

SIEMENS EDA

Calibre® 3DSTACK 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Unpublished work.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		
Introduction to Calibre 3DSTACK.....		11
Overview		11
3D-IC Description Language Usage Restrictions		16
Requirements		16
Documentation Conventions		17
Chapter 2		
Getting Started With Calibre 3DSTACK.....		19
Calibre 3DSTACK Invocation.....		20
calibre -3dstack		21
Invoking Calibre Interactive for Calibre 3DSTACK		23
Using Calibre 3DSTACK		25
Creating a Calibre 3DSTACK Rule File		25
Running a Calibre 3DSTACK Verification		28
Running a Calibre 3DSTACK Verification from the Command Line.....		29
Running a Calibre 3DSTACK Verification from Calibre Interactive		30
Running Calibre Interactive 3DSTACK From Calibre DESIGNrev		32
Calibre 3DSTACK Output		34
Assembly Views		35
Calibre 3DSTACK Results Analysis Examples.....		38
Highlighting DRC Results from a Calibre 3DSTACK Run.....		39
Opening a Calibre 3DSTACK DFM Database in Calibre RVE.....		41
Debugging Connectivity Errors in Calibre 3DSTACK		44
Using Path Isolation to Debug Shorts		49
Viewing Design Elements in a Schematic View		52
Verification Results Format		52
Calibre Interactive for Calibre 3DSTACK		54
Creating a Source Netlist		54
Specifying a Source Netlist		56
Specifying Report and Results Options		58
Specifying Assembly Instructions		59
Selecting Specific Checks		60
Writing a Calibre Interactive Customization File		61
Triggers in Calibre Interactive 3DSTACK		62
Chapter 3		
Command Reference		65
Rule File Format and Syntax		67
System Variables		69

Assembly and Configuration Commands	71
die	72
-layout	82
-lefdef	83
-layer_info	84
component	90
stack	94
connect	100
config	102
-layout_primary	104
-layer_props_file	105
-layout_case	107
-source_netlist	108
-report	111
-ignore_trailing_chars	113
-export_connectivity	115
-net_map	118
-import_net_map	119
-pin_map	120
-import_pin_map	122
-set_auto_rve_show_layers	124
-set_rve_cto_file	125
-svrf_specs	126
-units	127
-warning_severity	128
process	129
export_layout	131
Rule Check Commands	139
centers	142
connected	147
copy	158
custom_check	160
dangling_ports	164
dangling_no_text	166
density	167
enclosure	175
external	178
extra_ports	182
floating_pads	184
floating_texts	186
internal	188
locations	190
missing_ports	193
multi_texts	195
no_texts	197
offgrid_centers	199
overlap	201
same_size	204
select_checks	206

Table of Contents

unconnected_ports	208
unselect_checks	209
3dstack_block	211
Check Text Override Comments for Calibre 3DSTACK	213
Chapter 4	
Calibre 3DSTACK Utilities	217
System Netlist Generator	218
System Netlist Generator Flow and Invocation	220
Workflow	220
sng	222
System Netlist Generator Graphical User Interface	227
System Netlist Generator GUI Tips	227
Main Window	230
Properties Panel	233
Set Net Name Template	234
System Netlist Generator Configuration File Format	235
PLACEMENTS	236
CONNECTIVITY	237
EXPORTS	239
System Netlist Generator Commands	240
sng::create_journal_file	242
sng::export_db	243
sng::export_netlist	244
sng::new_db	245
sng::open_db	246
sng::save_db	247
sng::filter	248
sng::get_begin	250
sng::get_buses	252
sng::get_chips	253
sng::get_nets	254
sng::get_pins	255
sng::get_placements	256
sng::get_ports	257
sng::get_property	259
sng::get_property_names	260
sng::incr	262
sng::set_property	263
sng::sort	264
sng::add_pin	266
sng::connect	267
sng::create_chip	269
sng::create_placement	270
sng::import_chip	271
sng::remove_chip	273
sng::remove_pin	274
sng::remove_placement	275

sng::set_net_name_template	276
sng::unconnect	277
AIF Converter Reference	278
AIF Support	278
AIF Export File Format	280
3dstack::aif2gds	282
3dstack::aif2oasis	283
3dstack::aif2spice	284
Spreadsheet Converters	285
3dstack::ss2gds	286
3dstack::ss2oasis	288
3dstack::ss2spice	290
3dstack::xsi2gds	292
3dstack::xsi2oasis	295
3dstack::xsi2spice	298
Appendix A Examples	301
Calibre 3DSTACK Flow Example	302
Example Design Information	302
Preparing the Dies for Assembly	303
Creating the Calibre 3DSTACK Rule File	304
Writing Assembly Operations in the Calibre 3DSTACK Rule File	306
Setting Up Global Options	306
Defining Dies Used in the Assembly	307
Defining Layers	309
Defining Layer Connectivity	317
Placing the Dies	319
Writing Verification Checks in the Calibre 3DSTACK Rule File	323
Checking Connectivity (LVS)	323
Checking Layer Spacing (DRC)	324
Checking For Sufficient Pad Overlap (DRC)	325
System Netlist Generator Flow Example	327
Creating a New Project	328
Importing Chips into the Database	329
Modifying and Adding Pins on Chips	331
Creating Placements for the Chips	332
Defining Net Connectivity	334
Exporting the Complete Design	336
Rule File Examples	339
Calibre 3DSTACK+ Extended Syntax Example	339
Standard (Legacy) Syntax Example 1	345
Standard (Legacy) Syntax Example 2	347
Report File Format	349
Report Header	349
Summary of Drawn Layers, Placements, and Text Layers	349
Verification Results	352

Table of Contents

Appendix B	
Standard Calibre 3DSTACK Syntax Commands	357
System and Miscellaneous Commands	358
assembly_path_extension	359
export_connectivity	360
export_layout (Standard Syntax)	365
export_template	369
layer_props_file	372
report	374
run	376
set_version	378
source_filter	379
source_netlist	381
svrf_specs	386
warning_severity	387
Assembly Commands	388
anchor	390
attach_text	392
connect (Standard Syntax)	394
derived_connect	396
ignore_pin	397
ignore_trailing_chars	398
import_net_map	400
import_pin_map	401
import_text_labels	402
layer	404
layout	407
layout_case	410
layout_primary	411
map_placement	412
net_map	414
pin_map	415
pin_swap	417
place_chip	418
place_layer	421
rename_text	424
Standard Rule Syntax	425
set_auto_rve_show_layers	432
set_rve_cto_file	433
tvf_block	435
Appendix C	
Error and Warning Messages	437
Error Messages	438
Warning Messages	447
Index	
Third-Party Information	

Chapter 1

Introduction to Calibre 3DSTACK

The Calibre® 3DSTACK application enables you to verify designs containing flip chips, Through Silicon Vias (TSVs), and other 2.5D and 3D-IC configurations.

For an introduction to the Calibre 3DSTACK tool, view the following getting started video:



Overview	11
3D-IC Description Language Usage Restrictions	16
Requirements	16
Documentation Conventions	17

Overview

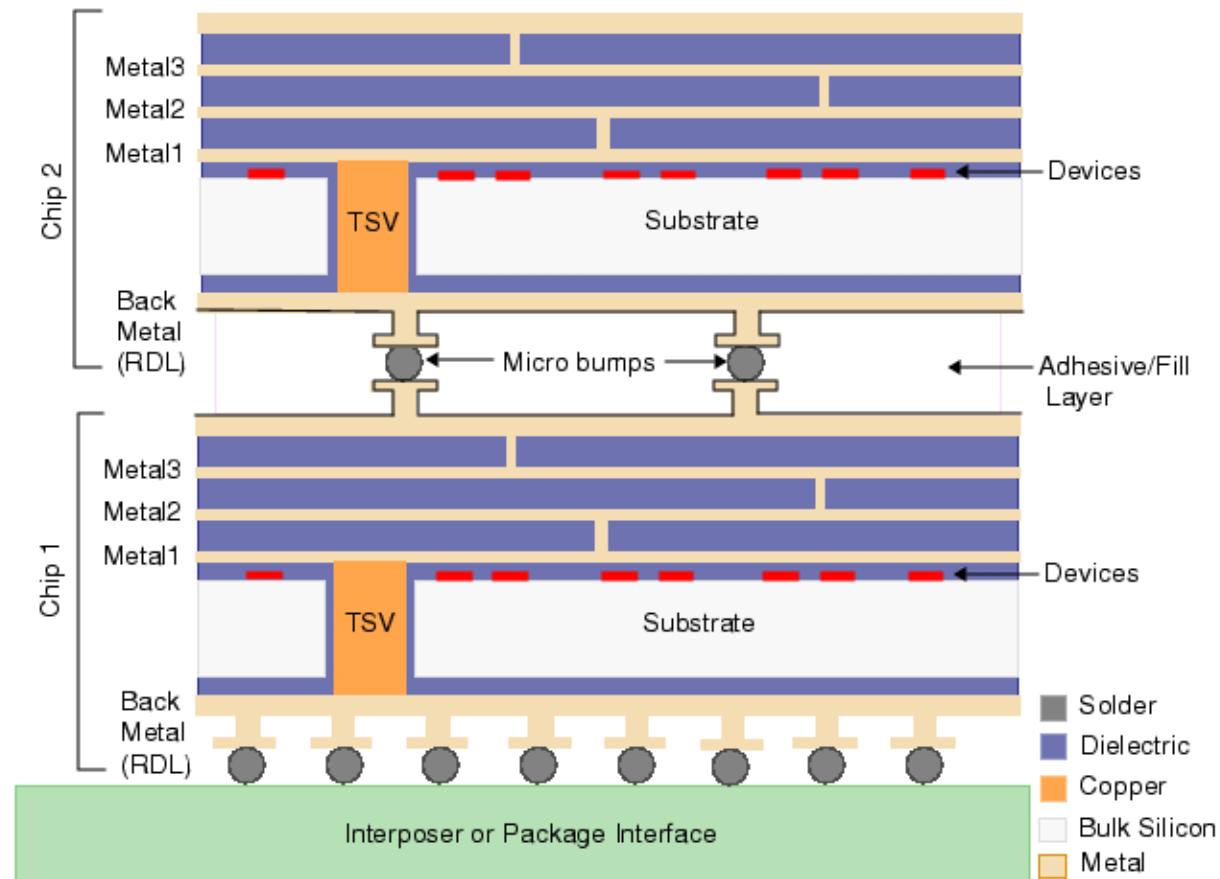
Traditional ICs are self-contained and are verified at the chip-level using Calibre nmDRC, Calibre nmLVS, Calibre RVE, and Calibre DESIGNrev. 3D-ICs consist of multiple stacked chips with connectivity achieved through traces on an interposer or with special vias or bumps. The Calibre 3DSTACK tool verifies the interfaces between these chips, which may consist of black box IP.

A cross-section of a 3D-IC is illustrated in [Figure 1-1](#). The primary difference between a 2.5D and 3D-IC configuration is the use of a silicon interposer. A silicon interposer, as illustrated in [Figure 1-2](#), electrically connects the pads between chips. These chips can either use TSVs (allowing for additional stacking), or may contain no TSVs (in which case the chips do not allow for additional stacking and are flip chips). 2.5D configurations use a silicon interposer, while 3D-IC configurations contain TSVs in active silicon that form a complete 3D stack.

A TSV is a via that passes through the substrate of a chip or silicon interposer. Typically, in a chip containing TSVs, devices and metal layers are manufactured on one side of the chip (the front), while additional metal layers are manufactured on the other side of the chip (the back). The front metal layers are normally manufactured at a more advanced process node than the

back metal layers. Small spheres of solder called micro bumps can be used to connect multiple chips together. Refer to [Figure 1-1](#).

Figure 1-1. TSV-based 3D-IC Cross-section (Conceptual Illustration)



Calibre 3DSTACK operates on these configurations by supplementing your existing Calibre verification flow. Your existing Calibre flow operates on each design independently, while Calibre 3DSTACK operates on the interfaces between designs. Designs are assembled according to instructions contained in a 3DSTACK rule file, which include offset values along the x and y axes, rotation angles, and magnification factors.

For example, in [Figure 1-2](#), two designs (chip1 and chip2) are to be assembled on a silicon interposer. To assemble the die stack, chip2 is shifted along the x-axis by x_offset and the interposer is rotated clockwise by 90 degrees. [Figure 1-2](#) summarizes the necessary inputs and outputs to a 3DSTACK assembly operation.

Figure 1-2. 3DSTACK Assembly (Conceptual Illustration)

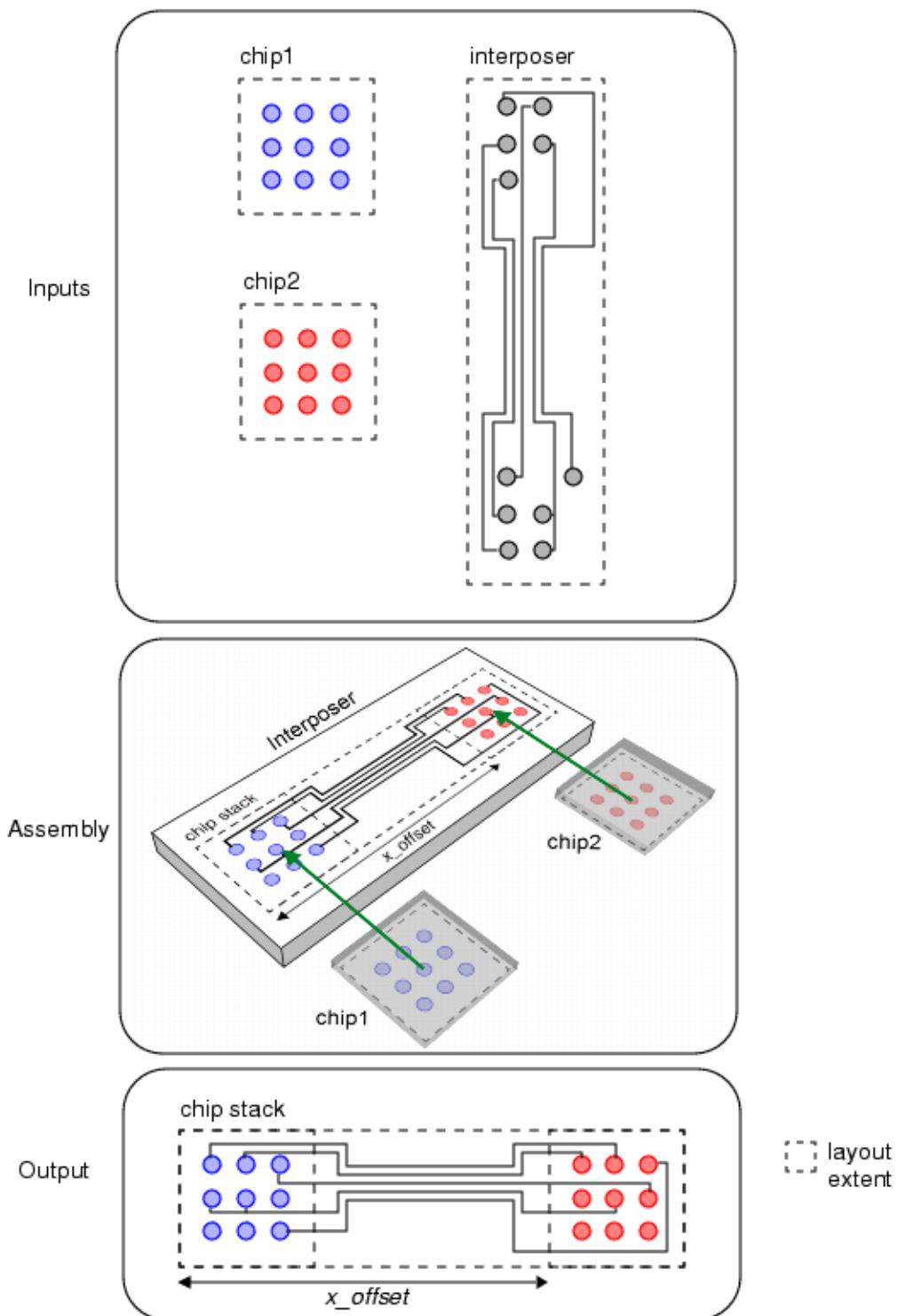
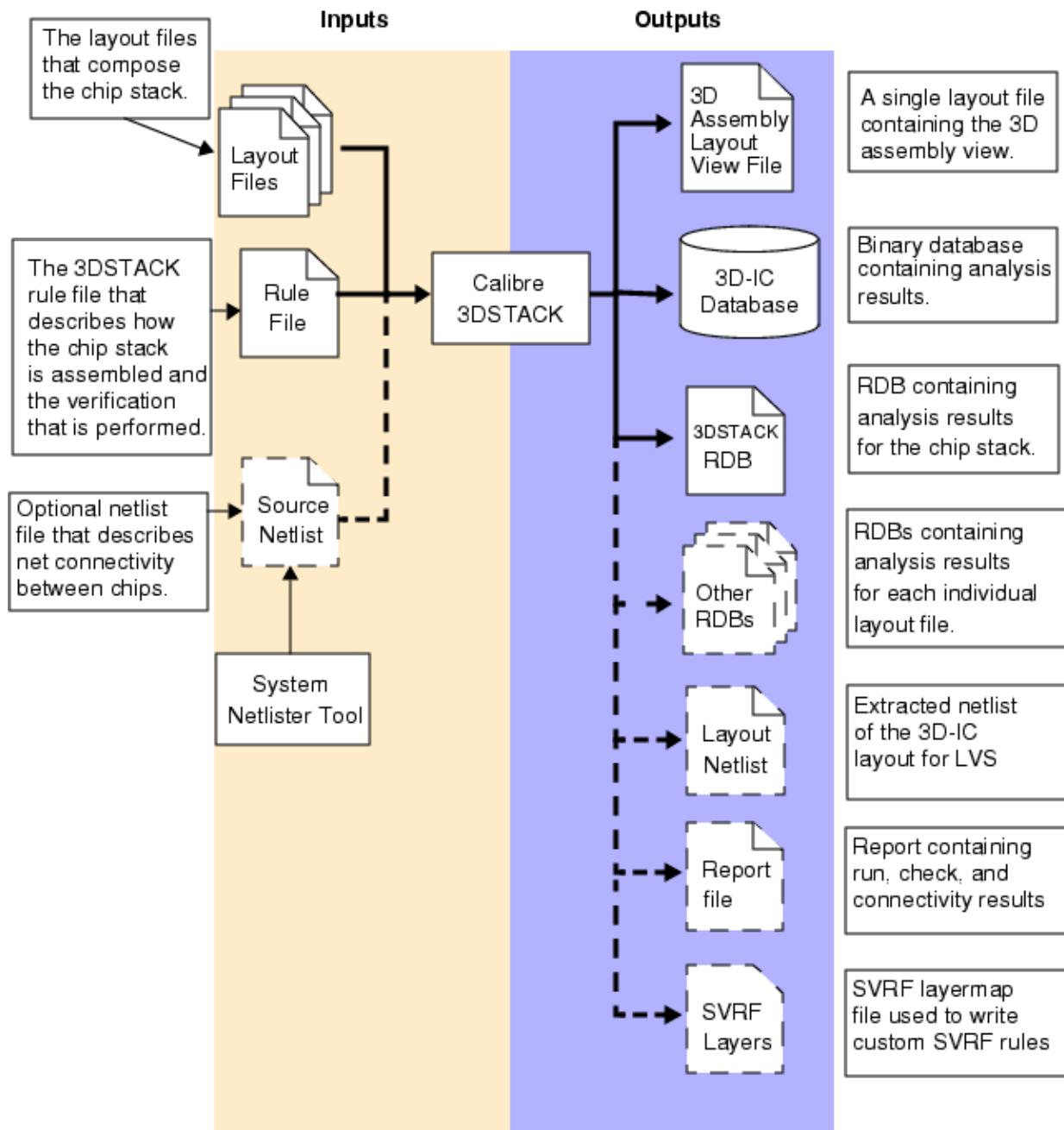


