unless the name would be disallowed in SPICE. The command line -i option specifies to ignore the rename statement, and the original EDIF name is used in the generated SPICE netlist.

For example, the statement:

```
(port (rename bus22 "bus(22)"))
```

generates a SPICE pin name bus22 when -i is specified. Otherwise, the renamed names (such as bus(22)) are used in the generated SPICE netlist.

Related Topics

[Differences Between EDIF and SPICE](#)

Translated EDIF Syntax Elements

The following EDIF elements are translated to SPICE.

cell	863
edif	864
instance	865
joined	866
net and netBundle	866
port and portBundle	867
rename	868
status	869

cell

The EDIF cell statement corresponds to a SPICE subcircuit name.

The 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, like this:

```
.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](#)” on page 866 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. Therefore, EDIF files must be in the correct order for input into E2LVS when the -l command line switch is used.

Related Topics

[Translated EDIF Syntax Elements](#)

edif

The edif statement contains all the hierarchy and design information transmitted within a file.

The 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 these:

```
(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
```

Related Topics

[Translated EDIF Syntax Elements](#)

instance

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 859. 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 recognized 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.

Related Topics

[Translated EDIF Syntax Elements](#)

joined

The EDIF joined statement is used to specify port connections.

The joined statement syntax is as follows:

```
(joined {portRef | portList | globalPortRef})
```

E2LVS translates all parameters. 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.

EDIF ports and nets within a cell can be connected to one another or to ports of other cells by using the joined statement. These connections are represented as follows in the SPICE netlist:

```
* .J <net> ==<port>
* .J ==<port1> ==<port2> ==<port3>
```

Ports are preceded by == to distinguish them from nets.

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 output either as a subcircuit net 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.

Related Topics

[Translated EDIF Syntax Elements](#)

net and netBundle

The EDIF net and netBundle statements declare nets and net groups.

Net statements correspond with SPICE node names. The 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*. Refer to the sections “joined” on page 866 and “instance” on page 865 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 netBundle statements and translates individual net names to SPICE node names. Refer to the section “[Arrays and Bundles](#)” on page 859 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*.

port and portBundle

The EDIF port and portBundle statements declare ports and port groups.

Ports are the basic means of communicating signal information between cells. The port syntax is as follows:

```
(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.

A port array is declared by prefixing the port name with the reserved word array. 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:

EDIF

```
(port (array c 3))
```

SPICE

```
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]
```

Related Topics

[Translated EDIF Syntax Elements](#)

rename

The EDIF 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 disallowed SPICE characters, a warning message is issued and # is substituted in place of the disallowed characters. Sometimes the original EDIF name can cause a conflict with a rename construct. In this case, like name collisions during array and bundle expansion, LVS creates a unique name for the duplicated 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 859 for more information.

If the -i command line option 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.

```
(port
  (rename portA0 "portA(0)")
  (direction Input))
```

Related Topics

[Translated EDIF Syntax Elements](#)

status

The EDIF status statement provides historical information about the EDIF file.

The syntax is as follows:

```
(status {written | comment | userData})
```

E2LVS ignores the *comment* and *userData* parameters. The *written* parameter is translated into a comment in the generated SPICE netlist.

There can be multiple *written* statements. The syntax is as follows:

```
(written timeStamp {author | program | dataOrigin | property | comment  
| userData})
```

E2LVS ignores the *property*, *comment*, and *userData* parameters. The *timeStamp* parameter is required. The *author*, *program*, and *dataOrigin* parameters are translated into comments in the generated SPICE netlist.

This example shows how a *written* statement is translated into SPICE:

```
(written  
  (timeStamp 1986 11 30 22 7 29)  
  (author "J. Smith")  
  (dataOrigin "Liverpool")  
  (program "EdifWriter"))
```

SPICE translation:

```
*WRITTEN:  
* timestamp 1986 11 30 22 7 29  
* author J. Smith  
* dataOrigin Liverpool  
* program EdifWriter
```

Related Topics

[Translated EDIF Syntax Elements](#)

Netlist Example

This section provides a sample EDIF netlist and its SPICE equivalent, as translated by E2LVS.

Assume the following EDIF netlist:

```
(edif DT
  (edifVersion 2 0 0)
  (edifLevel 0)
  (keywordMap (keywordLevel 0))
  (status(written
    (timestamp 97 12 09 10 49 26)
    (program "XLDF2EDIF X8830" (Version "V00.03")))
  (external CellLib
    (edifLevel 0)
    (technology(numberDefinition))
    (cell GCG001
      (cellType GENERIC)
      (view LOGIC(viewType NETLIST)
        (interface
          (port P2(direction INOUT))
          (port P1(direction INOUT)))
      )
    )
    (cell GCG002
      (cellType GENERIC)
      (view LOGIC(viewType NETLIST)
        (interface
          (port P3(direction INOUT))
          (port P1(direction INOUT))
          (port P2(direction INOUT)))
      )
    )
  )
)
```

```

(library Logic
  (edifLevel 0)
  (technology(numberDefinition))
  (cell (rename b31BLK_h0 "b31BLK-0")
    (cellType GENERIC)
    (comment
      "A FILE:b31BLK-0 CDATE:970714 UDATE:970714 USEQ:000 ATRB:IBLK
JISSOU:JSK1"
      "B MODEL:MDL1"
      "PE LOC:VAS"
      "FILE:0 CELL:2 NET:5 PORT:6"
    )
    (view LOGIC(viewType NETLIST)
      (interface
        (port (rename P_q1 "P?1") (direction INOUT))
        (port (rename P_q2 "P?2") (direction INOUT))
        (port (rename P_q3 "P?3") (direction INOUT))
        (port (rename P_q4 "P?4") (direction INOUT))
        (port (rename P_q5 "P?5") (direction INOUT))
        (port (rename P_q6 "P?6") (direction INOUT))
      )
      (contents
        (instance In1
          (viewRef LOGIC (cellRef GCG002(libraryRef CellLib))))
        (instance In2
          (viewRef LOGIC (cellRef GCG001(libraryRef CellLib))))
        (net i1
          (joined
            (portRef P1(instanceRef In1))
            (portRef P_q1)
            (portRef P_q4)))
        (net i2
          (joined
            (portRef P2(instanceRef In1))
            (portRef P_q2)))
        (net i3
          (joined
            (portRef P1(instanceRef In2))
            (portRef P_q3)))
        (net o2
          (joined
            (portRef P3(instanceRef In1))
            (portRef P_q5)))
        (net o3
          (joined
            (portRef P2(instanceRef In2))
            (portRef P_q6)))
      )
    )
  )
  (cell (rename b31REG_h0 "b31REG-0")
    (cellType GENERIC)
    (comment
      "A FILE:b31REG-0 CDATE:970714 UDATE:970714 USEQ:000 ATRB:IREG
JISSOU:JSK1"
      "B MODEL:MDL1"
      "PE LOC:VAS"
      "FILE:1 CELL:0 NET:7 PORT:6"
    )
  )
)

```

```
)  
(view LOGIC(viewType NETLIST)  
(interface  
  (port P1001(direction INOUT))  
  (port P1002(direction INOUT))  
  (port P1003(direction INOUT))  
  (port P2001(direction INOUT))  
  (port P2002(direction INOUT))  
  (port P2003(direction INOUT))  
)  
(contents  
  (instance B1  
    (viewRef LOGIC (cellRef b31BLK_h0)))  
  (net ib1  
    (joined  
      (portRef P1001)  
      (portRef P_q1(instanceRef B1))))  
  (net ib2  
    (joined  
      (portRef P1002)  
      (portRef P_q2(instanceRef B1))))  
  (net ob1  
    (joined  
      (portRef P2001)  
      (portRef P_q4(instanceRef B1))))  
  (net ob2  
    (joined  
      (portRef P2002)  
      (portRef P_q5(instanceRef B1))))  
  (net ob3  
    (joined  
      (portRef P2003)  
      (portRef P_q6(instanceRef B1))))  
  (net GND  
    (joined  
      (portRef P_q3(instanceRef B1))))  
)  
)  
)  
(design TOP  
(cellRef b31REG_h0 (libraryRef Logic))  
)  
)
```