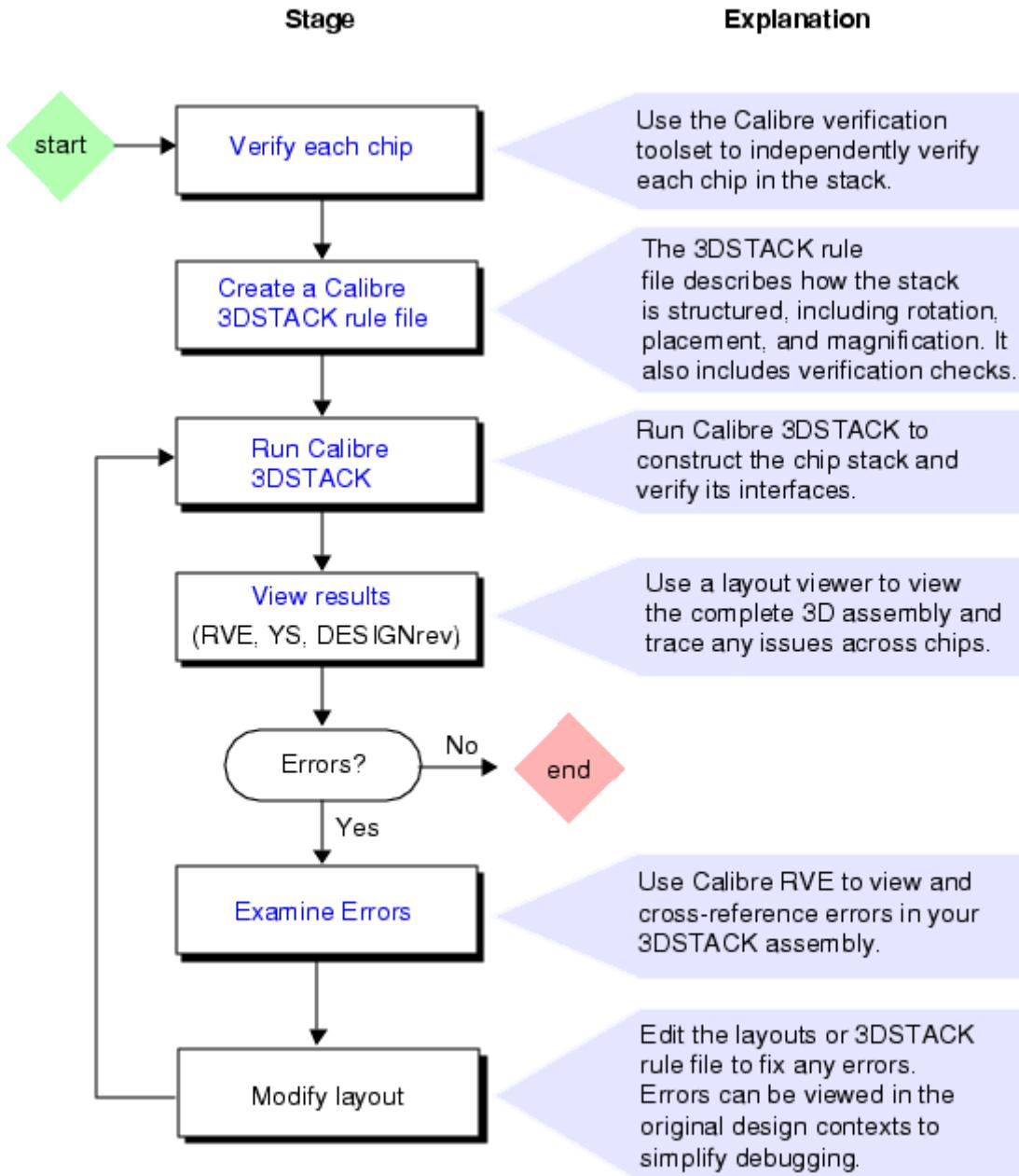
Figure 1-3 illustrates the input files that Calibre 3DSTACK accepts and the output files that it creates. Dimensional check (DRC) and connectivity operations (LVS) can be performed on the assembled chip stack using commands specified in the 3DSTACK rule file.

Figure 1-3. Inputs and Outputs



The following figure illustrates the typical workflow for verifying a design with Calibre 3DSTACK.

Figure 1-4. Calibre 3DSTACK Flow



Related Topics

[Requirements](#)

[Calibre 3DSTACK Invocation](#)

3D-IC Description Language Usage Restrictions

You must adhere to specific requirements with respect to the usage of the rule file language used in the Calibre 3DSTACK tool.

This document includes code from a 3D-IC Description Language (3DIC_DL) used within/by Mentor Graphics' products supporting 2.5D/3D IC applications. You shall not use this 3DIC_DL unless you are a Mentor Graphics customer. The exact terms of your obligations and rights are governed by your respective license.

You shall not use this 3DIC_DL except:

- (a) for your internal business purposes and
- (b) for use with Mentor Graphics' Calibre tools.

The 3DIC_DL may constitute or contain trade secrets and confidential information of Mentor Graphics or its licensors. You shall not make the 3DIC_DL available in any form to any person other than your employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality.

Requirements

Before running the Calibre 3DSTACK tool, you must have a full Calibre installation and the appropriate licenses. The chips in your stack must already be DRC and LVS clean.

You must have the following Calibre licenses:

- Calibre nmDRC-H and Calibre nmDRC.
- Calibre nmLVS-H and Calibre nmLVS.
- Calibre DESIGNrev.
- Calibre RVE (highly recommended for debugging results)

Note

 If you specify the -turbo invocation option, the number of Calibre nmDRC-H/nmDRC and nmLVS-H/nmLVS licenses are scaled according to the Standard Multithreaded License Consumption Formula used for all the Calibre tools. See “[Licensing for Multithreaded Operations](#)” in the *Calibre Administrator’s Guide* for details.

Calibre 3DSTACK also automatically reserves all available Calibre DESIGNrev licenses available at the time of invocation.

You must set the CALIBRE_HOME or MGC_HOME environment variable to point to the location of your Mentor Graphics software tree. See “[Setting the CALIBRE_HOME Environment Variable](#)” in the *Calibre Administrator’s Guide* for details.

The following inputs must be defined:

- The layout files that compose the 3D assembly. The layout must be in GDS, OASIS®¹, or LEF/DEF format. Cell and pin names should not have spaces, as this causes the exported netlist to be incorrect.
- A valid Calibre 3DSTACK rule file containing only commands specified in the [Command Reference](#), plus any standard Tcl constructs.
- A source netlist for the chip stack is highly recommended.

Related Topics

[Calibre 3DSTACK Invocation](#)

Documentation Conventions

The command descriptions for Calibre 3DSTACK statements use font properties and several metacharacters to document the command syntax.

The commands described in this document and any examples follow the syntax conventions outlined in the following table.

Table 1-1. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	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.

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.

Table 1-1. Syntax Conventions (cont.)

Convention	Description
{ }	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>' ...]

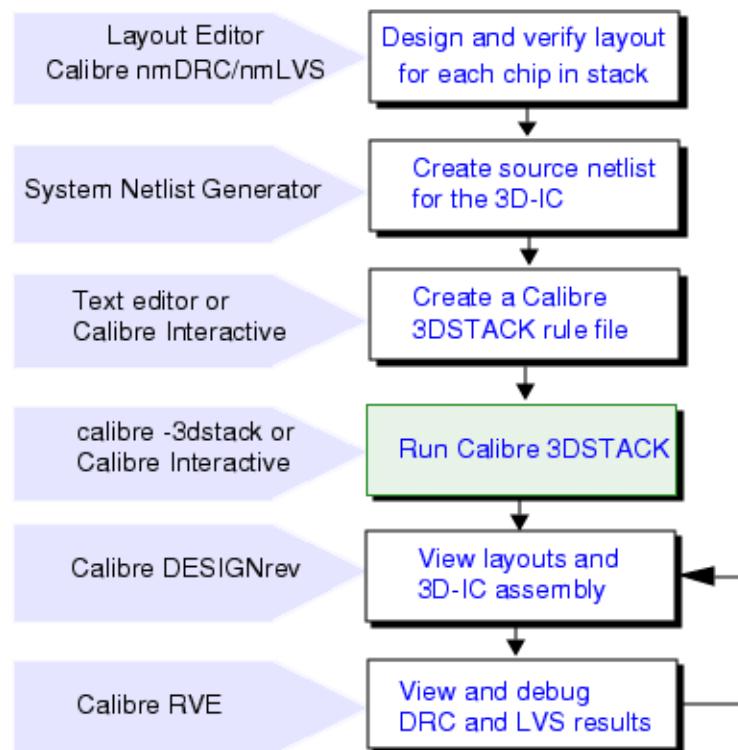
[('swap_list') ...]

[BY NET | BY SHAPE]
```

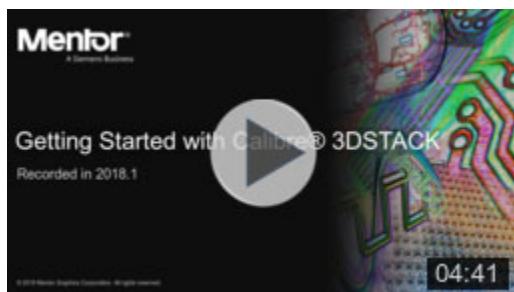
Chapter 2

Getting Started With Calibre 3DSTACK

The Calibre 3DSTACK tool uses the powerful and familiar set of verification and analysis features included with the Calibre nmDRC, Calibre nmLVS, Calibre RVE, and Calibre DESIGNrev products. Calibre 3DSTACK uses its own rule file format and commands, but the principles of operation are familiar to existing Calibre verification flows.



For a quick introduction, view the following getting started video:



Calibre 3DSTACK Invocation	20
Using Calibre 3DSTACK	25

Calibre 3DSTACK Invocation

Invoke Calibre 3DSTACK in batch mode from the command line or from Calibre Interactive. The majority of rule file operations can be modified from Calibre Interactive 3DSTACK. After a run is complete, use Calibre RVE and Calibre DESIGNrev to view the results.

calibre -3dstack	21
Invoking Calibre Interactive for Calibre 3DSTACK.....	23

calibre -3dstack

Command-line invocation arguments control the mode and operations performed by Calibre 3DSTACK.

Note

 The -cs command-line option and the “check source” checkbox in Calibre Interactive have been removed from the documentation and the Calibre Interactive GUI. The options continue to work in the 2020.4 release if you specify them in a sunset file or batch script. The options will be no longer function starting with the 2021.1 release.

Usage

```
calibre -3dstack [-help]
  [-turbo [number_of_processors]]
  { [-create_assembly assembly_name [-system {OASIS | GDS}]]
    | [-use_assembly assembly_path]
    | [-assembly assembly_name] }
  [-run_dir directory]
  [-compile_only]
  rule_file_name
```

Arguments

- **-3dstack**
Required argument that enables Calibre 3DSTACK.
- **-help**
Optional argument that displays the command line usage.
- **-turbo [number_of_processors]**
Optional argument that enables multi-threaded parallel processing. You can optionally specify the number of processors to use (the default is all).

Note

 If you specify the -turbo invocation option, the number of Calibre nmDRC-H/nmDRC and nmLVS-H/nmLVS licenses are scaled according to the Standard Multithreaded License Consumption Formula used for all the Calibre tools. See “[Licensing for Multithreaded Operations](#)” in the *Calibre Administrator’s Guide* for details.

Calibre 3DSTACK also automatically acquires all available Calibre DESIGNrev licenses available at the time of invocation.

- **-create_assembly assembly_name**

Optional argument that limits the run to creating the 3D assembly. Cannot be specified with -use_assembly or -assembly.

- **-system {OASIS | GDS}**
Optional argument that specifies the layout system. The default is OASIS. This argument cannot be specified with **-use_assembly**.
- **-use_assembly *assembly_name***
Optional argument that instructs the tool to use the specified *assembly_name* for the run. All specified verification checks are executed on the supplied *assembly_path* file. Cannot be specified with **-create_assembly**, **-system**, or **-assembly**.
- **-assembly *assembly_name***
Optional argument set that specifies a custom name for the generated Calibre 3DSTACK assembly layout. This option is mutually exclusive with the **-create_assembly** or **-use_assembly** options. If you do not specify this option, the default name is **3dstack_assembly**.
- **-run_dir *directory***
Optional argument that specifies a path to a *directory* to place output files. The default is the current directory.
- **-compile_only**
Optional argument that specifies to only compile the extended Calibre 3DSTACK+ rule file to the internal standard syntax. The compiled rule file is written to your working directory with the name *<name>.3dstack*. The standard syntax file is useful for debugging your specified operations.
- ***rule_file_name***
Required argument that specifies the path to the 3DSTACK rule file.

Description

When a Calibre 3DSTACK run is executed, specified layout files are assembled according to the instructions present in your Calibre 3DSTACK rule file. Calibre 3DSTACK then performs specified verification checks and generates 3DSTACK results databases, an extracted layout netlist, and a report file, if specified by your rule file.

Additional information about the *command options* for Calibre Interactive can be found under the topic “[Invoking Calibre Interactive from the Command Line](#)” in the *Calibre Interactive User’s Manual*.

Note

 The **-use_assembly** argument is mutually exclusive with the **-create_assembly**, **-assembly**, and **system** arguments. When **-create_assembly** is specified, only the assembly and **export_template** commands are executed (no rule checks are done). When **-use_assembly** is included in the invocation, no assembly and **export_template** commands are performed; all the specified checks are executed on the specified assembly file.

Examples

Invoke Calibre 3DSTACK in batch mode:

```
calibre -3dstack rules.3dstack
```

Invoke Calibre Interactive for Calibre 3DSTACK with the following command:

```
calibre -gui -3dstack [-runset runset] [gui_options]
```

Invoke Calibre Interactive batch mode for Calibre 3DSTACK with the following command:

```
calibre -gui -3dstack -runset runset [gui_options] -batch
```

Generate a compiled version of a Calibre 3DSTACK+ rule file in your working directory without performing any other operations.

```
calibre -3dstack -compile_only extended_rules.3dstack
```

Related Topics

[System Netlist Generator Flow and Invocation](#)

[Invoking Calibre Interactive for Calibre 3DSTACK](#)

Invoking Calibre Interactive for Calibre 3DSTACK

Invoke the Calibre 3DSTACK GUI from the command line or from Calibre DESIGNrev.

Procedure

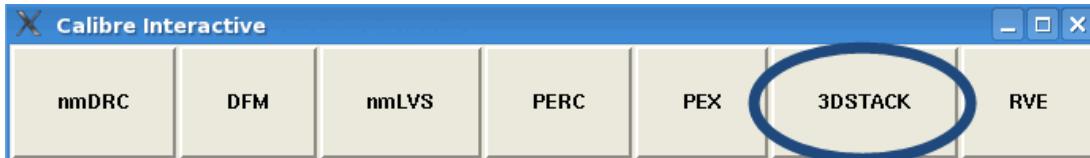
1. Start Calibre Interactive for Calibre 3DSTACK using any of the following methods:

- Enter the following command to directly invoke Calibre Interactive 3DSTACK:

```
calibre -gui -3dstack
```

- Enter the following command to invoke the Calibre Interactive GUI launcher and click the **3DSTACK** button:

```
calibre -gui
```



- Choose **Verification > Run 3DSTACK** from Calibre DESIGNrev.

2. When prompted, choose whether to specify an optional runset file.

A runset file sets GUI options and can be useful for managing different types of verification operations.

Using Calibre 3DSTACK

In order to run Calibre 3DSTACK, you must write a rule file that contains assembly and verification commands.

To perform a verification or to generate an assembly view, you invoke Calibre 3DSTACK with the rule file as an argument. You can view the physical assembly layout view in Calibre DESIGNrev and highlight rule check results from Calibre RVE.

This section contains procedures that explain the recommended flow for setting up and running Calibre 3DSTACK.

Creating a Calibre 3DSTACK Rule File	25
Running a Calibre 3DSTACK Verification.....	28
Calibre 3DSTACK Output	34
Assembly Views	35
Calibre 3DSTACK Results Analysis Examples	38
Calibre Interactive for Calibre 3DSTACK.....	54

Creating a Calibre 3DSTACK Rule File

Calibre 3DSTACK supports two rule file syntaxes: standard and extended (3DSTACK+). The extended syntax offers a more general approach to the assembly and verification of your stacked IC and is recommended for all rule files.

Verification commands are similar between the conventional and extended syntax, but the rule checks in the conventional rule file only work for a single assembly. This is because the conventional syntax relies on placement layer names defined by assembly commands. The extended syntax allows you to write rules that can be applied to different assemblies based on the functionality of a layer in the design.

The extended syntax rule file is compiled into a standard syntax file in your working directory during a run. You can use the standard syntax file for debugging.

This procedure steps you through the creation of an extended syntax rule file.

Note

 The rule file must only consist of commands defined in the [Command Reference](#) and supported Tcl constructs. Rule checks cannot share names with Calibre 3DSTACK file format keywords. See the “[Examples](#)” chapter to view sample rule files using the 3D-IC Description Language.

Procedure

1. Create a new text file using an editor of your choice.

2. Add the following statements to the first lines of the file:

```
#!3dstack+
set_version -version 1.0
```

3. Specify the [config](#) command to define optional run settings for Calibre 3DSTACK:

```
config \
    -layout_primary TOP_CELL \
    -source_netlist {-file top_cell.spi -format SPICE -case YES } \
    -report {-file 3dstack.report} \
    -export_connectivity {-file 3dstack.v -format verilog }
```

This command configures the following options:

- The name of the top cell for the generated assembly layout view.
 - The netlist that defines the ideal connectivity of all of the chips in the stack (recommended).
 - Generates a report that provides detailed summaries of all verification and assembly operations (recommended).
 - Writes out the assembly netlist in Verilog format.
4. Define all the dies (and their respective interacting layers) using the [die](#) command. The die command specifies layout files (which must already be LVS and DRC clean) and layers within the files that interact in the stack. You can read in the entire connectivity with the -wb_connect option.

```
die -die_name chip1 \
    -layout { \
        -path ./my_chip.gds \
        -type gdsii \
        -primary top_cell \
        -depth top-only \
    } \
    -layer_info { \
        -type pad \
        -name landing_pads \
        -ext_connect \
        -layer { \
            255 \
            -depth top-only \
        } \
        -text { 255 } \
        -bottom \
    }
...
...
```

In this example, only the top-level layers are brought into Calibre 3DSTACK. The -layer_info argument set defines all layers used in the assembly of the stack.

5. Define how the dies are physically placed and stacked using the [stack](#) command. This operation creates an assembly layout view that represents your 3D-IC die stacking configuration.

```
stack -stack_name assembly \
-die { \
    -name interposer \
    -source pInterp \
    -placement 0 0 \
    -invert \
} \
-z_origin 0 \
-tier { \
    -die {-name chip1 -placement 14 12 -source pChip1}\ \
    -die {-name chip2_a -placement 80 12 -source pChip2}\ \
    -die {-name chip2_b -placement 80 45 -source pChip3}\ \
}
...
...
```

The `-placement` option defines the x and y position of the die. Tiers are collections of dies that occupy the same vertical starting position (z-coordinate). The `-source` option allows you to map the assembly chip name to the instance name in the source netlist.

Tip It is helpful to preview your physical model in Calibre DESIGNrev before writing any rules. You can instruct Calibre 3DSTACK to only generate the assembly layout view from your rule file using the `-create_assembly` option as follows:

```
calibre -3dstack -create_assembly assembly_name 3dstack.rules
```

6. Write connectivity checks for your assembly using the [connected](#) command:

```
connected -check_name CONNECT_PAD_TO_BUMP \
-layer_type1 pad \
-layer_type2 bump \
-black_box \
-comment "CONNECT::IO Connectivity Check"
```

7. Apply other verification checks as required. See “[Rule Check Commands](#)” on page 139 for details.

Related Topics

[Command Reference](#)

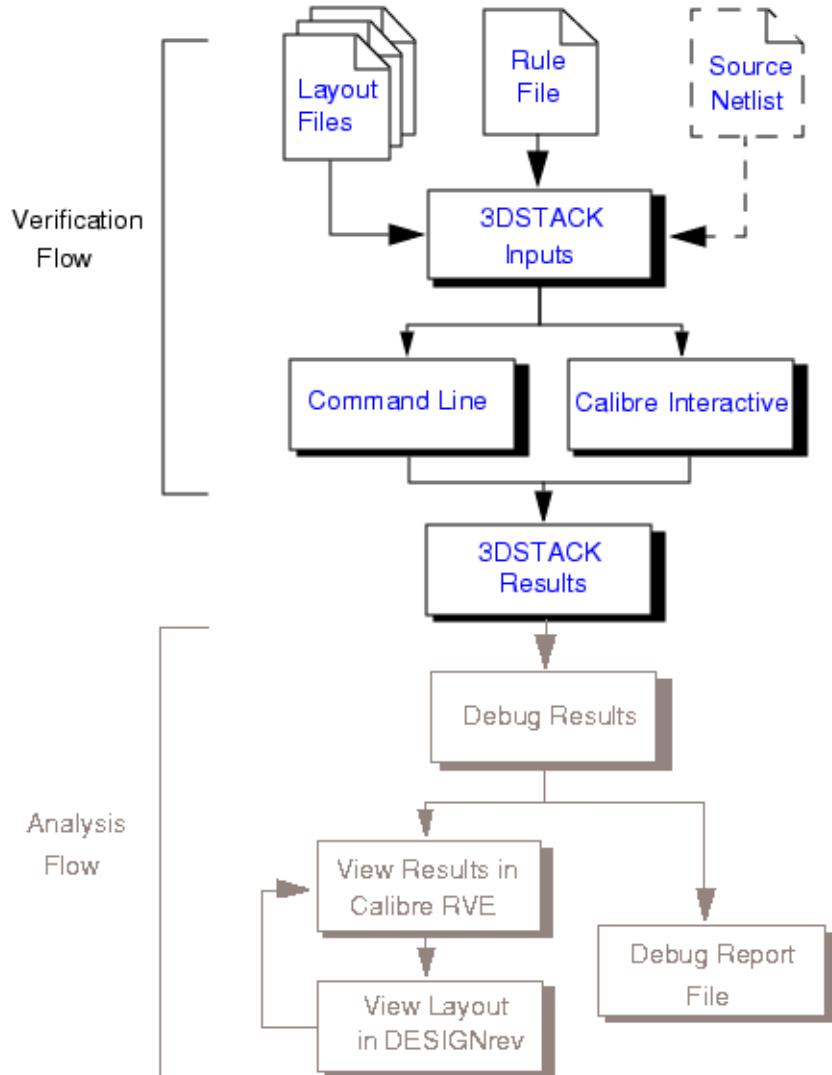
[Examples](#)

Running a Calibre 3DSTACK Verification

You can perform a Calibre 3DSTACK verification run from the command line, Calibre DESIGNrev, or Calibre Interactive.

The flow for running a Calibre 3DSTACK verification is highlighted in [Figure 2-1](#) below.

Figure 2-1. Calibre 3DSTACK Verification Flow



Running a Calibre 3DSTACK Verification from the Command Line.....	29
Running a Calibre 3DSTACK Verification from Calibre Interactive	30
Running Calibre Interactive 3DSTACK From Calibre DESIGNrev.....	32

Running a Calibre 3DSTACK Verification from the Command Line

You must have a complete rules file before running Calibre 3DSTACK from the command line.

Prerequisites

- You have met the “[Requirements](#).”
- You have a valid 3DSTACK rule file. Refer to “[Creating a Calibre 3DSTACK Rule File](#)” on page 25 for more details on writing Calibre 3DSTACK rule files.

Procedure

1. Modify your Calibre 3DSTACK rule file so that it includes the `config -report` command option. The report provides essential details about a Calibre 3DSTACK run that can aid in debugging and design validation.
2. From the command line, run Calibre 3DSTACK with the following command:

```
calibre -3dstack rule_file
```

where `rule_file` is the path to your 3DSTACK rule file. Sample rule files are provided in the “[Examples](#)” section of Appendix A.

By default, Calibre 3DSTACK writes all associated verification output files to your working directory. The outputs are described in detail under “[Calibre 3DSTACK Output](#)” on page 34.

3. During the verification run, observe the transcript output to view the assembly and rule checking process. Investigate and correct any warnings or error messages as necessary.
4. When the verification run completes, open the report file, `3dstack_report.rpt`, in your output directory in a text editor and become familiar with the different sections. Understanding the structure of the assembly and verification process greatly aids in debugging complex designs. Information on the report file can be found under “[Report File Format](#)” on page 349 of Appendix C.

Results

Your output directory now contains the assembled layout view, `3dstack_assembly.oas`, and results database files that can be opened in Calibre RVE. If specified, your run directory also contains a report file used to debug the verification run. To understand how to view and analyze the verification results produced by a Calibre 3DSTACK run, proceed to “[Calibre 3DSTACK Results Analysis Examples](#)” on page 38.

Related Topics

[Calibre 3DSTACK Invocation](#)

[Calibre 3DSTACK Results Analysis Examples](#)

Running a Calibre 3DSTACK Verification from Calibre Interactive

You can set up and run a Calibre 3DSTACK verification from Calibre Interactive.

Detailed procedures for Calibre Interactive 3DSTACK are found under the section “[Calibre Interactive for Calibre 3DSTACK](#)” on page 54. Additional information about Calibre Interactive can be found in the [Calibre Interactive User’s Manual](#).

Note

 In the case of a conflict between commands in the rule file and Calibre Interactive, the settings in Calibre Interactive take precedence.

Prerequisites

- You have met the “[Requirements](#).”
- You have a valid Calibre 3DSTACK rule file. Refer to “[Creating a Calibre 3DSTACK Rule File](#)” on page 25 for more details.

Procedure

1. Enter the following command to invoke Calibre Interactive for Calibre 3DSTACK:

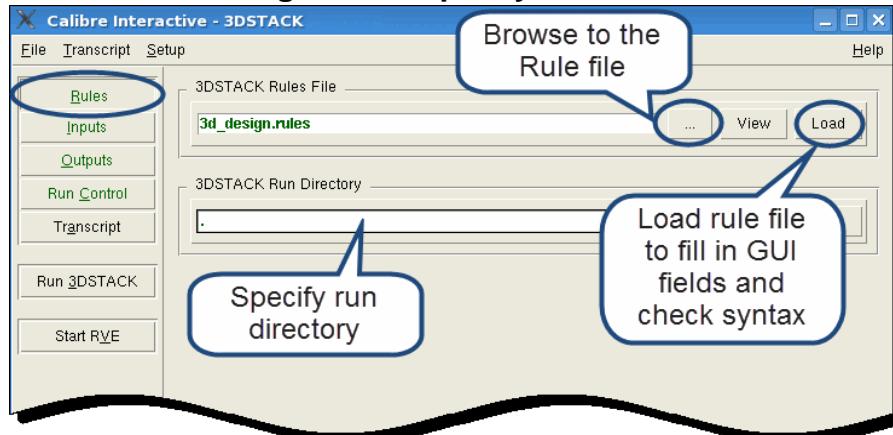
```
calibre -gui -3dstack
```

2. After you invoke Calibre Interactive, you are prompted to specify a runset file.

A runset sets GUI options and can be useful for managing different types of verification operations. You can continue without a runset by clicking **Cancel**, or load an existing runset with the **Browse (...)** button.

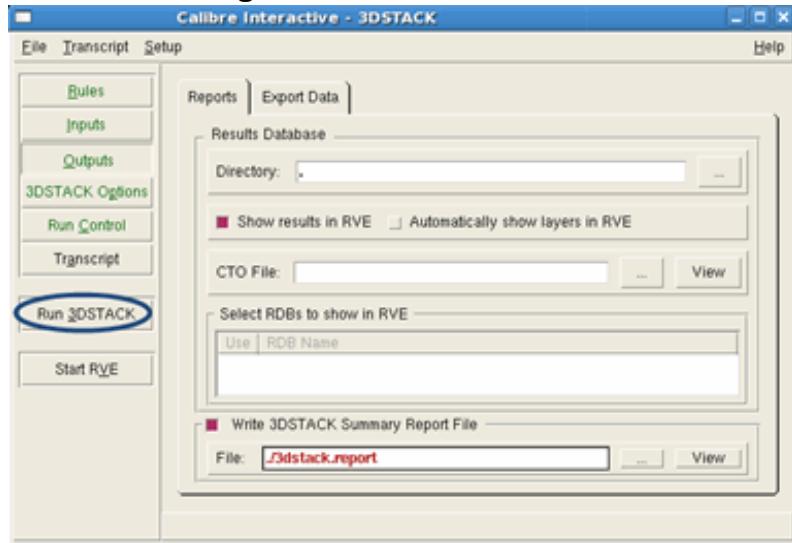
3. Click on the **Rules** button in the upper-left pane of the CI window and specify the path to your Calibre 3DSTACK rule file and run directory.

Figure 2-2. Specify Rule File



4. (Optional) Click the **Load** button to load instructions from the rule file and apply them to the current Calibre Interactive session. Any syntax errors are reported.
5. Click on the **Outputs** button and enable the “Show results in RVE” checkbox under the **Reports** tab to display results in Calibre RVE following a Calibre 3DSTACK run.
6. Enable the “Write 3DSTACK Summary Report File” checkbox and specify a path to a file. The report file contains important information about the design and verification results. Enabling this option is highly recommended.
7. Ensure that the input and output file options are correct and then click **Run 3DSTACK**.

Figure 2-3. Run 3DSTACK



Results

Your output directory now contains the assembled 3DSTACK layout view (if specified) and results databases that can be opened in Calibre RVE. Your run directory also contains a report file used to debug the verification results.

To understand how to view and analyze the verification results produced by a Calibre 3DSTACK run, proceed to “[Calibre 3DSTACK Results Analysis Examples](#)” on page 38.

Related Topics

[Calibre 3DSTACK Invocation](#)

[Calibre 3DSTACK Results Analysis Examples](#)

Running Calibre Interactive 3DSTACK From Calibre DESIGNrev

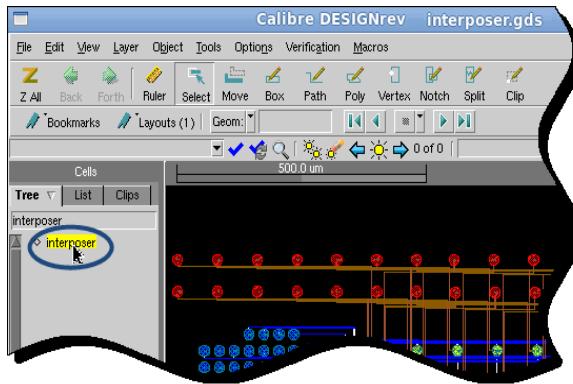
Calibre DESIGNrev includes a menu option to export your layout view from an active session to Calibre Interactive 3DSTACK. You can then perform a verification run directly from Calibre Interactive.

Prerequisites

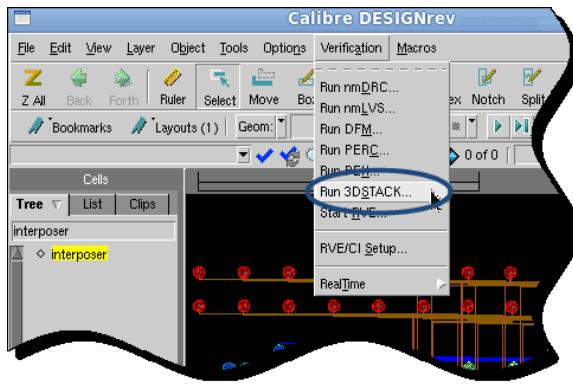
- You have met the “[Requirements](#).”
- You have a valid Calibre 3DSTACK rule file.
- You have a layout open in Calibre DESIGNrev that is specified in your rule file.

Procedure

1. Select the top cell of your layout in the cells list on the left pane of Calibre DESIGNrev, as shown in the following figure:



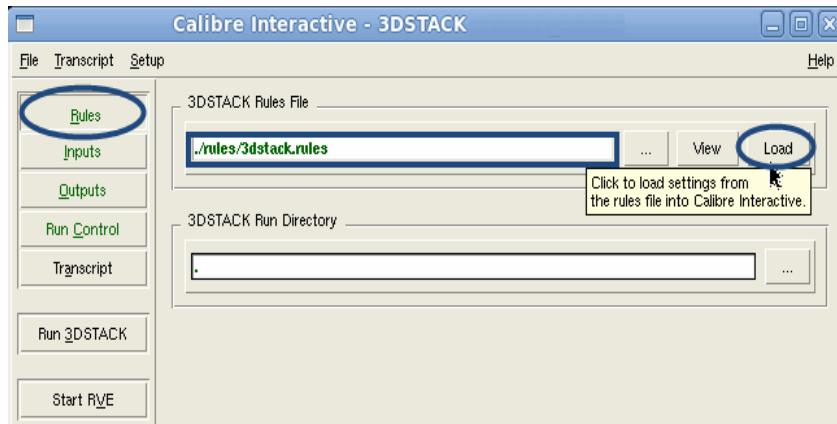
2. Select **Verification > Run 3DSTACK**. You do not have to save your layout file. The current layout view is exported to Calibre Interactive, including unsaved changes.



3. After you invoke Calibre Interactive, you are prompted to specify a runset file.

You can continue without a sunset by clicking **Cancel**, or load an existing sunset with the **Browse (...)** button.

- Click the **Rules** button, specify the path to your Calibre 3DSTACK rule file, and click **Load**. Loading a rule file updates the fields in the Calibre Interactive GUI with the settings from your rule file.



- Click the **Inputs** button, enable the “Export from layout viewer” checkbox, and specify the layout view in Calibre DESIGNrev using the **Chip Name** dropdown box.



The above snapshot of Calibre Interactive shows that the *interposer* top cell was exported from Calibre DESIGNrev. The Layout Path field of the Layouts table lists the path as *interposer.calibre.db*.

- Click **Run 3DSTACK** to start a verification run from Calibre Interactive.

Results

The layout view of one of the specified chips in your 3DSTACK assembly was exported from Calibre DESIGNrev and used in a Calibre 3DSTACK verification run. This process allows you to modify layouts in Calibre DESIGNrev and view changes in the verification results without saving the layout.

Related Topics

[Calibre 3DSTACK Invocation](#)

[Calibre 3DSTACK Results Analysis Examples](#)

Calibre 3DSTACK Output

All output files are saved to your working directory during a Calibre 3DSTACK run. The transcript is updated in real-time.

Output Files

The following output files are created in your working directory during a run:

- ***3dstack.log*** — Complete transcript from a run.
- ***3dstack.dfmdb* (directory)** — Binary database containing analysis results. The DFM database enables advanced cross-referencing in Calibre RVE between the error results, the source schematic, and the design layout. See “[Opening a Calibre 3DSTACK DFM Database in Calibre RVE](#)” on page 41 for more information.
- ***3dstack.rdb*** — RDB file containing analysis results. An RDB is an ASCII-formatted results database generated from a verification run. The RDB file can be directly opened in Calibre RVE.

Note



If you do not specify the `config -report` command in your rule file, then there is no limit to the number of results written to the results database.

- ***<name>.oas*** — Assembled chip stack view file. The *name* is specified by the `-create_assembly` or `-assembly` invocation argument. The default is `3dstack_assembly`.
- ***<name>.oas.layerprops*** — Layer properties file for the chip stack. This file defines the layer properties for a set of design layers, polygon connectivity, and via cell information.
- ***<name>.oas.layermap*** — SVRF layer definitions generated from the assembly process that define layers and connectivity information. This file can be used to write custom SVRF rules for verification runs on the assembly.
- ***<name>_cross_section.oas*** — Optional view of your layout in the x, y, and z dimensions. See “[Assembly Views](#)” on page 35 for details. The *name* is either “`3dstack`” or the name specified by the `-create_assembly` or `-assembly` invocation argument.
- ***<name>_cross_section.oas.layerprops*** — Layer properties for the cross-sectional assembly view. The *name* is either “`3dstack`” or the name specified by the `-create_assembly` or `-assembly` invocation argument.

- ***3dstack_deck.svrf.enc*** — Intermediate encoded SVRF file generated by Calibre 3DSTACK at runtime. Use this file to debug tvf_block syntax errors.
- ***3dstack_overlay_generator.tcl*** — Tcl script used to generate the chip stack view from the individual layouts.
- ***<3dstack+_rule_file>.3dstack*** — Compiled 3DSTACK+ rule file.
- ***3dstack.warnings*** — List of all warnings produced during a run.
- ***readerprefs*** — Settings for the Layout Input Exception Severity statements that are read from the config -svrf_spec svrf_file are saved to this file in your run directory. To configure Calibre DESIGNrev to read this file, set the MGC_CWB_CONFIG_DIRS environment variable to your run directory.
- **Layout files in GDS or OASIS** — Optional layout files from derived layer or export operations.

In addition to these files, RDB files are optionally created for each layout file in the chip stack. If the config -report command is specified (recommended), a report is saved in your working directory.

Run Time Status Notifications

The transcript output updates continuously during the 3DSTACK run with the following information:

- Chips currently being processed.
- Cell and placements under creation.
- Calibre 3DSTACK assembly file path and physical size.
- Run duration.
- Summary statistics for each stage of the run.

Related Topics

[Command Reference](#)

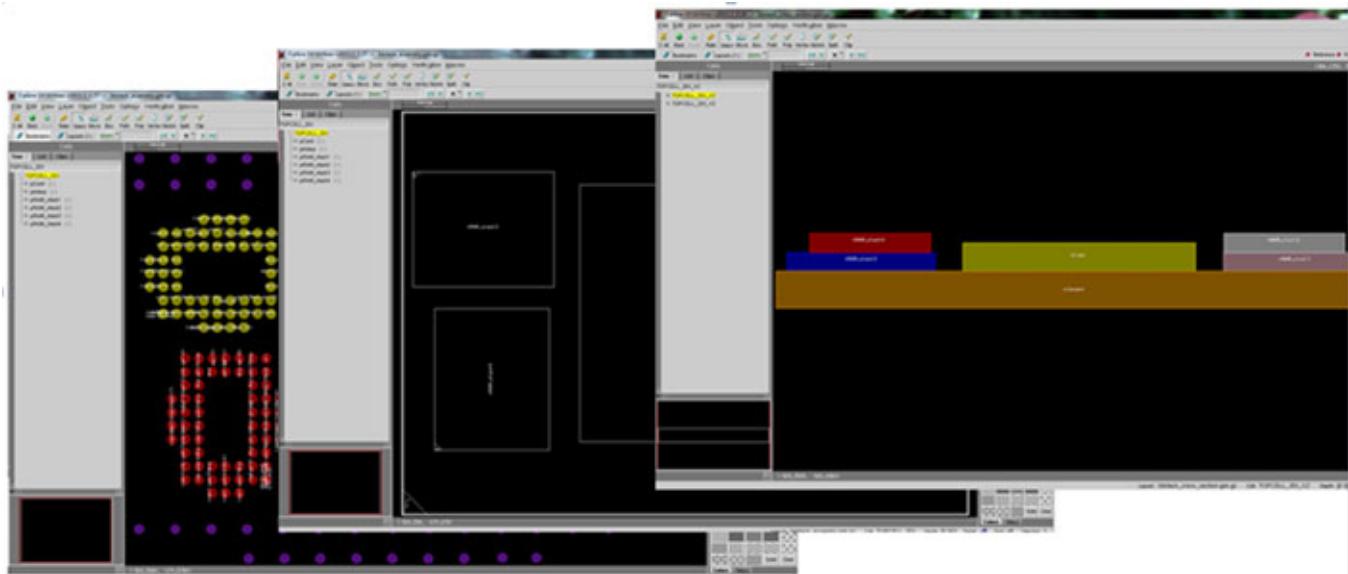
[Report File Format](#)

Assembly Views

Calibre 3DSTACK generates an overhead assembly view of your stacked package when you perform a verification run or when you invoke the tool with the -create_assembly option. You can also create an optional assembly view that displays the package in the x, y, and z dimensions.

Calibre 3DSTACK saves the overhead view of your assembly in your working directory as *3dstack_assembly.oas*. Open this file in Calibre DESIGNrev to see the layout of your stack.

You can optionally generate a cross-sectional x, y, and z view of your assembly. The tool generates the *<name>_cross_section.oas* assembly view file in your working directory if you specify thicknesses and z-origins for your dies (where *name* is “3dstack” or the name specified by the -create_assembly invocation argument).



The cross-section view includes die bumps. Die bumps are drawn as a thin rectangular object on the interface layer of the die.

The following arguments enable the generation of the cross-section view:

- Apply the -thickness option of the `die` command. This option defines the thickness of the die in microns. If this is not specified, you can use the `CALIBRE_3DSTACK_DEFAULT_THICKNESS` environment variable to set a global thickness for all dies.
- Apply the -z_origin option in the `stack` command. The -z_origin option specifies the height of the object from the bottom of the stack.

The following is an example:

```
die -die_name interposer \
    -layout { \
        -path ./design/interposer.gds \
        -type gdsii \
        -primary interposer \
    } \
    -thickness 20 \
    ...
```

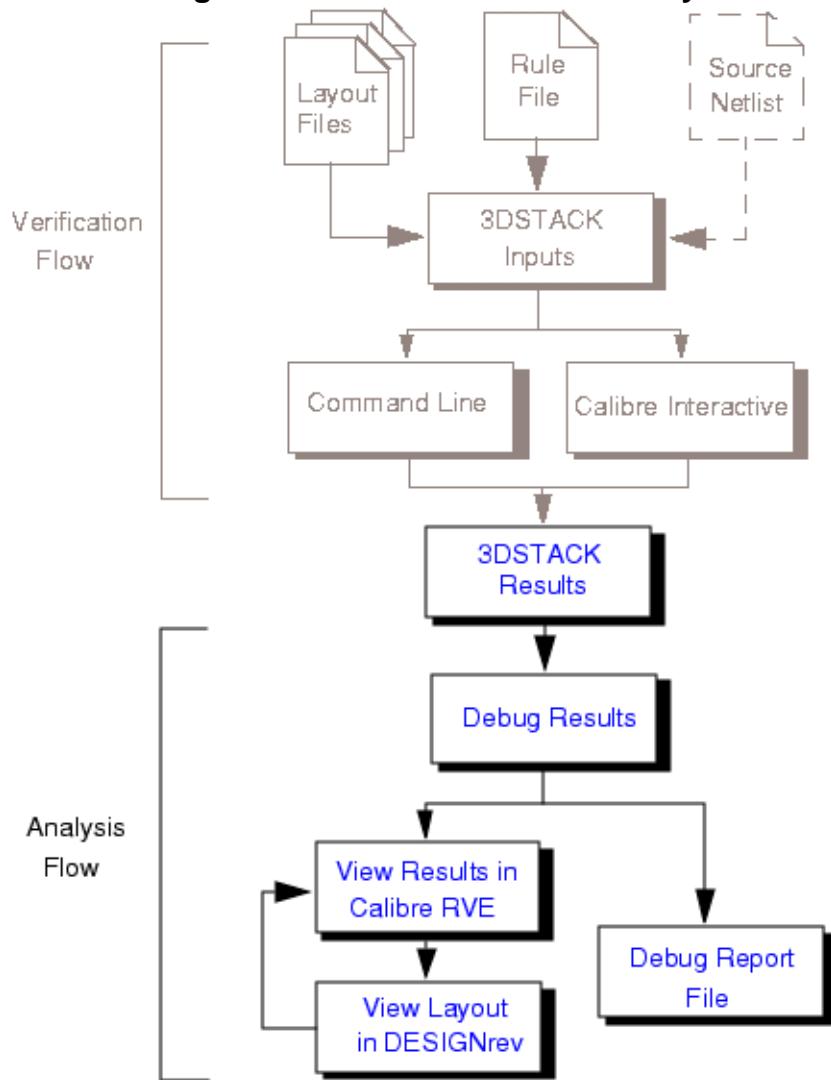
```
die -die_name ram \
    -layout { \
        -path ./design/ram.gds \
        -type gdsii \
        -primary ram \
    } \
    -thickness 11 \
    ...
...
die -die_name controller \
    -layout { \
        -path ./design/controller.gds \
        -type gdsii \
        -primary controller \
    } \
    -thickness 13 \
    ...
...
stack -stack_name assembly \
    -die { \
        -name interposer \
        -placement 0 0 \
        -invert \
    } \
    -tier { \
        -die {-name controller -placement 14 12} \
        -die {-name ram -placement 80 12} \
        -die {-name ram -placement 80 45} \
    } -z_origin 20
```

Calibre 3DSTACK Results Analysis Examples

You can view and analyze verification results produced by Calibre 3DSTACK using a combination of Calibre RVE and Calibre DESIGNrev. Reports in text format are also available after a run.

The typical Calibre 3DSTACK results analysis flow is highlighted in the following figure:

Figure 2-4. Calibre 3DSTACK Analysis Flow



Highlighting DRC Results from a Calibre 3DSTACK Run	39
Opening a Calibre 3DSTACK DFM Database in Calibre RVE	41
Debugging Connectivity Errors in Calibre 3DSTACK	44
Using Path Isolation to Debug Shorts	49
Viewing Design Elements in a Schematic View.....	52
Verification Results Format	52

Highlighting DRC Results from a Calibre 3DSTACK Run

Use Calibre RVE to highlight DRC results from a Calibre 3DSTACK run. DRC results are produced by the geometrical rule checks in the 3DSTACK rule file.

For advanced connectivity debugging using Calibre 3DSTACK DFM databases, follow the procedure described in “[Debugging Connectivity Errors in Calibre 3DSTACK](#)” on page 44.

Prerequisites

- Calibre 3DSTACK results databases generated by a verification run. See “[Running a Calibre 3DSTACK Verification](#)” on page 28.
- Layout files that compose the assembly.

Procedure

1. Load the Calibre 3DSTACK assembly in a supported layout editor and launch Calibre RVE with the Calibre 3DSTACK RDB file. For Calibre DESIGNrev:

```
calibredrv 3dstack_assembly.oas -rve 3dstack.rdb
```

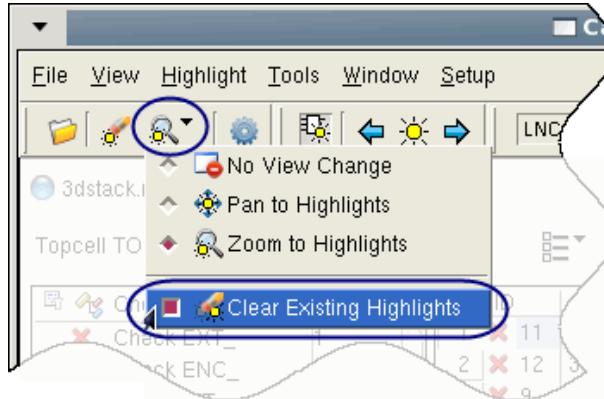
Calibre DESIGNrev displays a top-level view of the assembled 3D chip stack. Calibre RVE displays results for each check defined in the Calibre 3DSTACK *rule_file*.

Tip

i It is also possible to debug a design by viewing the Calibre 3DSTACK report file (generated with the [report](#) command). The report file can be opened from the Report tab in Calibre RVE, or in any supported text viewer. Refer to the [Report File Format](#) in Appendix C for more details on report files.

2. Choose **Clear Existing Highlights** from the Calibre RVE Highlight options dropdown menu.

This option causes Calibre RVE to clear existing highlights before creating a new highlight.



3. Select the check you want to view in the tree view, or use Shift- and Ctrl-click to select multiple checks.
4. In the detailed view, double-click on a result to highlight it, as shown in [Figure 2-5](#), or use the highlight toolbar. Browse each result by clicking the next and previous icons ().

Figure 2-5. Highlighting DRC Results from a Calibre 3DSTACK Run

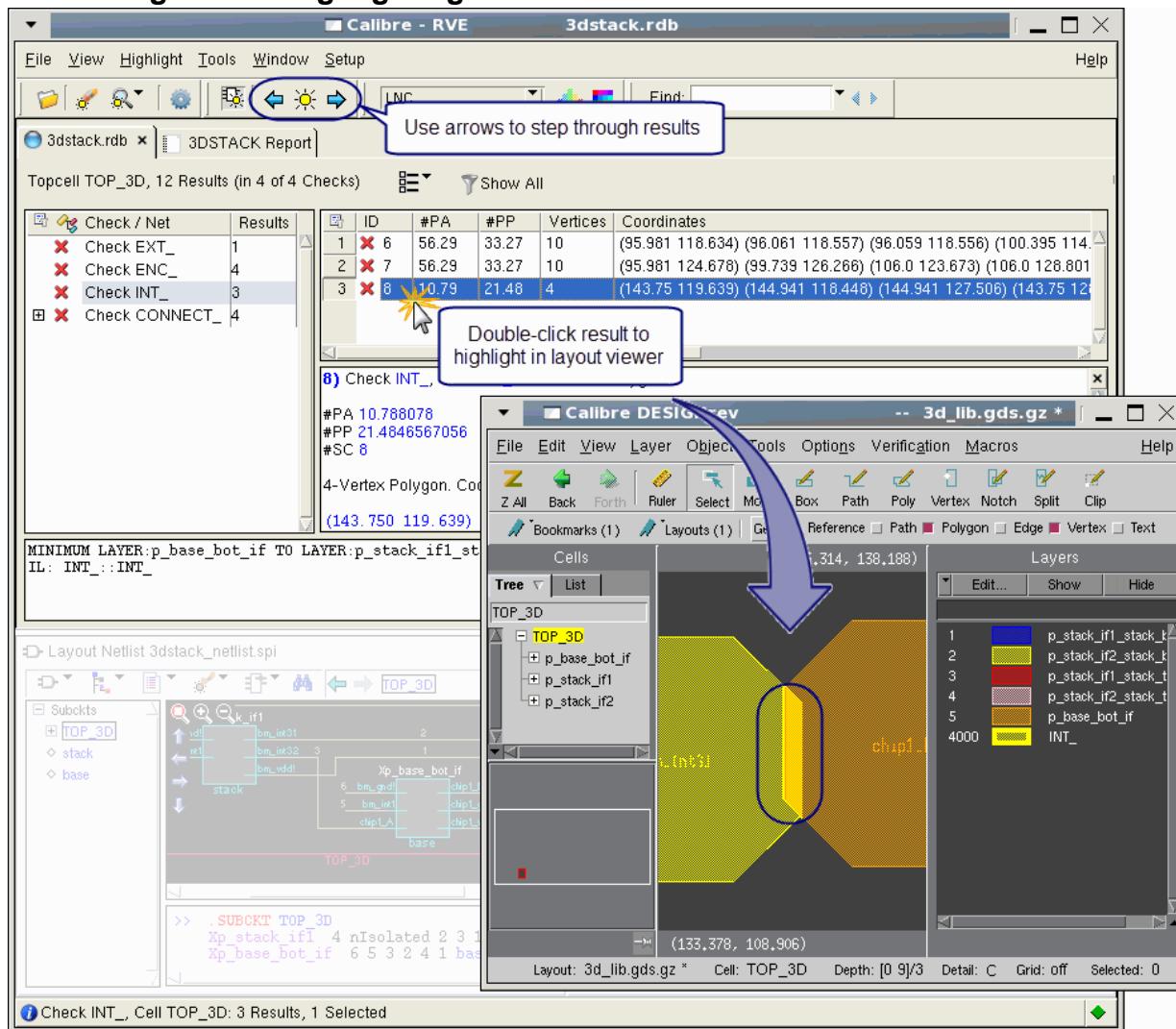


Figure 2-5 shows an [internal check result highlighted in Calibre DESIGNrev](#). See “[Using Calibre RVE for DRC](#)” in the *Calibre RVE User’s Manual* for more information on debugging with Calibre RVE for DRC.

Tip

You can also view DRC errors in the context of the separate components for the assembly. To do this, open the design (the base chip, for example) in your layout viewer, then highlight the result using Calibre RVE.

Results

You performed the following actions while completing this procedure:

- Used Calibre RVE to view DRC results generated by a Calibre 3DSTACK verification.
- Highlighted DRC results from Calibre RVE in Calibre DESIGNrev.

You can control the highlighting behavior in Calibre RVE from your rule file by applying check text override comments.

For complete details, see “[-set_rve_cto_file](#)” on page 125 and “[-set_auto_rve_show_layers](#)” on page 124.

Related Topics

[Calibre 3DSTACK Invocation](#)

[Running a Calibre 3DSTACK Verification](#)

Opening a Calibre 3DSTACK DFM Database in Calibre RVE

Use the cross-referencing capability provided by DFM databases in Calibre RVE to open and debug results from a Calibre 3DSTACK verification run.

Prerequisites

- You have met the “[Requirements](#).”
- You have layout files in GDS or OASIS format that compose the 3D assembly.
- You have specified a netlist that defines connectivity in your chip stack with the [source_netlist](#) rule file command.
- You have a valid 3DSTACK verification results database produced from a Calibre 3DSTACK run. See “[Running a Calibre 3DSTACK Verification from the Command Line](#)” on page 29 or “[Running a Calibre 3DSTACK Verification from Calibre Interactive](#)” on page 30 for information on how to execute a Calibre 3DSTACK run.

Procedure

1. Open the *3dstack_assembly.oas* assembly file in a supported layout editor.

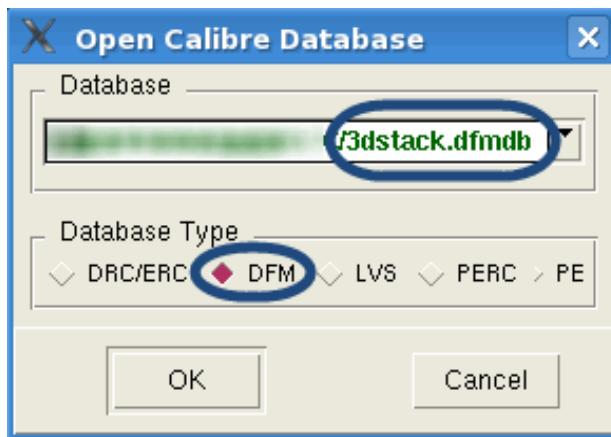
```
calibredrv 3dstack_assembly.oas
```

2. Start Calibre RVE from the layout editor’s graphical user interface:

- Calibre DESIGNrev and Calibre layout viewers — **Verification > Start RVE**.
- Other supported layout viewers — **Calibre > Start RVE**.

3. In the Calibre RVE dialog box, do the following:
 - a. Select a Database Type of DFM (do this before browsing to a database).
 - b. Enter the path to your 3DSTACK database (3dstack.dfmdb):
 - c. Click **Open**.

Calibre RVE automatically opens 3dstack.rdb and the 3DSTACK report generated by the run, as shown in [Figure 2-6](#) on page 43.



4. Setup the Internal Schematic Viewer to display the source and layout netlists as follows:
 - a. Click **Setup > Options** to open the Options tab.
 - b. Select the **Schematic Viewer** category.
 - c. Enable Show netlist schematics when highlighting connectivity objects.
 - d. Enable the Schematic, Hierarchy, and Text options for Show Panes.
 - e. Click **Apply**.
 - f. Choose **View > Schematics > All** to open the layout and source netlists in the Internal Schematic Viewer.

Now, when you highlight a net it is automatically highlighted in the Internal Schematic Viewer in addition to the physical layout.

- g. Close the Options tab, or click the 3dstack.rdb tab to return to viewing 3DSTACK results.

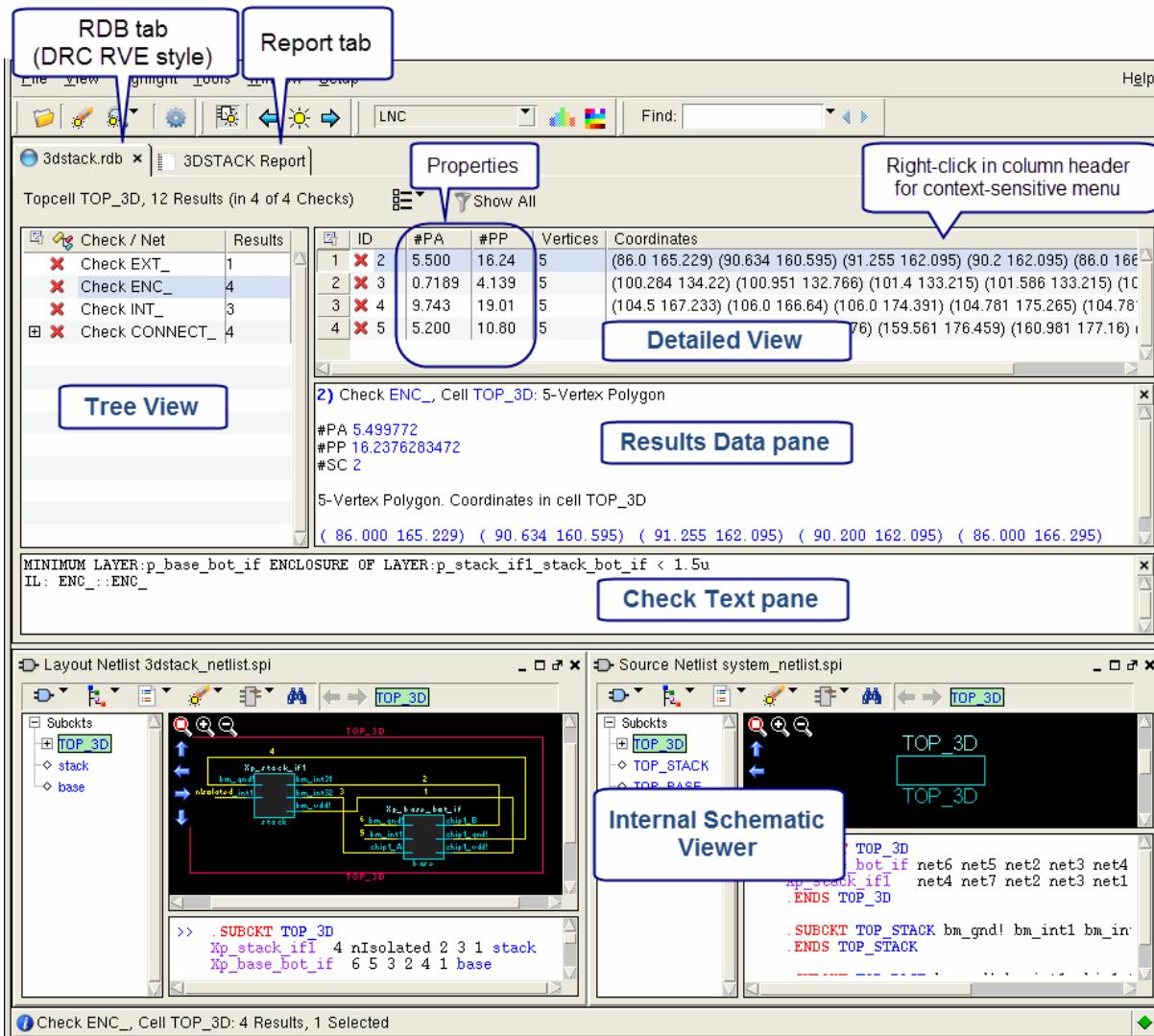
Results

Calibre RVE automatically opens the *3dstack.rdb* results database and the 3DSTACK report file generated by the run, as shown in [Figure 2-6](#). You can also choose **View > Results** and **View > Reports** to see the available results databases and reports. Schematic views of the source and layout netlists are shown in windows below the RDB tab.

The RDB tab is displayed using Calibre RVE for DRC. The display includes the following areas:

- **Tree View** — (left side) Choose **View > Tree Options** to change the grouping.
- **Detailed View** — (top right) Displays property data in table format. Choose **View > Result Options** to change the view.
- **Results Data** — (center right) Shows property data for the result(s) selected in the detailed view.

Figure 2-6. Calibre 3DSTACK Results in Calibre RVE



Tip

i Check names are defined in the rule file with the `-check_name` option. For example:

```
external -check_name EXT_ ...
```

Related Topics

[Highlighting DRC Results from a Calibre 3DSTACK Run](#)

[Viewing Design Elements in a Schematic View](#)

Debugging Connectivity Errors in Calibre 3DSTACK

Use the advanced debugging capability in Calibre RVE to view error results in Calibre 3DSTACK DFM databases.

Connectivity errors are reported by the [connected](#) command in a 3DSTACK rule file.

Note

 In the case where multiple text labels overlap a pad in the layout, the tool generates a multi-text error and chooses to use one of the text labels attached to the pad for the connectivity analysis. Because the chosen label may not match your design intent, it is important to review and resolve all multi-text errors to ensure that the layout is correct. If you do not resolve the multi-text errors, your connectivity analysis may not be correct.

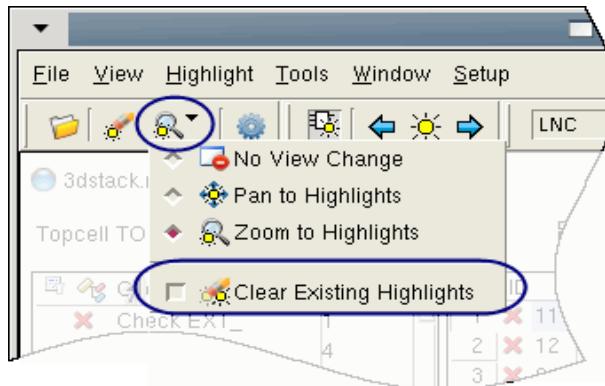
Prerequisites

- A Calibre 3DSTACK database with a connected check using the -detailed option.
The -detailed option adds the LNC (Layout Net Component) and SNC (Source Net Component) properties to the database.
- A Calibre 3DSTACK database opened in Calibre RVE, as described in “[Opening a Calibre 3DSTACK DFM Database in Calibre RVE](#)” on page 41.
- The 3DSTACK assembly open in an attached layout viewer.
- The source and layout netlists open in the Internal Schematic Viewer, as described in Step 4 of “[Opening a Calibre 3DSTACK DFM Database in Calibre RVE](#)” on page 41.

Procedure

1. Disable “Clear Existing Highlights” in the Calibre RVE Highlight options dropdown menu.

This causes Calibre RVE to keep existing highlights, and allows you to view layout net, source net, and result highlights together



2. Choose **Highlight > Highlight Net/Device/Pin** and enable Highlight by layer.

This selection causes Calibre RVE to highlight the different layers of an object in different colors. See “[Highlight Layout Net/Device/Pin Dialog Box](#)” in the *Calibre RVE User’s Manual* for more information on this dialog box.

3. Select a connected check in the tree view of Calibre RVE, then select a result in the detailed view. Review the LNC (Layout Net Component) and SNC (Source Net Component) properties in the Result Data pane; these properties report the ports connected to the respective nets. For the MissingPads check, see Step 6.

Display properties aid you in debugging connectivity errors (refer to [Figure 2-7](#)).

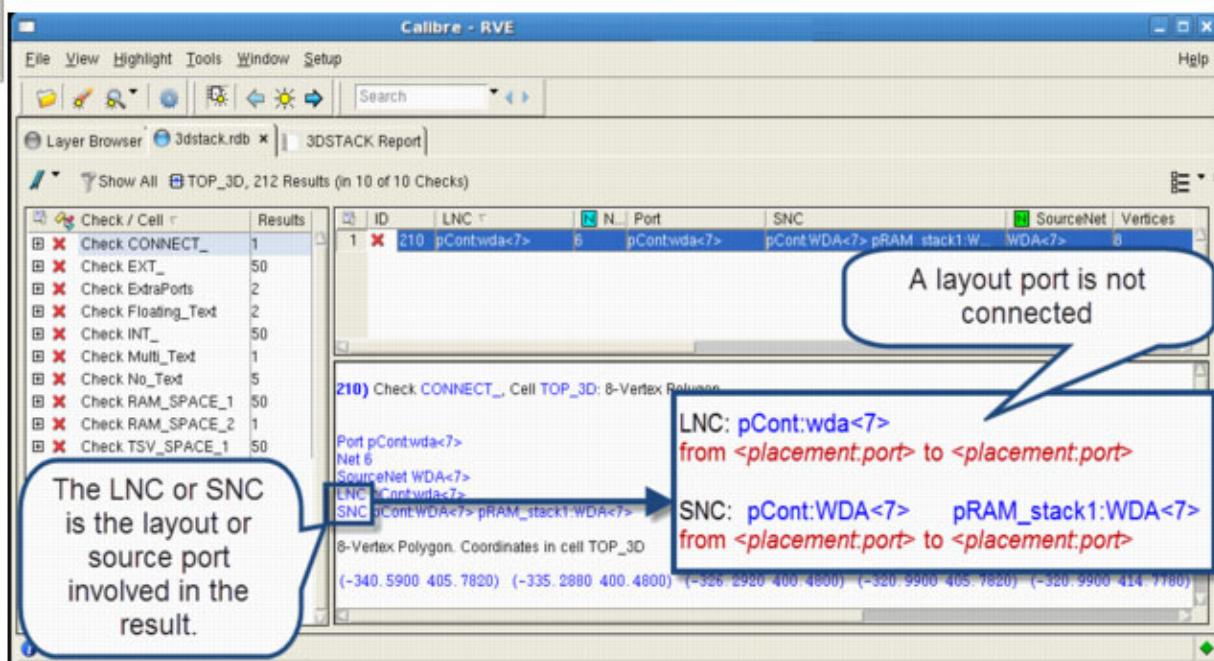
In Calibre RVE, the SourceNet property indicates the source net number of each particular result, and the SNC property indicates the ports that the source net is connected to in the form *placement_name:port*.

For the result shown in [Figure 2-7](#), net WDA<7> is connected to port WDA<7> of placement pCont and to port WDA<7> of placement pRAM_stack_1. Calibre RVE displays these connections as shown in the following table.

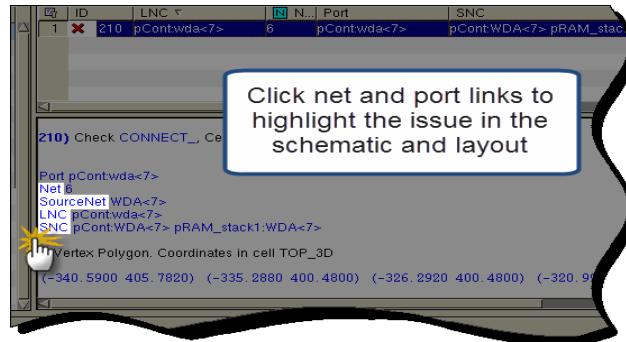
Table 2-1. Source Properties

Property Name	Property Value
SNC	pCont:WDA<7> pRAM_stack1:WDA<7>
SourceNet	WDA<7>

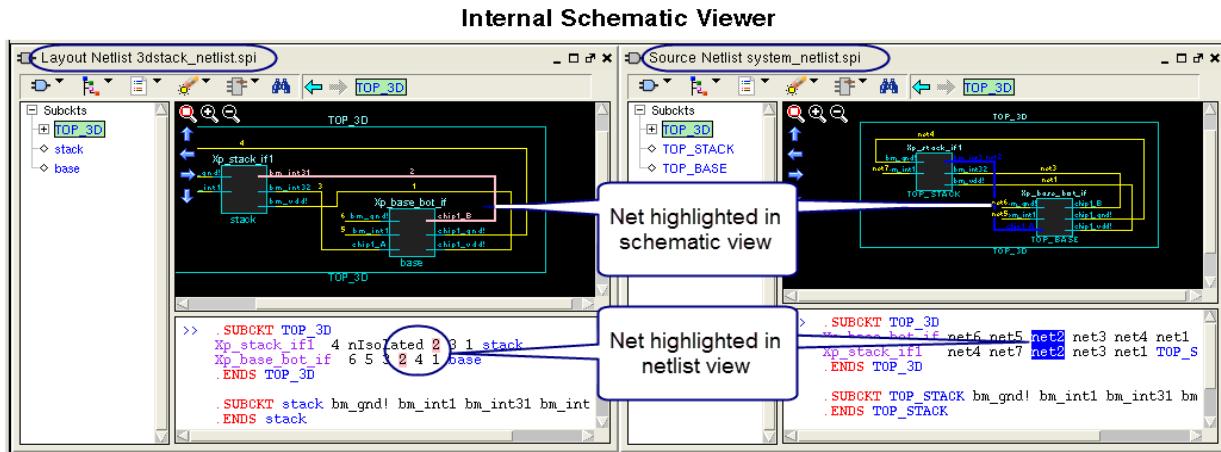
Figure 2-7. Layout and Source Net Details in Calibre RVE



- Click on the layout net (Net ID) and SourceNet links to highlight the nets in the Internal Schematic Viewer.

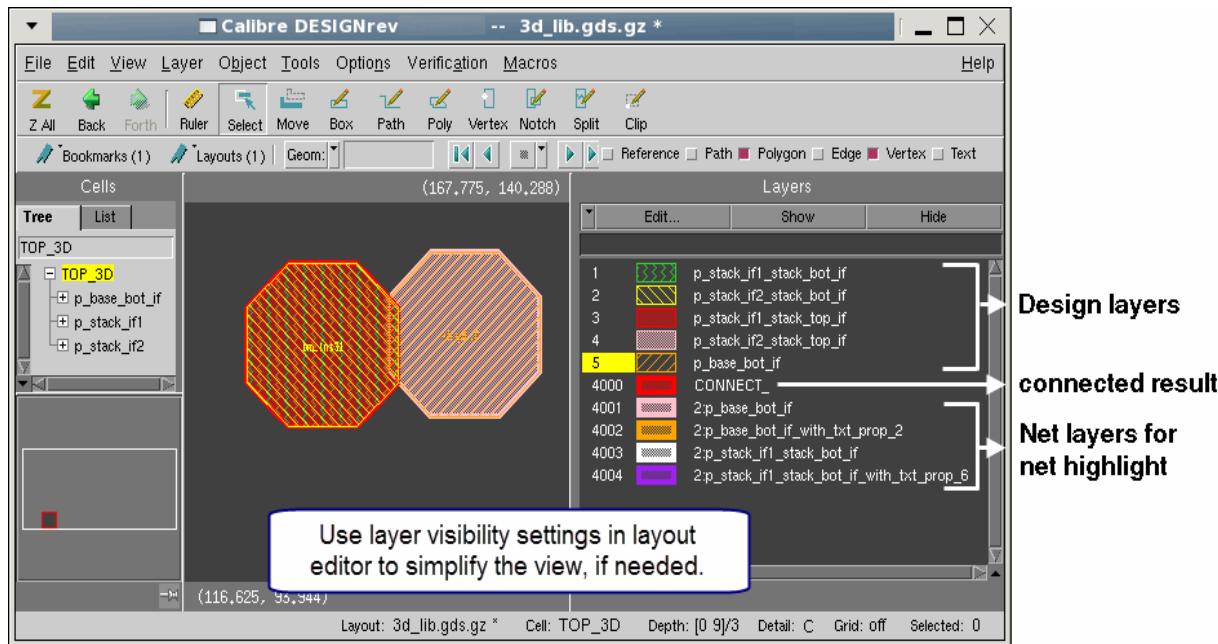


You can also click on the LNC and SNC text strings to highlight the ports on the placements involved in the result. The layout net is also highlighted in the layout viewer.



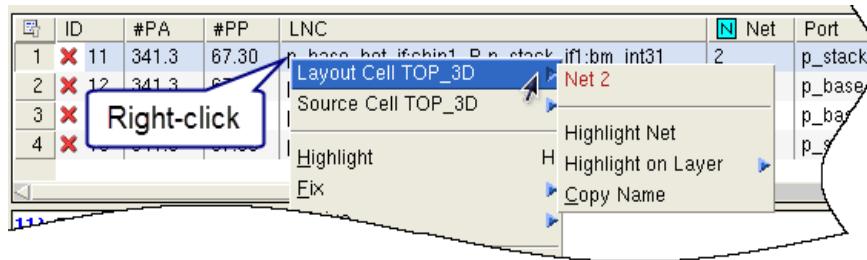
- Click the highlight button () in the highlight toolbar to highlight the result in the layout viewer. Since highlighted shapes and layers may overlap, you may need to use your layout viewer controls to set layer visibility.

The following figure shows the result from a connected check highlighted in Calibre DESIGNrev. The layout net is also highlighted from Step 4.



6. For a MissingPads result, the result properties include links for the missing pad and the source net. (The MissingPads results occurs when a port in the source netlist has no corresponding layout pad.) Do the following to highlight the net and source port:
 - a. Click the “SourceNet” link in the Result Data Pane in Calibre RVE. The net is highlighted in Calibre DESIGNrev and in the source and layout netlist views in the Internal Schematic Viewer.
 - b. Click the “MissingPad” link. The port is highlighted in the source netlist.
 - c. (Optional) Double-click the result in the Details View to highlight it in Calibre DESIGNrev. When the result is highlighted, there is no layout object to highlight, so the result properties are displayed in the bottom left corner of the top cell in the assembly.
7. (Optional) Click the 3DSTACK Report tab to view the text report.

Tip: You can right-click a result in the detailed view for a context-sensitive highlight menu:



The available menu items depend on the properties included with the result.

Results

You performed the following actions while completing this procedure:

- Viewed the LNC and SNC properties in the Calibre RVE display.
- Viewed the source and layout nets in the Internal Schematic Viewer.
- Highlighted the layout net in the layout viewer.
- Highlighted the result from the connected check in the layout viewer.

All these actions provide useful information for debugging a connectivity result in Calibre 3DSTACK results.

Related Topics

[Calibre 3DSTACK Invocation](#)

[Running a Calibre 3DSTACK Verification](#)

Using Path Isolation to Debug Shorts

Shorts can often be difficult to debug in the context of the whole design. Calibre 3DSTACK includes optional features that can help identify only the physical objects that are involved in a connectivity short.

Try It! 	Calibre 3DSTACK Debugging Shorts Tutorial and Example Kit (eKit) <p>This eKit uses example data to demonstrate how to find shorts using path isolation in Calibre 3DSTACK.</p> <p>Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.</p>
---	--

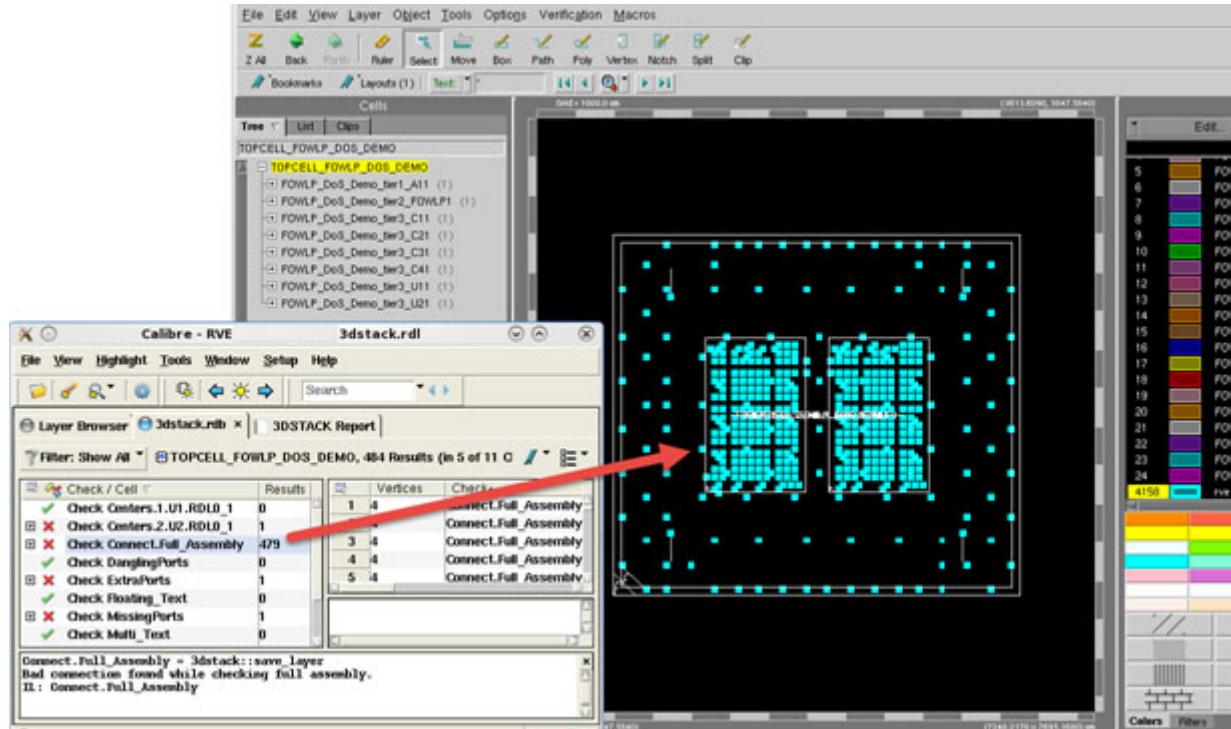
Procedure

1. Ensure that your rule file contains a connected check, for example:

```
connected -check_name Connect.Full_Assembly \
    -comment "Incorrect connection."
```

2. Run Calibre 3DSTACK and open the results in Calibre RVE.
3. In Calibre RVE, right-click the “<connect_check_name>” check and choose **Highlight**.

In the following figure, the path isolation feature has not been enabled and there are 479 connected results. Since the results are distributed throughout the layout, finding the cause of the short is difficult.



4. Close Calibre RVE and add the `-isolate_path` option to a connected check in your rules:

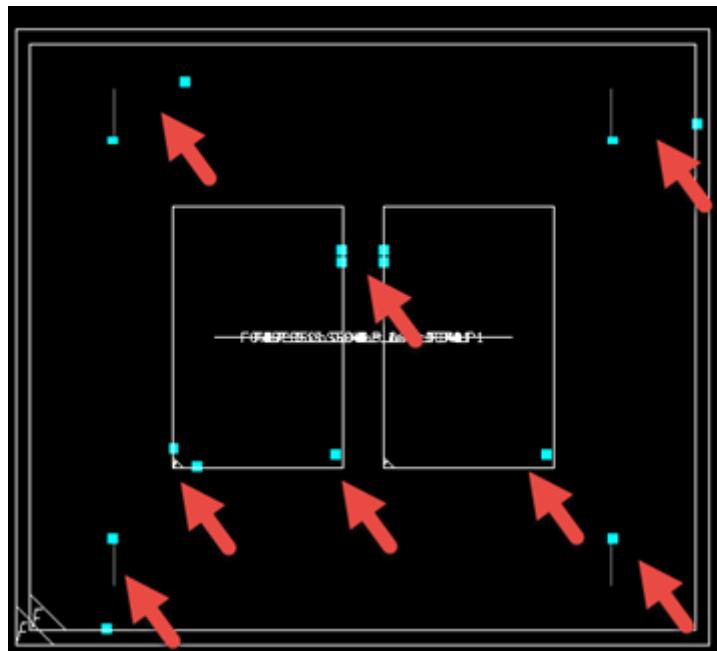
```
connected -isolate_path ...
```

5. Re-run Calibre 3DSTACK and open the results in Calibre RVE.

The results now include an additional check named `<connect_check_name>_isolated`.

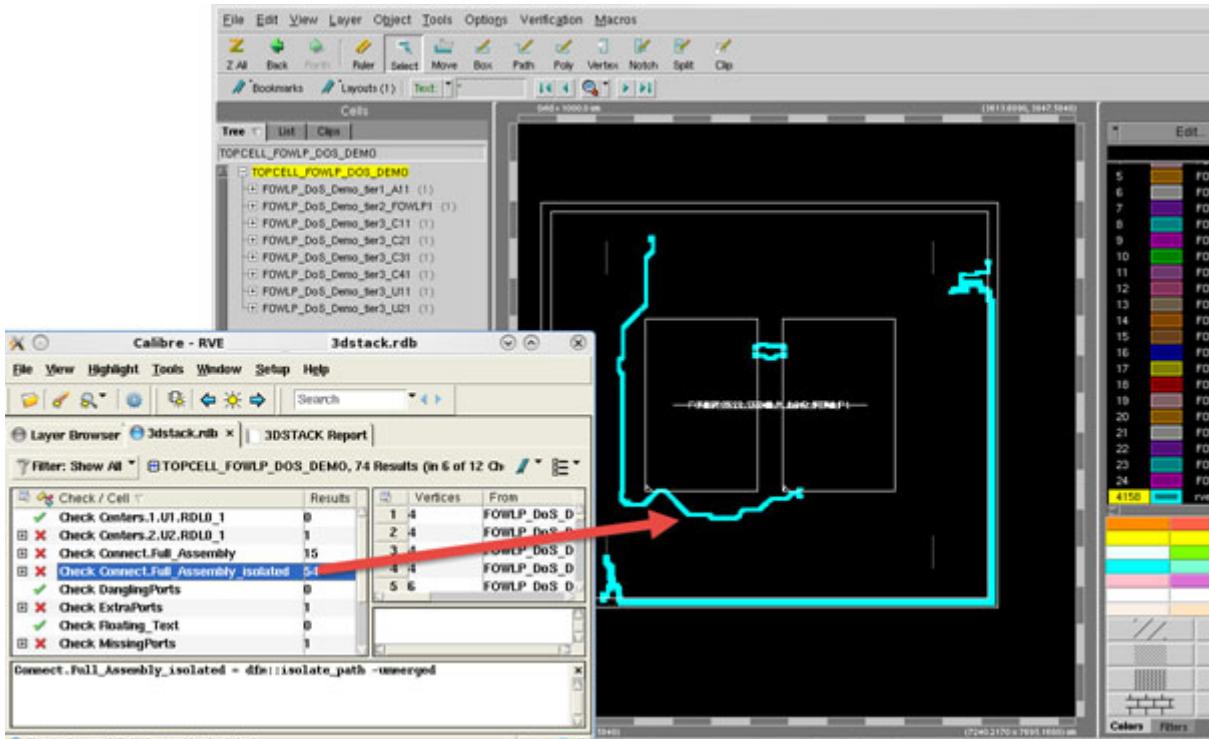
6. In Calibre RVE, right-click the same “`<connect_check_name>`” check that you viewed in the previous run and choose **Highlight**.

The highlighted layout now includes only the physical pins associated with the short instead of all of the objects on the involved nets.

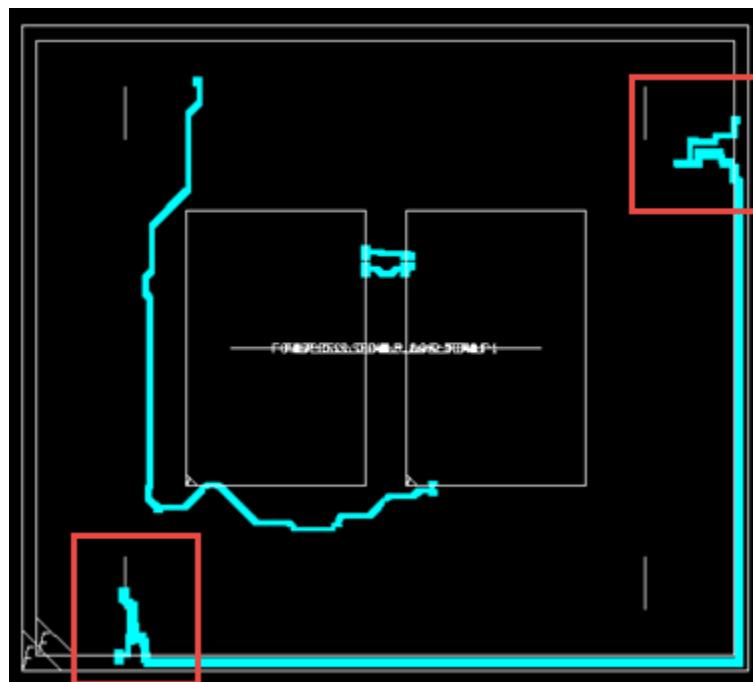


7. Highlight the `<connect_check_name>_isolated` check in Calibre RVE.

The highlighted path now includes only the physical objects associated with the short instead of all of the objects on the involved nets.



8. Investigate the most congested areas of the path. For example, the following areas should be reviewed for shorts:



9. Continue investigating high-density areas until you identify all shorts in the design.

Viewing Design Elements in a Schematic View

Calibre RVE includes an Internal Schematic Viewer which displays a rendered schematic view for both source and layout. The Internal Schematic Viewer is disabled by default for Calibre 3DSTACK results viewing.

Prerequisites

A Calibre 3DSTACK database opened in Calibre RVE, as described in “[Highlighting DRC Results from a Calibre 3DSTACK Run](#)” on page 39.

Procedure

1. Choose **Setup > Options** to open the Options tab.
2. Select the **Schematic Viewer** category.
3. Enable the “Show netlist schematics when highlighting connectivity objects” option.
4. Click **Apply**.
5. Choose **View > Schematics > All** to open the layout and source netlists in the Internal Schematic Viewer.

Tip

 If you close the schematic views, you need to repeat this step to reopen them.

Results

When you highlight a net, device, instance, or pin, the design element is highlighted in both the attached layout viewer and the Internal Schematic Viewer. The Internal Schematic Viewer is displayed in a new window below the results and report tabs.

See “[Internal Schematic Viewer](#)” in the *Calibre RVE User’s Manual* for complete information.

Related Topics

[Calibre 3DSTACK Invocation](#)

[Running a Calibre 3DSTACK Verification](#)

Verification Results Format

Results from a verification run are formatted with certain naming conventions.

Verification results from a run have the following properties that can be used to group results by check or layer type:

Property	Description
Check+	Check name
type1	<i>placed_layer_type</i> or <i>placed_layer_type1</i>
type2	<i>placed_layer_type2</i>
types	<i>placed_layer_type</i> list for density check

Note

 These properties are not added for custom_check results.

Check Result Naming Conventions

The generated check names use the following format (the < > brackets are not part of the format):

`<check-name>_<check-index>`

where:

- *check-name* is the name of the check
- *check-index* is a generated number for the check, starting at 1.

Calibre Interactive for Calibre 3DSTACK

Calibre Interactive 3DSTACK is a graphical interface to the Calibre 3DSTACK verification tool. It allows you to specify command line options and modify rule file statements that execute desired Calibre 3DSTACK operations.

Calibre Interactive facilitates the following types of Calibre 3DSTACK verification operations:

- **Chip assembly operations** — Modify the layouts, assembly operations of the chip stack, layers, and placements.
- **Netlist specification** — Specify a source netlist, verify its syntax, and export the extracted connectivity of your stack.
- **Text operations** — Import, map, or modify text.
- **Verification check selection** — Select specific checks to execute in the verification run.
- **Customization files** — Customize panels in the GUI.

Entries made in the Calibre Interactive interface take precedence over commands in the rule file. Loading the rule file in Calibre Interactive updates the interface with the rule file settings.

For detailed information on Calibre Interactive, see the [Calibre Interactive User's Manual](#).

Note

 If you specify a LEF/DEF database for a die and load your rule file, the tool automatically converts the die to OASIS format and references the OASIS file in Calibre Interactive.

Creating a Source Netlist	54
Specifying a Source Netlist	56
Specifying Report and Results Options	58
Specifying Assembly Instructions.....	59
Selecting Specific Checks	60
Writing a Calibre Interactive Customization File	61
Triggers in Calibre Interactive 3DSTACK	62

Creating a Source Netlist

Use Calibre Interactive to generate a netlist based on specified options if you do not have a complete source netlist for your 3D assembly.

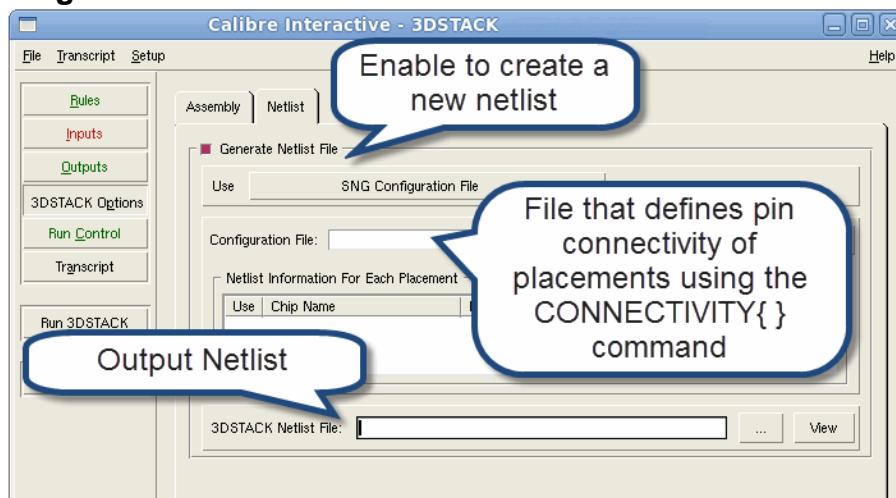
Prerequisites

- Source netlists for each chip in your stack.

Procedure

1. Select the **Rules** button in the left panel of the CI window, define the path to your rule file, and click **Load**.
2. Click **Setup** and enable the “3DSTACK options” checkbox.
3. Click **3DSTACK Options** and click the **Netlist** tab.

Figure 2-8. Create Source Netlist With Calibre Interactive



4. Enable the “Generate Netlist File” checkbox.
5. Specify the path to an existing configuration file that defines the pin connectivity between placements.

The configuration file uses the syntax of a System Netlist Generator configuration file except that it only contains the **CONNECTIVITY** statement’s placement and pin mapping lists. See “[System Netlist Generator Configuration File Format](#)” on page 235 for more details.

The following example connects two pins on two placements together:

```
CONNECTIVITY {
    place_1 {pin_a clk} place2 {connect_to_a connect_to_clk}
}
```

All pins for each placed chip must be connected. Any unconnected pins are reported as errors.

6. Click **Run 3DSTACK**.

The configuration file and your defined placements are passed to the System Netlist Generator tool. The System Netlist Generator uses the given information to generate a source netlist for your assembly.

Results

You have created a source netlist for your 3D assembly by defining connectivity in a configuration file. Calibre 3DSTACK passes this information to the System Netlist Generator and generates a source netlist for your design. The netlist is used in the verification run to perform LVS.

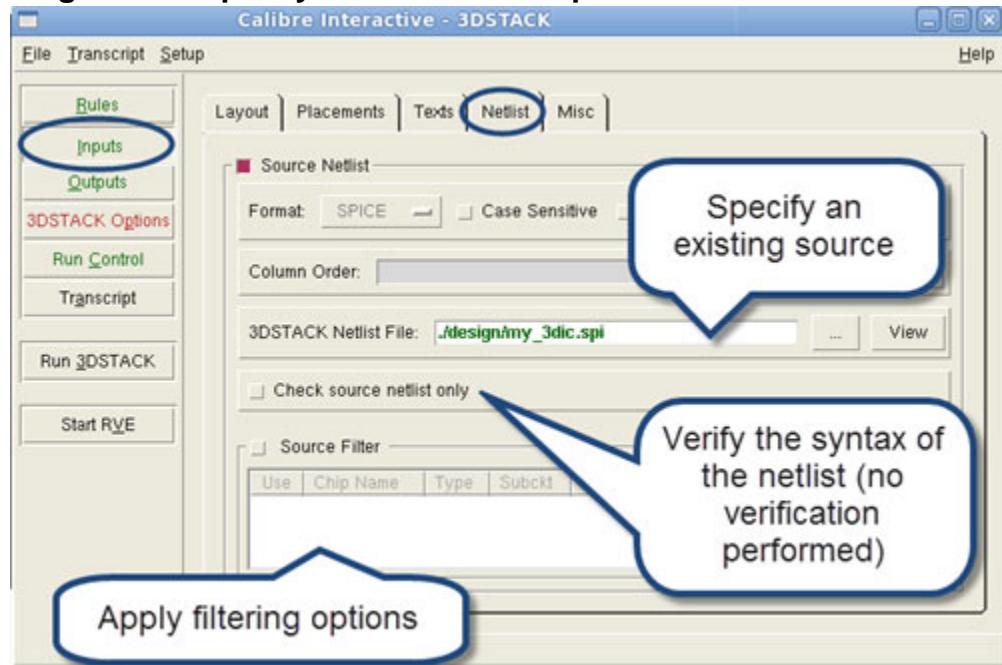
Specifying a Source Netlist

Define and check a source netlist in Calibre Interactive. A source netlist defines the connectivity of your 3D IC. If a source netlist is specified, the connectivity can be traced and compared to the extracted layout netlist. This procedure is optional.

Procedure

1. Click **Inputs** in the left panel of the CI window and click the **Netlist** tab.

Figure 2-9. Specify Source Netlist Options in Calibre Interactive



2. Enable the Source Netlist checkbox
3. Specify the format of the netlist file using the Format dropdown box.
4. (Optional) Enable the Case Sensitivity checkbox if your source netlist file is case-sensitive.
5. (Optional) Enable the Hierarchical checkbox if the source netlist is hierarchical. See “[sng](#)” on page 222 for information on creating a hierarchical source netlist.
6. Specify the path to your netlist file using the 3D Netlist File field. If necessary, select the **View** button to view or modify your source netlist.

Note

 If you specify a CSV netlist or spreadsheet netlist, you can use the Column Order field and Subckt Pins dropdown list to customize the reading of the file.

7. To verify the syntax of the source netlist file and validate placement operations without executing a full verification run, enable the “Check source netlist only” checkbox, and click **Run 3DSTACK**.

Calibre 3DSTACK displays a report file that summarizes the results of the syntax check of your source netlist file. It also reports any invalid placements found in the assembly.

Caution

 When you are ready to run a full verification on your 3D IC, disable the “Check source netlist file only” checkbox before clicking **Run 3DSTACK**.

Results

A source netlist file has been specified and checked for syntax errors and placement issues. A source netlist allows you to perform advanced connectivity debugging with your DFM database verification results. For complete details, see “[Debugging Connectivity Errors in Calibre 3DSTACK](#)” on page 44.

If your source netlist was clean, the report contains the following text:

```
*****
OVERALL SYNTAX CHECK RESULTS
*****
#####
#          #
#      SYNTAX OK      #
#          #
#####
```

Any syntax error is reported as follows:

```
*****
OVERALL SYNTAX CHECK RESULTS
*****
#
#          ######
#  #          #
#          #      SYNTAX CHECK FAILED      #
#  #          #
#  #          ######
#          #
```

Syntax errors were found in the source.

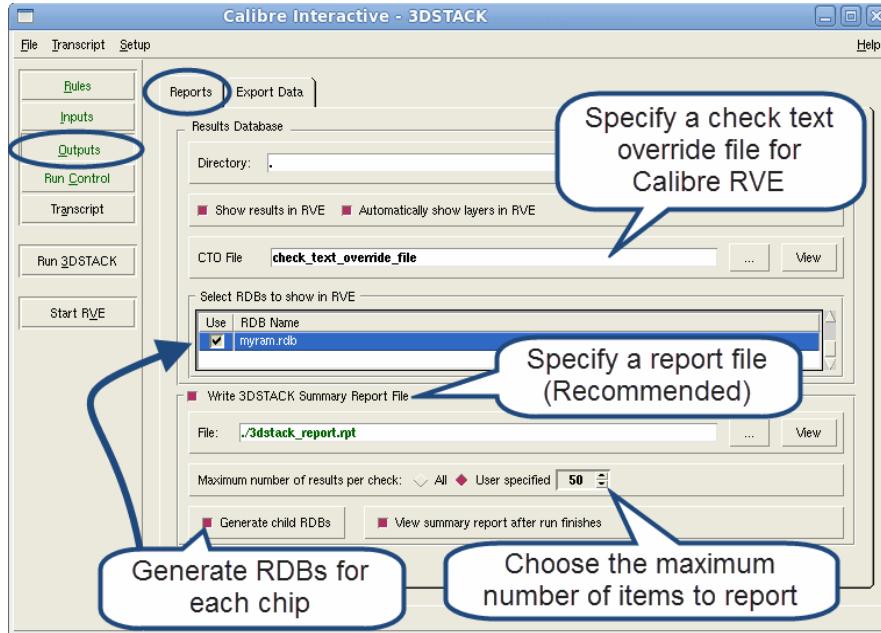
Specifying Report and Results Options

Use Calibre Interactive to specify the path and location of the results database and Calibre 3DSTACK report file. Report files are very useful for debugging runs. This procedure is optional.

Procedure

1. Click on the **Outputs** button to specify optional reporting and output files.

Figure 2-10. Specify Outputs in Calibre Interactive



2. Enable “Show results in RVE checkbox” under the Reports to display results in Calibre RVE following a Calibre 3DSTACK run.

This checkbox is disabled if the Create Only operation is enabled under **Setup > 3DSTACK Options**.

3. Specify a CTO file that controls how Calibre RVE displays rule check results. See [“Check Text Override Comments for Calibre 3DSTACK”](#) on page 213 for details on this file.
4. Enable the “Generate child RDBs” checkbox and use the table to select additional RDBs that you want to generate.

These databases are produced by Calibre 3DSTACK and are opened along with *3dstack.rdb* when the run completes.

5. Enable the “Write 3DSTACK Summary Report File” checkbox and specify a pathname for the report.

You can limit the number of results per check in the report by enabling the User Specified checkbox and entering a positive integer value.

The report file contains important information about the design and verification results. Enabling this option is highly recommended.

Results

The results database file and directory have been specified. These files contain verification results used by Calibre RVE. Additionally, a report file and reporting options have been specified.

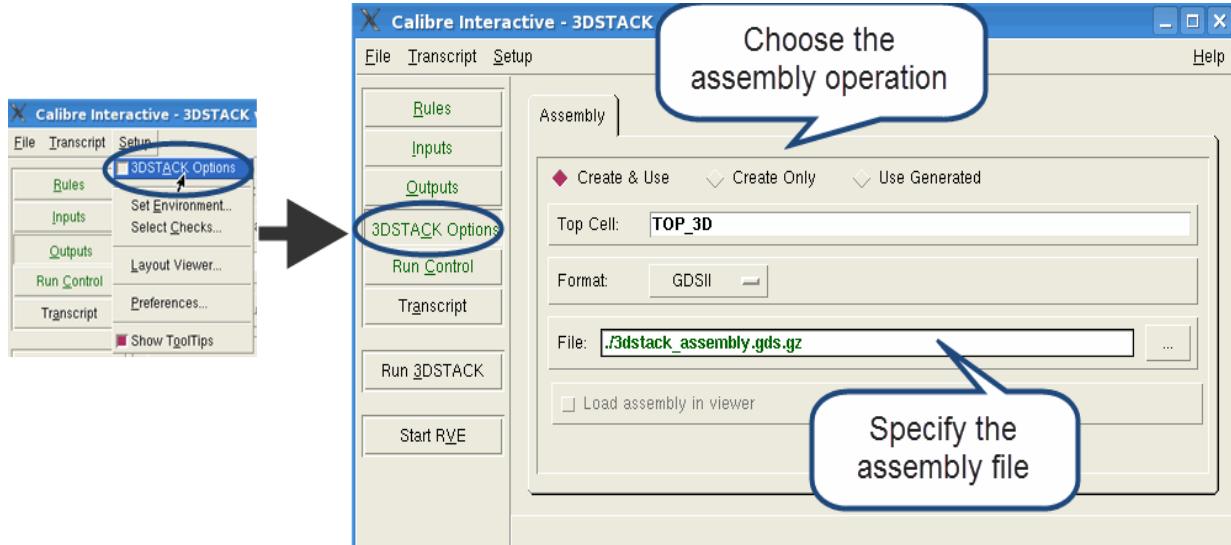
Specifying Assembly Instructions

You can specify the assembly instructions to apply at run time in Calibre Interactive.

Procedure

- From Calibre Interactive, choose **Setup > 3DSTACK Options**.

Figure 2-11. Specify the Assembly Operations in Calibre Interactive



- Click the **3DSTACK Options** button on the left pane of Calibre Interactive and specify the assembly operations to apply at run time.

The assembly options define how Calibre 3DSTACK uses or generates the 3DSTACK assembly view file. The assembly options in Calibre Interactive behave as follows:

- **Create & Use** — Calibre 3DSTACK generates a 3DSTACK assembly file internally and uses it in a verification run. This option invokes Calibre 3DSTACK without the `-create_assembly` and `-use_assembly` options, which is the default mode. See “[Calibre 3DSTACK Invocation](#)” on page 20 for a description of the assembly command options.

- **Create Only**— Calibre 3DSTACK generates a 3DSTACK assembly file and exports template files without executing any further operations (no verification run is performed). This is the same as invoking Calibre 3DSTACK from the command line with the `-create_assembly` option. If this option is selected, the “Show results in RVE” checkbox is disabled.
- **Use Generated** — Calibre 3DSTACK uses an existing assembly file for the verification run. This is the same as invoking Calibre 3DSTACK with the `-use_assembly` option. When this option is enabled, the **Top Cell** and **Format** fields are disabled.

The **Top Cell** field allows you define the top cell of the 3D layout. The **File** field specifies the path to the assembly file.

Note

 The “Load assembly in viewer” checkbox is only enabled when you open Calibre Interactive from Calibre DESIGNrev. Enable this option to open the assembly view layout in Calibre DESIGNrev immediately following a run.

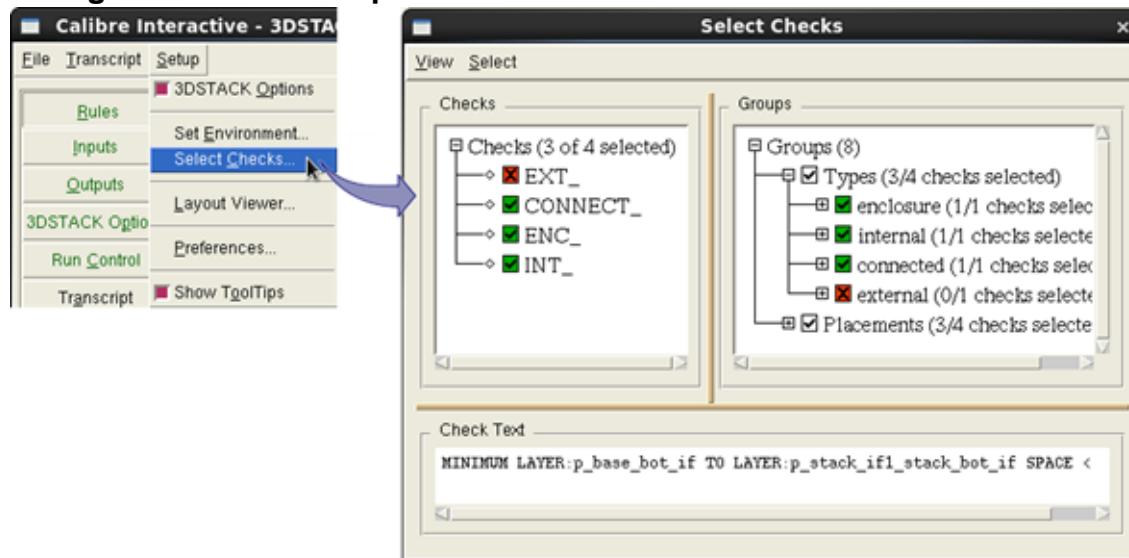
Selecting Specific Checks

Use Calibre Interactive to select specific checks for a verification run.

Procedure

1. Select **Setup > Select Checks**. The Select Checks dialog box opens. Checks in the Groups area are grouped by type and placement—expand the hierarchy as needed.

Figure 2-12. Select Specific Verification Checks in Calibre Interactive



2. Click the checkbox next to the check name or check group to enable or disable the checks at run time. A green check mark indicates that the check is enabled, a red cross indicates that the check is excluded.

Note

 Use the **View** menu to sort the checks and groups. Use the **Select** menu to toggle check selection.

3. Enable or disable the “Types” checkbox to select or unselect all checks.

Results

Checks to execute at run time have been specified. The selections made in the Select Checks dialog box are not saved to the Calibre Interactive runset file.

Writing a Calibre Interactive Customization File

Use Calibre Interactive to write a Calibre Interactive Customization settings file.

See “[Customization Files](#)” in the [for more details](#).

Procedure

1. Open Calibre Interactive for Calibre 3DSTACK and load a rule file.
2. Create a new text file called *select_checks.tcl* in your working directory and insert the following text:

```
set _customM_0 [CUSTOM::VARIABLE -name "select_by" -choices \
    {{"All checks" 0} {"by group" 1}} -initval {1} -select 1 -boolean 0 \
    -prompt "select checks by:" -display 1 -tool 3DSTACK ]
CUSTOM::LABEL -prompt "Select checks by group " -tool 3DSTACK \
    -master [list $_customM_0 1]
CUSTOM::CHECK -name "connected" -choices {connected} -initval {connected} \
    -select 1 -boolean 1 -prompt "connected" -display 1 -tool 3DSTACK \
    -master [list $_customM_0 1]
CUSTOM::CHECK -name "internal" -choices {internal} -initval {internal} \
    -select 1 -boolean 1 -prompt "internal" -display 1 -tool 3DSTACK \
    -master [list $_customM_0 1]
CUSTOM::CHECK -name "external" -choices {external} -initval {external} \
    -select 0 -boolean 1 -prompt "external" -display 1 -tool 3DSTACK \
    -master [list $_customM_0 1]
CUSTOM::CHECK -name "enclosure" -choices {enclosure} -initval {enclosure} \
    -select 0 -boolean 1 -prompt "enclosure" -display 1 -tool 3DSTACK \
    -master [list $_customM_0 1]
```

Save the file and close it.

3. In Calibre Interactive, select **Setup > Preferences** and enable the “Customization File” checkbox.

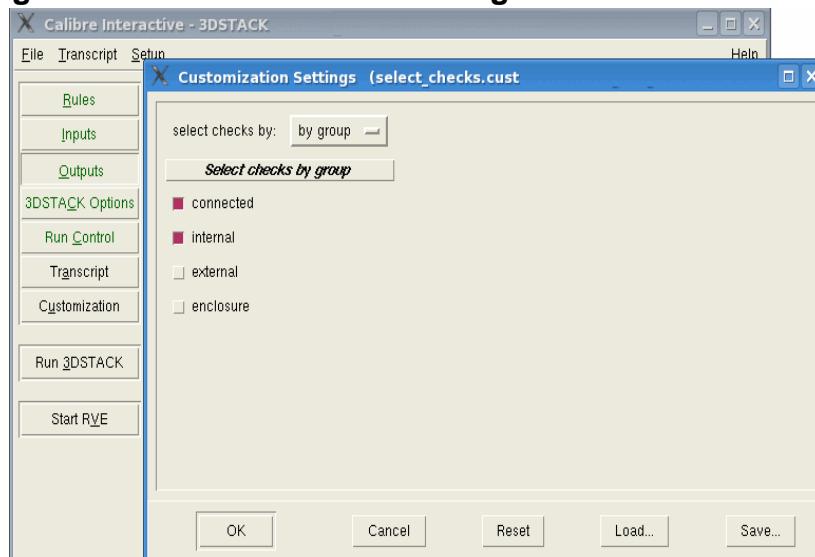
4. In the file field, browse to the path where you saved the *select_checks.tcl* file and click **OK**.

A new **Customization** button appears under the **Transcript** button on the left panel of Calibre Interactive.

5. Click on the **Customization** button.

The *select_checks.tcl* file, shown in Step 2, enables modification of the select checks command.

Figure 2-13. Customization Settings in Calibre Interactive



6. Choose the checks to include in the verification run and click **OK** to close the dialog box.
7. Choose **Setup > Select Checks** to verify that the checks have been selected as intended.

Results

A Calibre Interactive customization file has been created and used to modify the select checks command. This particular example is useful for saving select check options after Calibre Interactive is closed.

In this case, if you choose the **All Checks** option, no commands are added to the control file, and therefore all checks are run. If you choose the **By Group** option, you may choose to select checks by the type of rule check.

Use this example to create your own customization file for Calibre Interactive.

Triggers in Calibre Interactive 3DSTACK

Calibre Interactive provides the ability to specify and run pre- and post-execution internal triggers. A trigger can be any program or script executed from a shell.

The trigger program is run immediately before or after a batch Calibre 3DSTACK run. For complete information on Calibre Interactive triggers, see “[Trigger Functions in Calibre Interactive](#)” in the *Calibre Interactive User’s Manual*.

To access the Trigger options in Calibre Interactive 3DSTACK, select **Setup > Preferences** and click on the **Triggers** tab. The following trigger parameters are available.

Table 2-2. Supported Trigger Parameters

Parameter	Runtime Replacement
%r	Calibre 3DSTACK rule file
%R	Calibre Interactive runset file
%t	Top cell of the 3D assembly
%e	Calibre termination status
%P	Netlist placement table content (Inputs>Netlist tab)
%d	Run directory
%m	Calibre 3DSTACK report file
%C	System Netlist Generator Configuration file
%n	3D Netlist file

Chapter 3

Command Reference

The Calibre 3DSTACK rule file uses its own set of commands to assemble and verify your 3D-IC design intent.

This section documents the extended (3DSTACK+) syntax, which is recommended for all new rule files. At runtime, your specified extended syntax rule file is translated to a standard syntax file in your working directory. The translated standard syntax file is useful for debugging your run.

The standard syntax is documented under “[Standard Calibre 3DSTACK Syntax Commands](#)” on page 357.

Rule File Format and Syntax	67
System Variables	69
Assembly and Configuration Commands	71
die	72
component	90
stack	94
connect	100
config	102
process	129
export_layout	131
Rule Check Commands	139
centers	142
connected	147
copy	158
custom_check	160
dangling_ports	164
dangling_no_text	166
density	167
enclosure	175
external	178
extra_ports	182
floating_pads	184
floating_texts	186
internal	188
locations	190
missing_ports	193
multi_texts	195
no_texts	197

offgrid_centers	199
overlap	201
same_size	204
select_checks	206
unconnected_ports	208
unselect_checks	209
3dstack_block	211
Check Text Override Comments for Calibre 3DSTACK	213

Rule File Format and Syntax

Input for: Calibre 3DSTACK

File format used to define the physical assembly of a 3D or 2.5D IC.

The Calibre 3DSTACK rule file supports an extended version of the 3D-IC Description Language (3D-IC DL) commands, referred to as 3DSTACK+. The 3DSTACK+ syntax is a universal language that allows you to define your assembly and rule checks and supports some advantages over the standard syntax.

The Calibre 3DSTACK+ syntax can be used to decouple rule and analysis specifications from the physical assembly, similar to a typical design flow. This enables you to reuse rule files for multiple assemblies.

Note

 The tool writes temporary files to `/tmp` by default. Set the `MGC_TMPDIR` environment variable to a different path to customize the temporary file output location.

Format

A Calibre 3DSTACK+ rule file must conform to the following formatting and syntax rules:

- The file is parsed as a Tcl file. It must follow standard Tcl syntax and formatting rules. Standard Tcl constructs are supported and can be useful for parameterizing your assembly (for example, using loops and variables to define layers and die locations).

Tip

 You can optionally create additional information about Tcl errors generated from your Calibre 3DSTACK rule files by setting the `CALIBRE_ENABLE_3DSTACKPLUS_DECK_DUMP` environment variable to a non-null value. This option will export a file named `<rule_file>_exported.3dstack+` that contains enables you to debug the source of Tcl errors in your rules.

- There is a required opening line and `set_version` command:

```
#!3dstack+
set_version -version 1.0
```

- The main assembly commands are `die`, `config`, and `stack`. These are summarized in the following section.
- All distances are specified in microns.
- The System and Assembly commands must be written in the order as listed in the following table. Rule checks should be written last, as the rule checks usually depend on the layers, dies, and assembly configuration.

- It can be convenient to include sets of operations in different files and then source them from a main rule file. For example, you may have the following files:

```
main.3dstack
processes.3dstack
die_definitions.3dstack
stacked_system.3dstack
rule_checks_<process>.3dstack
```

The *main.3dstack* rule file calls the other rule files as follows:

```
#!3dstack+
set_version 1.0

#define layers
source "./processes.3dstack"

#define dies
source "./die_definitions.3dstack"

#define assembly
source "./stacked_system.3dstack"

#run verification checks
source "./rule_checks_16.3dstack"
```

Parameters

Table 3-1. Extended Calibre 3DSTACK+ Command Summary

Keyword	Usage	Description
System Commands		
run	Optional	Enables you to run other Calibre verification rules, such as DRC, LVS, and extraction, on layouts in your stack.
set_version	Required	Must specify the version as 1.0.
Assembly Commands		
process	Optional	Specifies information about a collection of layers that can be referenced in a die or component.
die	Required	Defines a single die (or interposer) in the assembly. This statement includes a number of arguments that define the name, layout path, and physical information about the die, such as the material, layers, and dimensions.

Table 3-1. Extended Calibre 3DSTACK+ Command Summary (cont.)

Keyword	Usage	Description
component	Required	Defines a single component (such as a resistor) in the assembly. This statement includes a number of arguments that define the name, layout path, and physical information about the component, such as the material, layers, and dimensions.
connect	Optional	Defines explicit connectivity between die layers in a stack.
stack	Required	Defines the locations of dies and components in the assembly.
Configuration and Export Commands		
config	Optional	Specifies configuration information, such as the top cell name, source netlist, report file, and connectivity.
export_layout	Optional	Generates layouts of dies in the assembly.
Rule Check Commands		

Examples

```
#!3dstack+
=====
# Calibre 3DSTACK+ Rule File
#
set_version -version 1.0

config <options>

die <options>
...

stack <options>
...