Note that CellLib is external, implying 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 this:

```
*  
.INCLUDE cells.spi  
*  
* edif DT  
* edifVersion 2 0 0  
* edifLevel 0  
* keyWordLevel 0  
* status  
* written  
* timestamp 97 12 09 10 49 26  
* program "XLDF2EDIF X8830"  
* Version "V00.03"  
.SUBCKT b31BLK-0 P?1 P?2 P?3 P?4 P?5 P?6  
*.XPINS  
XIn1 GCG002 $PINS P1=i1 P2=i2 P3=o2  
XIn2 GCG001 $PINS P1=i3 P2=o3  
*.J i1 P?1  
*.J i1 P?4  
*.J i2 P?2  
*.J i3 P?3  
*.J o2 P?5  
*.J o3 P?6  

```

Related Topics

[E2LVS Overview](#)

Chapter 18

Rule File Analyzer

The Rule File Analyzer is used for detailed rule file analysis. In particular, it is used for examining layer derivations and connectivity chains. It supports both SVRF and TVF formats.

calibre -svrf	876
Rule File Analyzer Commands	878

calibre -svrf

The Calibre Rule File Analyzer is a utility for SVRF and TVF rule file analysis.

The analyzer is used to trace layer derivations, layer connectivity, rule check components, and device definitions.

Usage

calibre -svrf *rule_file_name*

Arguments

- **-svrf**
A required argument that specifies to start the Rule File Analyzer shell.
- ***rule_file_name***
A required name of an SVRF or TVF rule file. It need not be a complete rule file, but it should be syntactically correct.

Description

After successfully compiling the rule file, “calibre -svrf” starts an interactive session and presents this command prompt:

SVRF>

The environment is an interactive shell that accepts rule file analysis commands. Command returns are printed to the terminal as text strings.

The rule file analyzer may also be used from a script, as shown in the Examples. The input rule file must pass the compiler for the analyzer script to run successfully.

The built-in commands use all upper- or lowercase text. Mixed text case for the built-in commands results in an invalid command message.

For encrypted rule file content, the Rule File Analyzer does not return decrypted layer operations.

The Rule File Analyzer reserves a Calibre nmDRC, Calibre nmLVS, or Calibre RVE license, in that order.

Examples

Here is an example C-shell script for running the tool:

```
#! /bin/csh -f
calibre -svrf rules << EOF
FILE outfile
FREEZE
QUIT
EOF
```

The output file is a syntactically complete rule file with current conditional compilation settings. Code that is not conditionally compiled is excluded.

Related Topics

[Rule File Analyzer Commands](#)

Rule File Analyzer Commands

Rule file analyzer commands give information about layers, rule checks, and connectivity and Device operations in a rule file.

For all of these commands, the following terms apply:

A *Layer statement* is an instance of the [Layer](#) statement in the rules.

A *layer definition* is a line in the rules that produces a layer from a layer operation. For example, this is the layer definition of a layer called “out”.

```
out = in1 AND in2
```

A *derived layer* is a layer that appears in a layer definition.

A *layer derivation* is the set of Layer statements and layer definitions that produce a derived layer. It provides the complete pedigree of a layer. For example, this shows the derivation of out_extents:

```
LAYER in1 1
LAYER in2 2
out = in1 AND in2
out_extents = EXTENTS out
```

In some cases, the Layer statements are omitted from a layer derivation and are simply assumed to exist. (A compiler error would result if they did not.)

Table 18-1. General Commands

Command	Description
FILE	Directs output from or to the shell terminal.
FREEZE	Writes the complete rule file for the current state of pre-processor statements.
HELP	Writes a usage message for all commands.
QUIT	Exits the analyzer shell.

Table 18-2. Layer Listing Commands

Command	Description
CHECK	Writes layer derivation information for all layers involved in a specified rule check.
CONN	Writes layer definition and connectivity information for a specified layer.

Table 18-2. Layer Listing Commands (cont.)

Command	Description
DEV	Writes layer derivation information for all layers involved in the specified Device statement.
LIST ALIASES	Writes all layers that have more than one layer name assigned in the rule file.
LIST LAYERS	Writes layer names appearing in the rule file.
LAYER	Writes the derivation trace of a specified layer.

Table 18-3. Connectivity, Device, and Rule Check Listing Commands

Command	Description
LIST CHECKS	Writes the rule check names contained in the rule file.
LIST CONNECTS	Writes all the Connect and Sconnect statements in the rule file.
LIST DEVICES	Writes all the Device statements in the rule file.

CHECK

Rule File Analyzer command. The layer derivations are reported in the same way as the [LAYER](#) command.

Writes the derivation trace of all layers used in the specified rule check *name*.

Usage

CHECK *name*

Arguments

- *name*

A required name of a rule check. This may be a DRC, ERC, or DFM rule check using the customary rule check syntax.

Return Values

Newline-delimited set of strings indicating the layer derivation trace. Periods (.) indicate the number of levels in a derivation. Asterisk characters (*) indicate the derivation of the layer is shown previously. Braces { } indicate nesting of derivations.

Examples

```
SVRF> CHECK rule5
"rule5::<1>" = INT "metall1" <= 0.00126 {
. "metall1" = OR "metall1" {
. . LAYER "metall1" 8 { } }
```

Related Topics

[LIST CHECKS](#)

CONN

Rule File Analyzer command.

Writes the definition of the input layer followed by the Connect or Sconnect statements involving the layer.

Usage

CONN *layer*

Arguments

- *layer*

A required name of a layer.

Return Values

Newline-delimited set of strings indicating the definition of the specified *layer*, followed by the Connect and Sconnect statements involving the layer. These are some special annotations in the output:

<==== [S]CONNECT LAYER — Indicates the layer definition.

// filename:N — Indicates the name of the rule file where the line is found, followed by the line number.

<==== NO CONNECTIVITY — Indicates the layer is not involved in connectivity.

Examples

This example shows the connectivity info for a derived layer:

```
SVRF> CONN ntap
"ntap" = "ndiff" AND "nwell" <==== [S]CONNECT LAYER
CONNECT "metall1" "ntap" BY "contact" // rules:63
```

The CONNECT statement is found in the file called rules on line 63.

Related Topics

[LIST CONNECTS](#)

[LIST LAYERS](#)

DEV

Rule File Analyzer command.

Writes the layer derivation traces for all layers in the Device statement indicated by the **number**.

Usage

DEV *number*

Arguments

- **number**

A required, non-negative integer that corresponds to the ordinal number of a Device statement in the rule file. The **Device** statements are indexed starting at 0 and are ordered in sequence from top to bottom in the (possibly concatenated) rule file.

Return Values

Newline-delimited set of strings indicating the layer derivation trace. Periods (.) indicate the depth of derivation. Asterisk characters (*) indicate the derivation of the layer is shown previously. Braces { } indicate nesting of derivations.

Description

The **number** can be taken from the // D = N annotation in the **LIST DEVICES** command output or the \$D=n property value in an extracted SPICE netlist. The layer derivations are reported in the same way as the **LAYER** command.

Examples

```
SVRF> LIST DEVICES
DEVICE mn "ngate" "ngate"(g) "nsd"(s) "nsd"(d) "pwell"(b) // D = 0
(rules.extract:103)
DEVICE mp "pgate" "pgate"(g) "psd"(s) "psd"(d) "nwell"(b) // D = 1
(rules.extract:124)
SVRF> DEV 0
"ngate" = "poly" AND "ndiff" {
. "poly" = OR "poly" {
. . LAYER "poly" 4 { } }
. "ndiff" = "diff" AND "nplus" {
. . "diff" = OR "diff" {
. . . LAYER "diff" 2 { } }
. . "nplus" = OR "nplus" {
. . . LAYER "nplus" 5 { } } } }
"ngate" = "poly" AND "ndiff" {
. "poly" = OR "poly" {
. . LAYER "poly" 4 { } }
. "ndiff" = "diff" AND "nplus" {
. . "diff" = OR "diff" {
. . . LAYER "diff" 2 { } }
. . "nplus" = OR "nplus" {
. . . LAYER "nplus" 5 { } } } }
"nsd" = "ndiff" NOT "poly" {
. "ndiff" = "diff" AND "nplus" {
. . "diff" = OR "diff" {
. . . LAYER "diff" 2 { } }
. . "nplus" = OR "nplus" {
. . . LAYER "nplus" 5 { } } }
. "poly" = OR "poly" {
. . LAYER "poly" 4 { } } }
"nsd" = "ndiff" NOT "poly" {
. "ndiff" = "diff" AND "nplus" {
. . "diff" = OR "diff" {
. . . LAYER "diff" 2 { } }
. . "nplus" = OR "nplus" {
. . . LAYER "nplus" 5 { } } }
. "poly" = OR "poly" {
. . LAYER "poly" 4 { } } }
"pwell" = OR "pwell" {
. LAYER "pwell" 1 { } }
```

Related Topics

[LIST LAYERS](#)

FILE

Rule File Analyzer command.

When specified without a filename, redirects output to the shell's terminal. When a filename is used, directs output to that file.

Usage

FILE [*filename*]

Arguments

- *filename*

An optional pathname of a file to which the responses of commands are directed.

Return Values

None.

Related Topics

[Rule File Analyzer Commands](#)

FREEZE

Rule File Analyzer command.

Writes the complete rule file for the current state of pre-processor statements.

Usage

FREEZE [ENV]

Arguments

- ENV

An optional keyword that causes #PRAGMA_ENV preprocessor statements to be written for defined environment variables referenced in the rule file.

Return Values

Rule file code for the current state of pre-processor directives.

Description

This command writes all rule file contents (including any [Include](#) files) that are active for the pre-processor directives (#DEFINE or #UNDEFINE statements) found in the rule file.

Encrypted content is printed verbatim. The output of this command is valid rule file code.

Conditional statements such as #IFDEF blocks are not written to the output.

Environment variable values are not substituted in the output by this command.

The ENV option causes #PRAGMA_ENV statements to be printed for all environment variables referenced in the active content of the rule file. Only defined environment variables that are used for SVRF or TVF commands are affected. Environment variables defined for Tcl commands that are not a part of TVF do not receive #PRAGMA_ENV statements.

Examples

Example 1

Assume this rule file is used for input:

```
LAYER a 1
LAYER b 2
LAYER c 3
#ifndef COND1
#DECRYPT
%2?~F0`QI_!7-G/6#)Y3OBXNIAF"P]1<LVXC"M>V,FF5._U!#>AM+'?)1.V)/
R1EVKC4;B1!!
'1[+I<VZ$SQ3M\A,]JR#TQ!!"Y9=:*!H>VR0JY@*: \Q"+G!!!!"
D7-647H1S3W%[8<! !#Q!!!!"18]P8JEMQ.]SB.C?SQY72Q!!"
XI4=/V%D^+.C[\$V]&>HRQ!!"&GO*4^9@C73S7)T%RSP.*!!!
#ENDCRYPT
#else
#DECRYPT
%2?~F0`QI_!7-G/6#)Y3OBXNIAF"P]1<LVXC"M>V,FF5._U!#>AM+'?)1.V)/
R1EVKC4;B1!!
5BI"V^>R2F^8*@]@<;9A?Q!!"NU"K/I=22&X(KA&%Z^KJ6#OY3AJNZI:0)D;E7)45[-%#!#
C=GY._6IHV+Y8*KYDYP9*A!!"-7NFCB&KUN^L<L%>X=5D="MD!RSRA+Z-\VQ#,,VP5RQ!#
R/Z9=IL?Z=K%),Y=-8`2!!!"HX^>G8PC&D006ZM>'`LH34.('G;@0@)G;S3>Z/YKM[!#!#
#ENDCRYPT
#endif
g = d OR e
h = f OR g
```

If the variable COND1 is not defined, the output of FREEZE command is this:

```
LAYER a 1
LAYER b 2
LAYER c 3
#DECRYPT
%2?~F0`QI_!7-G/6#)Y3OBXNIAF"P]1<LVXC"M>V,FF5._U!#>AM+'?)1.V)/
R1EVKC4;B1!!
5BI"V^>R2F^8*@]@<;9A?Q!!"NU"K/I=22&X(KA&%Z^KJ6#OY3AJNZI:0)D;E7)45[-%#!#
C=GY._6IHV+Y8*KYDYP9*A!!"-7NFCB&KUN^L<L%>X=5D="MD!RSRA+Z-\VQ#,,VP5RQ!#
R/Z9=IL?Z=K%),Y=-8`2!!!"HX^>G8PC&D006ZM>'`LH34.('G;@0@)G;S3>Z/YKM[!#!#
#ENDCRYPT
g = d OR e
h = f OR g
```

The second encrypted block of SVRF is written out verbatim, and the first block is suppressed because it is not selected by the #IFDEF statement.

Example 2

Assume this rule file is used for input:

```
LAYOUT PATH $LAYOUT
LAYOUT PRIMARY top
LAYOUT SYSTEM GDSII
DRC RESULTS DATABASE $DRCDB
LAYER A 1
LAYER B 2
RES { A OR B }
```

Assume that the environment variables are defined as follows:

```
LAYOUT=lay.gds
DRCDB=drcdb
```

The FREEZE command echoes the rule file exactly as it is written. The variable values are not substituted. The file does not compile unless the environment variables LAYOUT and DRCDB are defined.

The FREEZE ENV command echoes the following rule file:

```
// Default values for defined environment variables
#PRAGMA ENV DRCDB "drcdb"
#PRAGMA ENV LAYOUT "lay.gds"
//
LAYOUT PATH $LAYOUT
LAYOUT PRIMARY top
LAYOUT SYSTEM GDSII
DRC RESULTS DATABASE $DRCDB
LAYER A 1
LAYER B 2
RES { A OR B }
```

This file compiles without any environment variables defined. The #PRAGMA_ENV values are used if the LAYOUT or DRCDB variables are not defined. Otherwise, defined variable values are used.

Example 3

These commands in a Rule File Analyzer script:

```
FILE compiled.rules
FREEZE
QUIT
```

generate a compiled rule file that can be used in a Calibre application. This can save compilation time as the rules have already gone through the compilation pre-processor.

Related Topics

[Pre-Processor Directives \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

HELP

Rule File Analyzer command.

Writes a usage message for all commands.

Usage

HELP

Arguments

None.

Return Values

Help message.

Related Topics

[Rule File Analyzer Commands](#)

LAYER

Rule File Analyzer command.

Writes the derivation trace of a specified layer.

Usage

LAYER *name*

Arguments

- *name*

A required layer name.

Return Values

Newline-delimited set of strings indicating the layer derivation trace. Periods (.) indicate the depth of derivation. Asterisk characters (*) indicate the derivation of the layer is shown previously. Braces { } indicate nesting of derivations.

Examples

Consider the following derivation:

```
LAYER A 1
LAYER B 2
C = A OR B
D = C NOT B
E = C AND D
```

The LAYER derivation trace for layer D is this:

```
D = C NOT B {
. C = A OR B {
. . A = OR A {
. . . LAYER A 1 { } }
. . B = OR B {
. . . LAYER B 2 { } }
. B = OR B { * }
```

Each Layer statement or layer definition is reported on a separate line, with the number of periods corresponding to the depth of the derivation.

If a layer was already reported, its definition and derivation are not repeated. Instead the asterisk character (*) is printed.

The braces enclose complete layer derivations. For example, the closing brace corresponding to the opening brace after C = A OR B is the brace at the end of the LAYER B 2 line. These braces completely enclose the derivation of the layer C, the first input layer necessary to derive layer D.

The next line begins the derivation of the layer B, which is the second input layer.

Related Topics

[LIST ALIASES](#)

[LIST LAYERS](#)

LIST ALIASES

Rule File Analyzer command.

Writes all layers that have more than one layer name assigned in the rule file. A layer has more than one name if all the layer names have identical derivations. The Calibre database constructor detects equivalent layers and optimizes the layer tree so that the layer is created only once. Any name in the group of aliases can be used in the rule file or in the Rule File Analyzer to refer to the layer.

Usage

LIST ALIASES

Arguments

None.

Return Values

Whitespace-delimited string of layer aliases. The format of the output is the following:

alias(*chosen_name*)

where *alias* is an alternate layer name for the *chosen_name*. The *chosen_name* is used by the Calibre tool as the primary name and is shown in the run transcript.

Examples

This shows an example output:

```
LIST ALIASES
thkoxngate_dev1(ngate_dev1) vtgngate_dev1(ngate_dev1)
```

In this case, *ngate_dev1* is the name chosen by the tool for the transcript, but the other two names are aliases in the rules.

Related Topics

[LAYER](#)

[LIST LAYERS](#)

LIST CHECKS

Rule File Analyzer command.

Writes the names of rule checks in the rule file. The optional arguments cause reporting of checks that would be selected in different Calibre invocation modes that support rule checks.

Usage

LIST CHECKS [ALL | DFM | DRC | ERC]

Arguments

- ALL
An optional keyword that specifies to list all the rule checks. This is the default.
- DFM
An optional keyword that specifies to list all checks executed by calibre -dfm or calibre -perc -lldl command lines.
- DRC
An optional keyword that specifies to list all checks executed by the calibre -drc command line.
- ERC
An optional keyword that specifies to list all ERC checks from LVS circuit extraction.

Return Values

Whitespace-delimited string of rule check names.

Examples

```
SVRF> LIST CHECKS
rule1      rule2      rule3
```

Related Topics

[CHECK](#)

LIST CONNECTS

Rule File Analyzer command.

Writes all Connect or Sconnect statements in the rule file, along with the rule file name and line number.

Usage

LIST CONNECTS

Arguments

None.

Return Values

Newline-delimited set of connectivity statements. Each line ends with the string:

```
// filename:line number
```

Examples

```
SVRF> LIST CONNECTS
CONNECT "metal1" "nsd" BY "contact" // rules:62
CONNECT "metal1" "ntap" BY "contact" // rules:63
CONNECT "metal1" "psd" BY "contact" // rules:64
CONNECT "metal1" "ptap" BY "contact" // rules:65
CONNECT "metal1" "poly" BY "contact" // rules:66
CONNECT "metal1" "metal2" BY "via" // rules:67
```

Related Topics

[CONN](#)

[LIST LAYERS](#)

LIST DEVICES

Rule File Analyzer command.

Writes all Device statements in the rule file, along with an ordinal number of each statement, the associated rule file name, and the line number.

Usage

LIST DEVICES

Arguments

None.

Return Values

Newline-delimited set of strings containing [Device](#) statements.

Each line ends with the string:

```
// D = integer (filename:line number)
```

The *integer* is an ordinal number starting from 0 indicating the sequence of Device statements in the rule file from top to bottom.

Examples

```
SVRF> LIST DEVICES
DEVICE mn "ngate" "ngate" (g) "nsd" (s) "nsd" (d) "pwell" (b) // D = 0
      (rules.extract:103)
DEVICE mp "pgate" "pgate" (g) "psd" (s) "psd" (d) "nwell" (b) // D = 1
      (rules.extract:124)
```

Related Topics

[DEV](#)

[LIST LAYERS](#)

LIST LAYERS

Rule File Analyzer command. To determine which, if any, layer derivations were optimized, use the [LIST ALIASES](#) command.

Lists currently active layer names in the rule file, including aliases.

Usage

LIST LAYERS [ALL | CONNECT | DEVICE]

Arguments

- **ALL**
An optional keyword that specifies to list all layers. This is the default.
- **CONNECT**
An optional keyword that specifies to list all layers needed for [Connect](#) or [Sconnect](#) statements.
- **DEVICE**
An optional keyword that lists only layers needed for [Device](#) statements.

Return Values

Whitespace-delimited string of layer names.

Examples

If the rule file contains these definitions:

```
X = A AND B
Y = A AND B
Z = A AND B
```

then the LIST LAYERS command lists layers A, B, X, Y, and Z. However, the Calibre transcript does not show the derivation of layers Y or Z, and in all subsequent references to these layers, the layer name X is used.

The LIST ALIASES command shows the chosen layer name (X) and all its aliases:

```
Y(X)  Z(X)
```

Related Topics

[CONN](#)

[LAYER](#)

QUIT

Rule File Analyzer command.

Exits the Rule File Analyzer.

Usage

QUIT

Arguments

None.

Return Values

None.

Related Topics

[Rule File Analyzer Commands](#)

Index

— Symbols —

- .GLOBAL statement, 287
- .MACRO statement, 696
- .SUBCKT statement, 284, 696
- [], 29
- {}, 29
- *.DIODE control statement, 656
- *.SEEDPROM control statement, 673
- |, 29

— A —

- Actual name (V2LVS), 771
- Advanced Device Properties (ADP), 316, 318
- Ambiguities
 - definition (LVS), 394
 - resolving, 394
- Analyze LVS report, 558
- Appropriate angles, 179
- Appropriate measurement criteria, 177
- Area-based filtering of layout databases, 96
- Array reference (AREF) output, 198
- ASCII DRC results database format, 236
- ASCII layout format, 107
- Attach statement, 261
- Auxiliary layer operations, 89

— B —

- Bad device reporting, 320
- Binary polygon file (BPF), 629
- Bipolar transistors, 387
 - filtering, 429
 - in SPICE, 690
 - parallel reduction, 421
- Bit rows, 515
- Bit structure, 513
 - core, 512
- BJT elements (Table), 690
- BNF, Backus-Naur Form
- Bold words, 28
- Boolean operations, 90

- two-layer, 90, 91
- BPF, *see* Binary polygon format
- BSIM4 device models, 330
- Built-in device types definition, 379
- Built-in property classification, 467

— C —

- Calibre nmLVS-H
 - command line examples, 61
- Calibre
 - verification overview, 25
- Calibre CB described, 76
- Calibre nmDRC
 - described, 25
 - invocation, 36
 - examples, 45, 66
- Calibre nmDRC-H
 - data storage, 143
 - efficiency of layer operations, 147
 - flat instantiations, 148
 - hcells and, 151
 - layer area printing, 150
 - results database, 246
 - statements specific to, 94
 - text mapping, 150
 - theory of operation, 93
- Calibre nmDRC-H, *see also* DRC
- Calibre nmLVS, 26
- Calibre nmLVS-H, 473
 - comparison, 496
 - hcells and, 474
- Calibre nmLVS-H, *see also* LVS
- CALIBRE_OLD_PASSIVE_MODELS
 - environment variable, 685
- Capacitors, 380
 - element (Table), 678
 - in SPICE, 677
 - parallel reduction, 415
 - property computations, 327
 - series reduction, 416

Case sensitivity, 375
ERC, 356
Cells
 duplicate, 97
 exclusion, 96
 FAUX BIN, 280
 hcells, 474
 correspondence file, 475
 cycles, 588
 statistics, 644
 ICV, 280
 parameterized, 503
 pushdown, 495
 renaming, 97
 SRAM bit-cell, 501
Check set, 116
Check text definition, 114
Checking large distances, 122
Circuit comparison in Calibre nmLVS-H, 496
Circuit extraction, 279
Circuit matching, 393
Clustered output, 171
CNET database format, 34
 source, 34
Command lines
 DRC, 36
 E2LVS, 855
 LVS, 45, 66
 V2LVS, 836
Command syntax, 28
Comments (rule check), 113
Commutative measurement for Opposite metric, 161
Compilation of rule file, 126
Component subtypes, 370
Component types, 370
Concurrency, 118
Conjunctive checks, 120, 175
Connect operation, 252
 Connect BY, 253
Connectivity extraction, 249
 assigning net names, 258
 connectivity statements, 252
 errors and warnings, 274
 layer operations, 251
LVS errors and warnings, 288
mask mode, 249
one-way connections, 256
recognizing electrical nets, 252
removing connectivity, 274
report (LVS), 545
suppress hierarchy modifications, 277
suppress warnings, 273
Virtual Connect statements, 286
with hcells, 491
Constraints, 157
 for output suppression, 175
Contact checks, 122
Control statements
 *.DIODE, 656
 *.SEEDPROM, 673
 in SPICE, 647
 other, 676
Convert database format, 26
Correspondence files for hcells, 52
Courier font, 29
CPU TIME, 208, 213

— D —

Data flow
 DRC, 109
 LVS, 365
Databases
 ASCII, 107
 CNET, 34
 DRC results, 234
 GDSII, 102
 LEF/DEF, 33
 mask results, 631
 OASIS, 104
 OpenAccess (OA), 33
 pre-me rging, 97
 source, 34
 SPICE, 34, 633
Datatypes, 79
Debug rule checks using Copy, 113
Deep shorts, 438
Derived layer, 80
Derived layers, 82
 edge, 83
 error, 84

polygon, 82

Device recognition

- definition, 307

Device operation, 323, 370, 372

- BY NET keyword, 311
- BY SHAPE keyword, 311
- elements, 308

elements of a device statement, 308

example, 315

hierarchical, 322

ill-formed devices, 314

layer relations, 310

logic, 310

pin fill-in algorithm, 313

pin relations, 311

property computations, *see* Property computations, 325

reporting bad devices, 320

Device reduction

- defined, 396
- generic, 398

parallel bipolar transistors, 421

parallel capacitors, 415

parallel diodes, 420

parallel MOS transistors, 398

parallel resistors, 417

programs, 424

semi-series MOS transistors, 402

semi-split gates, 409

series capacitors, 416

series MOS transistors, 400

series resistors, 419

split gates, 405

tolerances, 424, 427

unequally reduced devices, 400

Devices

- bipolar transistors, 387
- built-in, 379
- capacitors, 380
- definition, 308
- diodes, 382
- filtering, 428
- ill formed, 314
- inductors, 390
- JFET transistors, 389

model names, 371

MOS transistors, 384

parallel and series reduction, 397

resistors, 386

signatures, 315

voltage sources, 391

Dimensional check operations, 89

- appropriate angles, 179
- appropriateness criteria, 177
- edge breaking, 182
- edge measurement, 156
- edge output clusters, 171
- edge-directed, 185
- error-directed, 185
- improving runtime, 121
- intersection criteria, 181
- key concepts, 156
- metrics, 159
- output suppression, 175
- polygon containment criteria, 184
- polygon-directed, 185
- region output, 187
- trivial edges, 172

Diodes, 382

- in SPICE, 680
- parallel reduction, 420
- property computations, 327

Disconnect statement, 274

Discrepancies in LVS report, 571

Disk-based layers, 203

Double pipes, 29

Drawn layer, 78

DRC

- data flow, 109
- ERC check selection, 364
- invocation examples, 45, 66
- maximizing capacity, 120
- minimizing runtime, 120
- required rule file statements, 33
- results database, 234
 - Calibre nmDRC-H, 246
 - empty rule checks, 118
 - format, 236
 - polygon segmentation, 125
 - precision, 235

- properties, 241
- results counts limits, 245
- transferring text, 114
- results limiting, 245
- results limits, 117
- summary report, 247
- DRC Cell Name statement, 241
- DRC Cell Text statement, 258, 280
- DRC Check Map statement, 194, 244, 246
- DRC Check Text statement, 238
- DRC Incremental Connect statement, 269
- DRC Incremental Connect Warning statement, 274
- DRC Keep Empty statement, 118
- DRC Map Text statement, 150, 201, 258
- DRC Maximum Results statement, 245
- DRC Maximum Vertex statement, 246
- DRC Print Area statement, 150
- DRC Print Perimeter statement, 150
- DRC results
 - in cell space, 241
 - management statements, 194
- DRC results database
 - ASCII format, 236
 - GDSII format, 194, 242
 - OASIS format, 194, 243
- DRC Results Database Precision statement, 235
- DRC Results Database statement, 236, 242
- DRC Summary Report statement, 247
- DRC Tolerance Factor statement, 189
- Duplicate .SUBCKT definitions, 698

— E —

- E2LVS
 - description, 853
 - netlist example, 869
 - overview, 27
 - translations, 863
 - untranslated EDIF syntax, 860
- Edge breaking, 182
- Edge faces, 83
- Edge measurement
 - description, 156
 - edge trimming, 158
 - region construction, 157
- Edge output clusters, 171
- Edge trimming, 158
- Edge-directed output, 186
- EDIF, 853
 - SPICE translation, 863
 - syntax, 858
- Efficiency
 - conjunctive checks, 121
 - hierarchical layer operations, 147
 - property computations, 334
 - rule file, 120
- Efficient layer operations, 122
- ELAPSED TIME, 213
- Element statements
 - .MACRO, 695
 - .SUBCKT, 695
 - BJT, 690
 - capacitor, 677
 - inductor, 683
 - JFET, 681
 - junction diode, 680
 - MOSFET, 685
 - resistor, 692
 - voltage source, 694
 - X, 699
 - Y, 708
- Empty rule checks, 118
- Enclosure operation, 153, 177
- Environment Variables
 - MGC_FDI_OA_VERSION, 106
- Environment variables, 32
 - CALIBRE_EXIT_MAXIMUM_RESULT_S, 245
 - CALIBRE_OLD_PASSIVE_MODELS, 678, 684, 685, 692
 - CALIBRE_PRINT_TIME_STAMPS, 214
 - echoed in logfile, 208
 - FDI, 105
 - ICC_PATH, 105
 - LD_LIBRARY_PATH, 105
 - MGC_CALIBRE_CELLMAP_FILE, 106
 - MGC_CALIBRE_DB_READ_OPTIONS, 105
 - MGC_CALIBRE_LAYERMAP_FILE, 105

-
- MGC_TMPDIR, 275
 - OA_HOME, 106
 - Equation-Based DRC
 - Advantages, 128
 - Overview, 128
 - Toolset, 135
 - Equation-Based DRC Examples
 - Wide Metal Spacing, 128
 - ERC
 - abort LVS run, 356
 - definition, 354
 - definition of path, 352
 - DRC execution, 364
 - LVS execution, 354
 - net printing, 360
 - polygon printing, 360
 - PRINT keyword, 353
 - PRINT NETS operation, 360
 - PRINT POLYGONS keyword, 360
 - results, 359, 546
 - in cell space, 241
 - rule check selection, 356
 - statements, 351
 - usage examples, 363
 - ERC Cell Name statement, 241
 - ERC Pathchk statement, 290
 - ERC Results Database statement, 546
 - ERC Summary Report statement, 361, 546
 - Error tolerances, 100
 - Error-directed dimensional check operations, 185
 - Error-directed layer operations, 84, 200
 - Errors and warnings in connectivity extraction, 274
 - Escaped identifiers, 838
 - Establishing connectivity, 249
 - Euclidean metric, 161
 - Exclude Cell statement, 97
 - Exclude statements, 200
 - Expand Cell statement, 95
 - Expand Cell Text statement, 258, 260, 280
 - Expand Text operation, 258
 - Expression evaluation failure (DFM Property), 318
 - External operation, 153, 177
 - Extracted layout device pin names, 372
 - Extracted netlist names, 278
- F —**
- Fall-through expressions (DFM Property), 319
 - False enclosure reduction, 189
 - False measurement algorithm, 190
 - False notch errors, 189
 - FAUX BIN cells, 280
 - fdi2gds, 26
 - fdi2oasis, 26
 - Feedthrough nets, 437
 - Filter unused devices
 - bipolar transistors, 429
 - MOS transistors, 428
 - Flag shapes statements, 199
 - Flat instantiations, Calibre nmDRC-H, 148
 - Flatten Cell statement, 95
 - Flatten operation, 148
 - Floating nets, 433
 - Floating pin, 285
 - Font conventions, 28
 - Formal name (V2LVS), 771
 - Free-standing layer operation, 81
- G —**
- Gates, 439
 - AOI, 444
 - CMOS, 442
 - in LVS report, 572
 - INV, 442, 452
 - logic injection, 509
 - inverter chains, 518
 - inverters, 517
 - multiplexer, 525
 - parallel, 521
 - series, 519
 - simple, 523
 - transmission multiplexer, 532
 - XOR and XNOR, 527
 - NAND, 442, 453
 - NMOS, 452
 - NOR, 443, 453
 - OAI, 444, 454
 - SDW, 446, 454
 - SM group, 448, 456

-
- SMP group, 449
SPDW, 447, 455
SPM group, 450, 456
SPUP, 447
SUP, 445
- GDSII
DRC results, 194, 242
layout database format, 102
layout input control, 96
- Geometry promotion in Calibre nmDRC-H, 95
- Grid snapping, 200
- Ground names, 356
- Ground nets, 430
- H —**
- Hcell statement, 278, 474, 496
Hcells, 474
Calibre nmDRC-H treatment, 151
connectivity extraction, 491
correspondence
file, 52
cycles, 588
pins, 496
statistics, 644
-hdbflex argument, 40
Heavy font, 28
Hierarchical device recognition, 322
Hierarchical DRC, *see* Calibre nmDRC-H
Hierarchical LVS, *see* Calibre nmLVS-H
Hierarchical versus flat database processing, 94
Hierarchy modification, suppressing, 277
High-short resolution, 498
-hyper opt ion, 72
-hyper option, 40, 53
Hyperscaling, 146
in connectivity extraction, 276
- I —**
- ICV cells, 280, 281
Ill-formed devices, 314
Implicit layer definitions, 82
Inductors, 390
element (Table), 684
in SPICE, 683
Initial correspondence points, 369
Injected components, 505, 507
- Input errors in LVS, 590
Instances
definition, 308
names, 373
pins and pin names, 372
Internal operation, 153, 178
Intersection criteria, 181
Interval constraints, 175
Inverters, *see* Gates
Italic font, 28
IXF file format, 624, 627
- J —**
- JFET, 389
element (Table), 682
in SPICE, 681
Junction diode elements (Table), 680
- L —**
- Label Order statement, 261
Labels, *see* Net names
Layer area printing
Calibre nmDRC-H, 150
Layer constructors, 89
Layer definition, 80
Layer definitions
implicit, 82
Layer Directory statement, 203
Layer Map statement, 79, 150
Layer of origin, 92
Layer operations, 89
hierarchical, 491
layer constructors, 90
layer selectors, 90
node-preserving, 90
redundancy elimination, 119
scheduling, 119
Layer Resolution statement, 200
Layer selectors, 90
Layer sets, 78
Layer statement, 78
Layer statistics, 213
Calibre nmDRC, 214
Calibre nmDRC-H, 215
flat count, 211
hierarchical count, 211

LVHEAP, 217
Layers, 78
Layout Allow Duplicate Cell statement, 97
Layout Base Layer statement, 95, 495
Layout Cell List statement, 277
Layout data input transcript section, 208
Layout databases
 area-based filtering, 96
 formats
 ASCII, 107
 CNET, 34
 GDSII, 102
 OASIS, 104
 SPICE, 34
 magnification, 98
 text, 258
Layout Depth statement, 102
Layout Input Exception RDB statement, 101
Layout Input Exception Severity statement, 101
Layout Magnify statement, 98
Layout Merge On Input statement, 97
Layout Path statement, 102
Layout Polygon statement, 283
Layout Preserve Case statement, 258, 260, 357
Layout Preserve Cell List statement, 53
Layout Primary statement, 102, 242
 wildcards, 100
Layout Process Box Record statement, 103
Layout Process Node Record statement, 103
Layout Rename Text statement, 258
Layout System statement, 102, 466
Layout Text File statement, 259
Layout Text statement, 258, 259
Layout versus schematic, *see* LVS
Layout Window statements, 96
LEF/DEF database format, 33
Library files (V2LVS), 714
Limiting DRC results counts, 245
Logic gates, 439
 CMOS and-or-invert, 444
 CMOS gates, 442
 CMOS inverter, 442
 CMOS NAND, 442
 CMOS NOR, 443
 CMOS or-and-invert, 444
 CMOS serial pulldown, 446
 CMOS serial pullup, 445
 CMOS serial-parallel pulldown, 447
 CMOS serial-parallel pullup, 447
 high level serial-parallel structures, 450
 NMOS gates, 452
 NMOS inverter, 452
 NMOS NAND, 453
 NMOS NOR, 453
 NMOS or-and-invert, 454
 NMOS serial pulldown, 454
 NMOS serial-parallel pulldown, 455
 NMOS serial-parallel structure, 456
 restrictions, 411
 serial-parallel structure of MP or MN devices, 449
 series of MD or ME devices, 456
 series of MP or MN devices, 448
Logic injection, 505
 gate recognition, 509
 in LVS report, 573
 pin swappability, 510
Logically equivalent pins
 default pin swapping for devices, 462
 definition, 462
LPH file format, 628
LVHEAP statistics, 217
LVS
 built-in device types, 379
 circuit comparison, 367
 circuit extraction, 249
 component subtypes, 370
 component types, 370
 data flow, 365
 device reduction, 396
 discrepancy types, 575
 ERC
 errors, 356
 execution, 354
 results, 546
 rule check selection, 356
 filtering unused devices, 428
 initial correspondence points, 369
 instance pins and pin names, 372

logic injection, 505
logically equivalent pins, 462
MS and MF schematic devices, 392
net and instance names, 373
overall comparison results, 562
ports and port names, 374
power and ground nets, 430
recommended configuration, 368
report, 549
 analysis, 558
 connectivity extraction errors and
 warnings, 560
 detailed instance connections, 559, 587
 discrepancies, 571
 discrepancy types, 575
 error and warning messages
 analysis, 558
 errors, 563
 incorrect objects, 559
 information and warnings, 559, 599
 injected components, 573
 instance pins, 574
 input errors, 590
 instance identification, 571
 instance pin identification, 573
 layout input errors, 590
 listingconventions, 570
 logic gate identification, 572
 naming errors, 560
 net identification, 571
 non-identical signal pins, 585
 numbers of objects after
 transformation, 559
 overall comparison results, 562
 port identification, 571
 power and ground nets, 588
 primary messages, 563
 secondary messages, 563
 source input errors, 590
 SPICE netlist errors and warnings, 560
 stamp errors, 560
 Trace Property tolerance errors, 584
 unconnected instance pin identification,
 575
 unmatched elements, 596
 warnings, 568
required rule file statements, 33
sample rule file, 61
short isolation results, 546
transcript, 538
user given names, 374
warnings, 599

LVS Abort On Softchk statement, 289
LVS All Capacitor Pins Swappable statement,
 462
LVS Box statement, 428, 495, 498
LVS Cell Supply statement, 374
LVS Compare Case statement, 357, 475
LVS Cpoint statement, 369
LVS Device Type statement, 372, 383
LVS Exclude Hcell statement, 475, 482
LVS Expand Seed Promotions statement, 482
LVS Expand Unbalanced Cells statement, 479
LVS Filter statement, 428
LVS Filter Unused Bipolar statement, 430
LVS Filter Unused MOS Transistors statement,
 429
LVS Filter Unused Option statement, 428
LVS Globals Are Ports statement, 374
LVS Ground Name statement, 373, 430, 496
LVS Inject Logic statement, 506
LVS Isolate Shorts statement, 301, 546
LVS Netlist Box Contents statement, 498
LVS Netlist Unnamed Box Pins statement, 498
LVS Power Name statement, 373, 430, 496
LVS Preserve Box Cells statement, 498
LVS Preserve Box Ports statement, 501
LVS Preserve Parameterized Cells statement,
 505
LVS Property Resolution Maximum statement,
 394
LVS Push Devices statement, 482
LVS Recognize Gates statement, 405, 411,
 439, 496
LVS Recognize Gates Tolerance statement,
 440
LVS Reduce group of specification statements,
 424
LVS Reduce Parallel Bipolar statement, 421
LVS Reduce Parallel Capacitors statement, 415

LVS Reduce Parallel Diodes statement, 420
LVS Reduce Parallel MOS statement, 398
LVS Reduce Parallel Resistors statement, 417
LVS Reduce Semi Series MOS statement, 402
LVS Reduce Series Capacitors statement, 416
LVS Reduce Series MOS statement, 400
LVS Reduce Series Resistors statement, 419
LVS Reduce Split Gates statement, 405
LVS Reduce statement, 398
LVS Reduction Priority statement, 397
LVS report
 Detailed Instance Connections, 587
LVS Report Option statement, 289
LVS Report statement, 54, 549
LVS softchk function, 295
LVS Softchk statement, 201, 289

— M —

Magnifying layout databases, 98
MALLOC, 538
Mask connectivity extraction, 249
Mask layout, 370
Mask results output, 194
Mask SVDB Directory statement, 58, 559, 624
Measurement metrics, *see* Metrics
Measurement regions, 156
Merged layers, transferring net names to, 261
Metrics, 159
 Euclidean, 159
 Opposite, 159
 special considerations, 160
 Opposite Extended, 159
 Opposite Extended Fsymmetric, 163
 Opposite Extended Symmetric, 163
 Opposite Extended1, 163
 Opposite Extended2, 163
 Opposite Fsymmetric, 163
 Opposite Symmetric, 159, 163
 Opposite1, 163
 Opposite2, 163
 Square, 159
 Square Orthogonal, 164
Minimum keyword, 29
Model names, 371
MOS transistors, 383
 element (Table), 686

filtering, 428
in SPICE, 685
parallel reduction, 398
property computations, 328
semi-series reduction, 402
series reduction, 400
MS and MF schematic devices, 392
MT and MTflex modes of operation, 146
-mtflex switch, 43
Multiplexer, *see* Gates
Multithreaded modes of operation, 146

— N —

Negative edge-directed output, 186
Net Area Ratio Print operation, 239
Netlisting of hierarchical ports, 284
Net-preserving operations, *see* Node-preserving operations

Nets

 feedthrough, 437
 floating, 433
 in LVS comparison, 430
 isolated, 430
 names, 259, 373
 label attachment, 261
 layout database text objects, 258
 specifying, 259
 passthrough, 433
 power and ground, 374, 430

Node-preserving operations, 90

Non-commutative measurements, 159

NXF file format, 626, 627

— O —

OASIS

 DRC results, 194
 DRC results database format, 243
 layout database, 104

Open circuit, 265

OpenAccess database format, 33

Operations

 connectivity-related layer operations, 251
 described, 32
 Device, 370, 371, 372
 in Calibre nmLVS-H, 491
 layer, 89

-
- Opposite Extended metric, 161
 - Opposite metric, 161
 - special considerations, 160
 - Opposite Symmetric and FSymmetric metrics, 164, 166
 - Optimization
 - DRC runtimes, 120
 - rule file, 118, 119
 - Original layer, 78
 - Overdriven master host, 228
 - P —**
 - Parallel device reduction, 397
 - Parallel reduction
 - bipolar transistors, 421
 - capacitors, 415
 - diodes, 420
 - MOS transistors, 398
 - resistors, 417
 - Parameterized cells, 503
 - Parentheses, 29
 - Partner properties, 468
 - Passthrough nets, 433
 - Path definition (ERC), 352
 - Performance optimization, 120
 - Pins, 282, 372
 - fill in algorithm for devices, 313
 - floating, 285
 - hierarchical, 496
 - injected instances, 508
 - swappability, 462, 496
 - with logic injection, 510
 - Pipes, 29
 - Point-to-point measurement output, 174
 - Polygon containment criteria, 183
 - edge breaking, 182
 - overview, 184
 - Polygon segmentation, 125
 - Polygon-directed dimensional check
 - operations, 185
 - Polygon-directed output, 187
 - Port Depth statement, 258, 283
 - Port Layer Merged Polygon statement, 282, 630
 - Port Layer Polygon statement, 282, 374, 630
 - Port Layer Text statement, 258, 282, 374
 - Ports, 282
 - depth, 284
 - port names, 374
 - text objects, 282
 - hierarchical netlisting, 284
 - trivial, 438
 - unattached, 285
 - Positive edge-directed output, 186
 - Power and ground nets, 374, 430
 - names, 356
 - Power nets, 430
 - Precision statement, 98, 242
 - Properties
 - DRC results databases, 241
 - missing and unknown, 423
 - partner, 468
 - tracing, 467
 - W/L, 468
 - Property computations
 - debugging, 341
 - default computations, 326
 - description, 325
 - devices, 326
 - efficiency considerations, 334
 - for capacitors, 327
 - for diodes, 327
 - for resistors, 329
 - MOS transistors, 328
 - structure, 332
 - units of measurement, 331
 - user defined, 330
 - Q —**
 - Query server, 559
 - Quotation marks, 29
 - R —**
 - RDB, 237
 - REAL TIME, 208, 213
 - Recognizing electrical nets
 - Connect By operation, 253
 - Connect operation, 252
 - hierarchical netlisting, 284
 - hierarchical port text objects, 283
 - hierarchical processing, 283
 - ports and pins, 282

-
- shapes on a single layer, 253
 - summary, 252
 - transferring logical information to merged layers, 261
 - REGION keyword, 187
 - Register file bit structure, 533
 - Required rule file statements, 32
 - Resistors, 386
 - element (Table), 692
 - in SPICE, 692
 - parallel reduction, 417
 - property computations, 329
 - series reduction, 419
 - Resolution statement, 200
 - Rule checks
 - comments, 113
 - conjunctive, 120
 - debugging, 113
 - empty, 118
 - ERC, 356
 - exclusion, 116
 - results limits, 117
 - selection, 116
 - statements, 112
 - Rule file analyzer, 876
 - Rule files
 - basics, 32
 - DRC required statements, 33
 - LVS required statements, 33
 - optimization, 118, 120
 - S —
 - Sconnect operation, 256
 - Secondary keywords, 155
 - Semi-series MOS transistor reduction, 402
 - Semi-split gate reduction, 409
 - Series device reduction, 397
 - capacitors, 416
 - MOS transistors, 400
 - resistors, 419
 - Shielding in connectivity extraction, 253
 - Short circuit, 265
 - Short isolation
 - results, 546
 - Signature-based circuit matching, 393
 - Signatures for devices, 315
 - Simple layer, 78
 - Slanted words, 28
 - Snap Offgrid statement, 200
 - Snap vertices to grid, 200
 - Soft connections
 - checks in DRC, 201
 - checks in LVS, 289
 - Source databases
 - accepted formats, 34
 - Source System statement, 34
 - Specification statements
 - Calibre nmDRC-H related, 94
 - described, 32
 - Disconnect, 274
 - DRC Cell Name, 241
 - DRC Cell Text, 258, 280
 - DRC Check Text, 238
 - DRC Incremental Connect, 269
 - DRC Incremental Connect Warning, 274
 - DRC Keep Empty, 118
 - DRC Map Text, 150, 258
 - DRC Print Perimeter, 150
 - DRC Results Database Precision, 235
 - DRC results management, 194
 - ERC Cell Name, 241
 - ERC group, 351
 - ERC Pathchk, 290
 - ERC Results Database, 546
 - ERC Summary Report, 546
 - Exclude Cell, 97
 - Exclude group, 200
 - Expand Cell Text, 258, 260, 280
 - Flag group, 199
 - Hcell, 278, 474, 496
 - Label Order, 261
 - Layer, 78
 - Layer Directory, 203
 - Layer Map, 79, 150
 - Layout Allow Duplicate Cell, 97
 - Layout Base Layer, 495
 - Layout Cell List, 277
 - Layout Depth, 102
 - Layout Input Exception RDB, 101
 - Layout Input Exception Severity, 101
 - Layout Magnify, 98

Layout Merge On Input, 97
Layout Path, 102
Layout Polygon, 283
Layout Preserve Case, 258, 260, 357
Layout Preserve Cell List, 53
Layout Primary, 100, 102
Layout Process Box Record, 103
Layout Process Node Record, 103
Layout Rename Text, 258
Layout System, 102, 466
Layout Text, 258, 259
Layout Text File, 259
layout text related, 261
Layout Window group, 96
LVS Abort On Softchk, 289
LVS All Capacitor Pins Swappable, 462
LVS Box, 428, 495, 498
LVS Cell Supply, 374
LVS Compare Case, 357, 475
LVS Cpoint, 369
LVS Device Type, 372, 383
LVS Exclude Hcell, 475, 482
LVS Expand Seed Promotions, 482
LVS Expand Unbalanced Cells, 479
LVS Filter, 428
LVS Filter Unused Bipolar, 430
LVS Filter Unused MOS Transistors, 429
LVS Filter Unused Option, 428
LVS Globals Are Ports, 374
LVS Ground Name, 373, 430, 496
LVS Inject Logic, 506
LVS Isolate Shorts, 301, 546
LVS Netlist Box Contents, 498
LVS Netlist Unnamed Box Pins, 498
LVS Power Name, 373, 430, 496
LVS Preserve Box Cells, 498
LVS Preserve Box Ports, 501
LVS Preserve Parameterized Cells, 505
LVS Property Resolution Maximum, 394
LVS Push Devices, 482
LVS Recognize Gates, 405, 411, 439, 496
LVS Recognize Gates Tolerance, 440
LVS Reduce, 398
LVS Reduce group, 424
LVS Reduce Parallel Bipolar, 421
LVS Reduce Parallel Capacitors, 415
LVS Reduce Parallel Diodes, 420
LVS Reduce Parallel MOS, 398
LVS Reduce Parallel Resistors, 417
LVS Reduce Semi Series MOS, 402
LVS Reduce Series Capacitors, 416
LVS Reduce Series MOS, 400
LVS Reduce Series Resistors, 419
LVS Reduce Split Gates, 405
LVS Reduction Priority, 397
LVS Report, 54, 549
LVS Report Option, 289
LVS Softchk, 289
Mask SVDB Directory, 58, 559, 624
Port Depth, 258, 283
Port Layer Merged Polygon, 282, 630
Port Layer Polygon, 282, 374, 630
Port Layer Text, 258, 282, 374
Precision, 98
Snap Offgrid, 200
Source System, 34
Text, 258, 259
Text Depth, 258, 259, 303
Text Layer, 258, 259
Text Print Maximum, 208, 211
Trace Property, 467
Unit Capacitance, 327
Unit Length, 242, 327
Unit Resistance, 329
Virtual Connect Box Colon, 287
Virtual Connect Box Name, 287
Virtual Connect Depth, 286
Virtual Connect Name, 287
Virtual Connect Report, 286
Virtual Connect Semicolon As Colon, 286
SPH file format, 628
SPICE
as layout database, 34
bipolar transistors, 690
capacitors, 677
comment-coded extensions, 639
comparison to EDIF, 858
control statements, 647
diodes, 680
inductors, 683

-
- JFET, 681
MOS transistors, 685
netlist
 notational conventions (Table), 634
 subcircuit statements, 695
resistors, 692
source database format, 34
string parameters, 640
subcircuits, 695
summary, 633
syntax check report, 619
syntax check results, 570
Verilog translation, 709
 voltage sources, 694
Split gates, 405
Square metric, 161
Square Orthogonal metric, 168
Square parentheses, 29
SRAM bit-cells, 501
Stamp operation, 257
Statistics
 Calibre nmDRC-H layers, 215
 DRC layers, 214
 layer, 213
 LVHEAP, 217
 object counts, 211
Stress effect properties, 330
String parameters in SPICE, 640
Subcircuit statements, 695
 duplicates, 698
 primitive, 697
Table, 696
Xcalls (Table), 699
Suppressing incremental connectivity
 warnings, 273
SVDB
 files, 623
 header, 623
 instance cross-reference (ixf) file
 flat, 624
 hierarchical, 627
 layout placement hierarchy (lph) file, 628
 net cross-reference (nxf) file
 flat, 626
 hierarchical, 627
 source placement hierarchy (sph) file, 628
 with Calibre RVE, 559
- T —**
- Text attachment, 261
Text Depth statement, 258, 259, 303
Text Layer statement, 258, 259
Text mapping in Calibre nmDRC-H, 150
Text Print Maximum statement, 208, 211
Text statement, 258, 259
Text usage, 258
Text-related statements, 258, 261
Texttypes, 79
Tolerances in device reduction, 424
Trace Property statement, 325, 467
Tracing method circuit matching, 393
Tracing properties, 467
Transcript, 206, 213
 layout data input section, 208
 LVS, 538
 rule file compilation section, 206
Trivial edges, 172
Trivial net (in connectivity extraction), 283
Trivial nets, 438
Trivial pins, 438
Trivial pins and ports (in connectivity
 extraction), 283
Trivial ports, 438
Two-layer Boolean operations, 90
- U —**
- Unattached ports, 285
Underlined words, 29
Unidirectional metrics, 163
Unit Capacitance statement, 327
Unit Length statement, 242, 327
Unit Resistance statement, 329
Units for device property computation, 331
Usage
 E2LVS, 855
 rule file analyzer (calibre -svrf), 876
 V2LVS, 836
Usage syntax, 28
User-defined device property calculations, 330
User-given names in LVS, 374
Utilities

E2LVS, 27, 853
fdi2gds, 26
fdi2oasis, 26
Foreign Database Interface (FDI), 26
V2LVS, 27, 709
Verilog-to-LVS, 27

— V —

V2LVS
actual name, 771
behavioral statements, 744
command line and Tcl command correspondence, 766
command line arguments, 836
declarations, 724
described, 709
errors and warnings, 843
expressions, 745
formal name, 771
generating Calibre xRC source template, 762
library files, 714
LVS Box subcircuits, 765
module instantiations, 718
modules, 714
overview, 27
primitive instances, 742
simulation output, 760
Tcl commands, 771
Tcl interface command line, 766
usage, 836
using SPICE libraries, 751
Verilog syntax, 833
without SPICE libraries, 753
without Verilog libraries, 758

Verilog

behavioral statements, 744
bit expressions, 720
calling conventions, 718
declarations, 724
expressions, 745
library files, 714
module instantiations, 718
modules, 714
primitive instances, 742
specify blocks, 745

syntax in V2LVS, 833
translation to SPICE, 709
user-defined primitives, 718
Verilog-to-LVS, *see* V2LVS
Virtual Connect statements, 286
with LVS Box, 287
Virtual nets, 280
Voltage sources, 391
element (Table), 694
in SPICE, 694

— W —

W/L partner properties, 468
Well enclosure device properties, 330
With Text operation, 258, 259

— X —

X calls, 699
X devices, 642
X+ devices, 392, 411
XOR, XNOR *see* Gates

— Y —

Y element, 708

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the `<your_software_installation_location>/legal` directory.