<verification commands>
...
```

System Variables

The 3DSTACK+ file format supports a set of system variables that allow you to easily reference layer placements based on the specified type.

The following variables are available:

- `_layer_type`
- `_layer_types`

- `_layer_type1`
- `_layer_type2`

The following examples demonstrate how to apply the variables in your rule file:

```
connected -check_name chk \
    -stack P \
    -layer_type1 pad1 \
    -layer_type2 pad1 \
    -comment "CONNECTION CHECKS between ${_layer_type1} pads
              interacting with ${_layer_type2} pads" \
    -set_rve_show_layers "${_layer_type1} ${_layer_type2}"

density   -check_name chk \
    -stack P \
    -layer_types {pad1 pad2} \
    -constraint ">0" \
    -comment {DENSITY CHECKS between ${_layer_types}} \
    -set_rve_show_layers "${_layer_types}"
```

Assembly and Configuration Commands

The 3DSTACK+ rule file can contain system, assembly, and rule check commands.

See the “[Rule File Format and Syntax](#)” on page 67 for a description of the rule file format. See “[Rule Check Commands](#)” on page 139 for a description of the rule check options specific to the 3DSTACK+ extended syntax.

In addition to the [run](#) command, the following commands are supported in Calibre 3DSTACK+ rule files.

Table 3-2. Assembly and Configuration Commands

Command	Description
die	Defines a single die in the assembly. This statement includes a number of arguments that define the name, layout path, and layers.
component	Defines a single component in the assembly. A component can be an object, such as a resistor, capacitor, or inductor, but it cannot be a die.
stack	Defines the locations of objects in the assembly.
connect	Defines explicit connectivity between die layers in a stack. This enables you to manually define connectivity between layers instead of using only the implicit connectivity provided by the physical assembly.
config	Specifies configuration information for the 3DSTACK+ syntax format rule file. The top cell name, source netlist, report file, connectivity, and other information can be specified.
process	Defines technology kit layer information for the dies, interposers, packages, and other components used in a 3D-IC.
export_layout	Generates an annotated GDS and extraction data for a specified portion of the assembly.

die

Required in 3DSTACK+ extended syntax file.

Defines a single die in the assembly. This statement includes a number of arguments that define the name, layout path, and layers.

Usage

```
die -die_name die_name
  {-layout '{' arguments '}' | -lefdef '{' arguments '}'}
  [-thickness value]
  [-process process_name]
  {-layer_info '{' arguments '}' } ...
  [-anchor '{' -name name {-placement x_offset y_offset | -layer number -text label} '}' ] ...
  [-interposer | -package | -laminate | -substrate]
  [-rename_text {"expression"... | {-file file} }]
  [-import_text_labels file
    [-order column_list]
    [-xsi {NET_NAME | PIN_NUMBER | PIN_NAME}]
    [-attach_to {-layer layer_name -component component_name} ...]
    [-top_level_coords]]
  [-netlist spice_file]
  [-wb_connect layer1_name layer2_name [BY layer3_name] [-use_in_svrf]] ...
```

Arguments

- **-die_name die_name**

Required argument and value set that specifies the name of the die. The die can be thought of as a cell in your design. The name must be unique within the assembly.

- **-layout '{' *arguments* '}'**

Required argument and value set that specifies the path, layout system type, and top cell of a layout. See “[-layout](#)” on page 82 for details.

- **-lefdef '{' *arguments* '}'**

Argument and value set that specifies the paths to LEF and DEF files, if a LEF/DEF layout system is used for the die. See “[-lefdef](#)” on page 83 for details.

- **-thickness *value***

Optional argument set that defines the thickness of the die in microns. Calibre 3DSTACK uses the thickness values to generate a cross-sectional assembly view of the stack. See “[Assembly Views](#)” on page 35 for details.

If this option is not specified, you can use the

CALIBRE_3DSTACK_DEFAULT_THICKNESS environment variable to set a global thickness for all dies. If neither the environment variable nor the -thickness option is specified, the thickness of the placement level is calculated based on the minimum width and height of the dies in the same level.

- -process *process_name*

Optional keyword and name for process layer information defined with the “[process](#)” command.

The process layer information is usually provided by the foundry that manufactures your IC. The advantage of applying this command is that you do not have to redefine layer information for each die in the 3D-IC. Instead, you can refer to a process that is already defined in the die command.

Note

 If you specify layer information in addition to a process in the die command, the layer information overwrites the process specification if the same information is defined in the process. If items are not defined in the layer information, the process data is merged with the layer information.

- **-layer_info ‘{’ *arguments* ‘}’**

Required argument and value set that specifies a layer in the die. You can specify more than one layer by applying the -layer_info argument set multiple times. Note that you must enclose the arguments for each layer within the {} braces.

See “[-layer_info](#)” on page 84 for details.

- **-anchor ‘{’-name *anchor_name* {-placement *x_offset* *y_offset* | -layer *number* -text *label*}‘}’**

Optional argument and value set that creates an anchor with a specified name at an x and y coordinate specified in microns. You can specify multiple anchors per die. The anchor is a labeled coordinate that can be referenced in assembly operations for alignment purposes.

The options are defined as follows:

- name *anchor_name* — specifies the name of the anchor.
- placement *x_offset* *y_offset* — specifies the coordinates of the anchor in microns.
- layer *number* — specifies the anchor layer.
- text *label* — specifies the label name to identify the anchor location on the layer.

- **-interposer | -package | -lamine | -substrate**

Optional arguments that specify that the layout serves the purpose of an interposer. The -interposer, -package, -lamine, and -substrate arguments are interchangeable; they are all equivalent to specifying -interposer. The different names serve as a label that more closely identifies the component in your package. Apply one of these options if you want to generate a hierarchical netlist or create specific rules for an interposer. You can apply this option more than once in your assembly. If you specify more than one interposer, Calibre 3DSTACK generates a flat assembly netlist.

- `-rename_text {"expression" ... | {-file file}}`

Optional argument set that renames text in the die layout. The expression is a GNU regular expression. Multiple expressions are supported, but the entire set of *expressions* must be enclosed in quotes or braces. For example:

```
-rename_text "/</\[/ />/\]/"
```

The above statement contains two regular expressions, the first (/</\|) replaces an opening angled bracket (<) with an opening square bracket ([) and the second />\|/) replaces the closing angled brackets with closing square brackets.

Instead of in-line expressions, you can specify the path to a file that contains a list of expressions, as follows:

```
-rename_text {-file ./rules/regex.txt}
```

Where the regex.txt file in this example contains one expression per line. For example:

```
/A/B/  
/A.*B/R/  
/^VDD.*/VCC/  
/A/B/g
```

- `-import_text_labels file {arguments}`

Optional argument set that specifies a text label file. See “[import_text_labels](#)” on page 402 for more details. If you specify an Xpedition Substrate Integrator® (XSI) netlist as the *file*, then the tool uses this file to generate and attach text labels in the layout.

Note



This argument set can only be used with an XSI file if you specify a LEF/DEF layout.

The following *arguments* only apply to XSI files:

`-order column_list`

Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF_DES, PIN_NUMBER, PIN_X, PIN_Y, PIN_NAME, NET_NAME, COMPONENT_NAME, and IGNORE. The default order is as follows:

```
NET_NAME IGNORE IGNORE COMPONENT_NAME PIN_NUMBER REF_DES
```

`-xsi {NET_NAME | PIN_NUMBER | PIN_NAME}`

Optionally specifies the column in the XSI netlist from which to retrieve text labels. The default is the NET_NAME column.

`-attach_to {-layer layer_name -component component_name} ...`

Specifies the layer number and datatype of the layout to which the labels are attached. The -component *component_name* argument set specifies to generate text labels for

the given component in the assembly. Specify at least one -layer and -component argument pairs.

-top_level_coords

Specifies that the coordinates for text labels in the *file* belong to the top level.

- **-netlist *spice_file***

Optional keyword and path to a SPICE netlist that contains the subcircuit definition for the specified layout. If you specify this option, Calibre 3DSTACK automatically calls the System Netlister tool to build a complete netlist during a verification run.

- **-wb_connect *layer1_name layer2_name [BY layer3_name] [-use_in_svrf]***

Optional argument set that defines the connectivity of the layers inside of the die. The syntax is similar to the Connect SVRF statement. This option is only required for hierarchical (white-box) connectivity extraction of the assembly.

The -use_in_svrf option enables you to use connectivity-based SVRF statements, such as Net Area Ratio, in the -layer_info -svrf option based on the derived connectivity layers in -wb_connect. For example:

```
die -die_name TOP \
    -layout {
        -primary TOPCELL_3DI
        -path ./3dstack.gds      -type gdsii
    } \
...
    -layer_info {
        -type LAY_BOTTOM_O
        -name LAY_BOTTOM_O
        -layer 1
    } \
    -layer_info {
        -type LAY_OR
        -name LAY_OR
        -svrf { LAY_TOP OR LAY_BOTTOM }
    } \
    -layer_info {
        -type BOTTOM
        -name LAY_BOTTOM
        -svrf { NET AREA RATIO LAY_BOTTOM_O LAY_OR > 0 }
    } \
...
    -wb_connect LAY_TOP LAY_OR -use_in_svrf \
    -wb_connect LAY_BOTTOM_O LAY_OR -use_in_svrf \
...
```

Description

The die command is the primary building block for chips in your assembly. It defines the following components of a die:

- Layout path and layout format (GDSII, OASIS, or LEF/DEF)
- Physical geometry (length, width, and thickness)

- Anchor points for assembly alignment
- Layers used in the die
- Layout type: interposer or regular die
- Hierarchical connectivity

The die layout is defined from a GDS or OASIS file or from LEF/DEF files.

The anchors are labels fixed to coordinates on specified dies that can be referenced in the stack definition for alignment purposes. For example, you can specify an anchor at the center of a die called controller_center and then use the controller_center label in your assembly operations to place other dies relative to the center of the controller.

The -interposer argument has no impact on the assembly, but it can be referenced in downstream rules to apply specific checks for 2.5D ICs.

Use the -text option to relate text labels to the layers of the dies. These associations are required if there are connectivity checking rules defined in the deck. The -no_update argument should be used when associating text with a source placement layer for use with a locations check. This prevents the source layer from being included in the extracted layout netlist.

Multiple layers may be specified for each die. Use the -svrf option to generate derived layers within the die.

The hierarchical (white-box) connections define the connectivity of the materials inside of the die for netlist extraction purposes. When specified, the intra-die and inter-die connectivity is taken into account during connectivity rule checking.

Through-tier connectivity checks are supported. This feature allows you to verify connections that pass through multiple dies. For example, if you stack ChipA on top of an interposer and ChipB on the bottom of the interposer, you can verify the connectivity of the traces from the pads of ChipA, through the interposer layers, and down to the pads of ChipB. This feature is achieved by defining internal layers for a die without specifying the -top or -bottom options for those layers.

The -layer_info argument of the die command allows you to specify whether the layer belongs to the top or bottom surface of the die. If you do not specify either -top or -bottom, the layer is considered internal to the die and is used for internal connectivity tracing.

Note

 If you do *not* want to check connectivity through tiers, apply the -nothrtier option in the connected check.

Examples

Example 1

The following command defines a new die based on an existing layout for a memory device.

```
die \
-die_name RAM_block \
-layout { -path ./RAM.oas -type OASIS -primary RAM } \
-thickness 150 \
-layer_info { -type pad -name pad1 -top \
    -layer { 40:2,41:2 } -text { 40:2,41:2 } } \
-layer_info { -type pad -name pad2 -bottom \
    -layer { 8-15:0 } -text { 24-39:1 } } \
-layer_info { -type bump -name bump1 -top \
    -layer { 2-7,16-23 } } \
-layer_info { -type bump -name bump2 -bottom \
    -layer { 40-41:3-7 -depth top_only } -text { 40:3 -depth 1 } } \
} \
-rename_text "/a_bump/a/"
```

The new die is given the name RAM_block. The die definition includes the following layers:

Table 3-3. Layer Information for the RAM_block Die

Name	Type	Vertical Location	Layer Numbers		Text Numbers		Depth
			Number	Datatype	Number	Datatype	
pad1	pad	Top	40, 41	2	40, 41	2	All
pad2	bump	Bottom	8-15	0	24-39	1	All
bump1	bump	Top	2-7, 16-23	any	none	none	All
bump2	bump	Bottom	40-41	3-7	40	3	Top cell

- pad1 and pad2 are of type pad, and bump1 and bump2 are of type bump. pad1 and bump1 are in the top surface of the die, and pad2 and bump2 are in the bottom of the die.
- pad1 is the merger of layer number 40 and datatype 2 with layer number 41 and datatype 2. The text on it is associated from the same layers.
- pad2 is the merger of layer numbers from 8 to 15 with datatype 0. The text on it is associated with layer numbers from 24 to 39 with datatype 1.
- bump1 is the merger of layer numbers from 2 to 7 with all datatypes and layer numbers from 16 to 23 with all datatypes. It has no associated text.
- bump2 is the merge of layer numbers 40-41 with datatypes from 3 to 7. The text on it is attached from layer number 40 with datatype 3 from only the top cell.

Any text on all layers of the die are from “a_bump” to “a” with the rename_text option and the GNU regular expression “/a_bump/a”.

Example 2

```
die \
-die_name INT \
-layout {-path ./INT.gds} \
-anchor {-name mem -placement 0 0 } \
-anchor {-name log -placement 1000 0 } \
-interposer \
-layer_info {-type pad -name pad1 -layer 40 -top} \
-layer_info {-type pad -name pad2 -layer 41 -top} \
-layer_info {-type route -name route -layer 42 -top} \
-wb_connect pad1 pad2 BY route \
```

The die command defines an interposer die with the name INT. The die has two anchors: mem at (0,0) location, and log at (1000,0).

The die also has three defined layers: pad1, pad2, and route. pad1 and pad2 are of type pad, and route is of type route. All three layers are on the top surface of the die.

pad1 and pad2 are connected via route.

Example 3

The following example demonstrates a LEF/DEF design:

```
die -die_name controller_die \
    -lefdef { \
        -lef { \
            ./inputs/tech.lef \
            ./inputs/controller.lef \
            ./inputs/tsv.lef \
        } \
        -def { \
            ./inputs/controller.def \
        } \
    } \
    -layer_info { \
        -type pad \
        -name TSV \
        -bottom \
        -ext_connect \
    } \
    -layer_info { \
        -type interconnect \
        -name M1 \
    } \
    -layer_info { \
        -type interconnect \
        -name V1 \
    } \
    -layer_info { \
        -type interconnect \
        -name M2 \
    } \
    -layer_info { \
        -type interconnect \
        -name V2 \
    } \
    -layer_info { \
        -type interconnect \
        -name M3 \
    } \
    -wb_connect M1 M2 BY V1 \
    -wb_connect M2 M3 BY V2 \
    -wb_connect M1 TSV
...
...
```

Example 4

This example shows the definition of a die with the name der. The die has three layers defined: pad1, pad2, and merge. pad1 and pad2 are original layers, and merge is a derived layer which is a union of pad1 and pad2.

```
die -die_name der \
    -layout {-path ./der.gds} \
    -layer_info {-type pad -name pad1 -layer 40 -top} \
    -layer_info {-type pad -name pad2 -layer 41 -top} \
    -layer_info {-type merge -name merge \
    -svrf {MERGE (EXTENT DRAWN pad1 pad2)}}}
```

Example 5

The following example demonstrates the anchor usage:

```
die -die_name T2 \
    -layout { -path $env(DESIGN_DIR)/dfm/3di/init/layout_1.gds -type gdsii \
    -primary top }
    -layer_info { -type bump -name 22 -layer 2 } \
    -anchor { -name anch3 -layer 22 -text VSS } \
    -anchor { -name anch4 -placement 200 300 }
```

Example 6

This example demonstrates how to use the -process option instead of the -layer_info argument set.

```
process -name N10 \
    -layer_info {-type pad -name pad1 -layer 40 -top} \
    -layer_info {-type pad -name pad2 -layer 41 -top} \
    -layer_info {-type merge -name merge \
    -svrf {MERGE (EXTENT DRAWN pad1 pad2)}}}
```

After defining a process, you can apply it in the die command as follows:

```
die -die_name INT \
    -layout {-path ./int.gds} \
    -process N10 \
    -anchor { -name mem -placement 0 0 } \
    -anchor { -name log -placement 10 10 } \
    -interposer \
    -import_text_labels texts
```

Example 7

This example demonstrates how to use Tcl variables and environment variables to control options in a rule file.

Read an environment variable that controls an operation that changes depending on a certain flow:

```
set dieName [info exists ::env(DIE_NAME)]
```

Set Tcl variables for the options:

```
set pad_name "bump"
set pad_layer "111:2"
set pad_location "-bottom"
```

Based on the environment variable, apply the die command with different options:

```
if { $dieName == "interposer" } {
    set pad_name "L_PAD"
    set pad_layer "13:2"
    set pad_location "-top"
}

die -die_name $dieName \
... \
    -layer_info {
        -type $pad_name
        -name $pad_name
        -layer $pad_layer
        $pad_location
        -ext_connect
    } \
...
...
```

Related Topics

[config](#)
[process](#)
[stack](#)

-layout

Option for the [die](#) command.

Defines the name of the top-level cell for the 3D assembly.

Note

 If physical precisions differ between layouts, the generated assembly file uses the least common denominator of the precisions of the input layouts.

For example, if two input layouts have precisions of 1/2000 and 1/3000, the output file is generated with a precision of 1/6000.

Usage

```
die ... -layout '{'
  -path layout_path
  [-type {gdsii | oasis}]
  [-primary primary_name]
  [-depth {all | top-only}]
}'
```

Arguments

- **-layout** '{' *arguments* '}'

Required argument and value set that specifies the path, layout system type, and top cell of a layout. If the **-layout** argument set is not used, the **-lefdef** argument set must be applied.

Note, you must enclose the arguments within the {} braces. The *arguments* for **-layout** are described as follows:

- **-path** *layout_path*

Required argument and value set that specifies the path to a GDS or OASIS layout file used for the die. Calibre 3DSTACK treats any input file with the extension .gz or .Z as a compressed file. The gunzip application must be available in your environment.

- **-type** {gdsii | oasis}

Optional argument set that specifies the layout system for the die. The default is gdsii.

- **-primary** *cell*

Optional argument set that specifies the name of a cell in the layout. If this argument set is not specified, the cell is assumed to use the name specified by the **die_name** argument.

- **-depth** {all | top-only}

Optional argument set that specifies whether all of the layout should be read or just the layout that belongs to the specified primary *cell*. The default is that all of the layout geometry is read. Specify -depth top-only to only import geometry that belongs to the specified primary *cell*.

-lefdef

Option for the [die](#) command.

Specifies the paths to LEF and DEF files, if a LEF/DEF layout system is used for the die or component.

Note

 You must enclose the arguments of the -lefdef option in braces {}.

Usage

```
die ... -lefdef '{'  
  {-lef technology_lef [lef_file ...]}  
  {-def def_file}  
'}
```

Arguments

- **-lef *technology_lef* [*lef_file* ...]**

Required argument and value set that specifies the paths to the technology LEF followed by LEF files used for the layout of the die. This argument cannot be specified with **-layout**.

- **-def *def_file***

Required argument and value set that specifies the path to a DEF file that contains the layout for the die. The die positioning options (-placement, -rotate, -flip, and -mag) are ignored if specified; the tool reads this information from the DEF.

Description

The -layer_info -name argument references layers in the LEF/DEF designs by name. The -layer option does not apply to LEF/DEF layouts.

Calibre 3DSTACK automatically translates the LEF/DEF die to OASIS format using the fdi2oasis application. If an error occurs during translation, view the <*topcell*>.calfdi.log file.

-layer_info

Option for the [die](#), [component](#), or [process](#) commands.

Specifies a layer in the die, component, or process. You can specify more than one layer by applying the -layer_info argument set multiple times.

Usage

```
{die | component | process} '{'
    {-layer_info '{' -type layer_type -name layer_name
        {-layer '{' layer_numbers [-depth {all | top-only}] '}'}| {-svrf '{' layer_derivation '}'
            [-show]}
        {[[-text '{' layer_numbers [-depth {number ...}] [-no_update] '}'] ...]
            | [-net_text layer_numbers] }
        [-ext_connect [die_name ...]]
        [-ircx file | -mipt file]
        [-rc_model]
        [-texted]
        [-texttype]
        [-top | -bottom]
        [-via]
        [-virtual]
        [-pex_map layer_name]
        } ...
    '}' ...
}'
```

Arguments

- **-type *layer_type***

Required argument set within -layer_info that specifies the type of layer. The *layer_type* is used in rule or analysis operations and can be any user-defined string. However, it is recommended that *layer_type* values should identify common building blocks of 3D ICs, such as pad, bump, C4, TSV, and RDL layers.

See “[Defining Layers](#)” on page 309 for an example.

- **-name *layer_name***

Required argument set that specifies the name of the layer. The *layer_name* does not have to be unique. The layer name helps you identify the layer in the assembly.

- **-layer '{' *layer_numbers* [-depth {all | top-only}] '}'**

Argument and value set that specifies the layer number, datatype, and depth for the layer. One of -layer or -svrf must be specified for each -layer_info argument. You must enclose the arguments for each layer within the {} braces.

- *layer_numbers* — Required argument with -layer that specifies the layer numbers and datatypes used for the layer in the specified die layout. Multiple layer numbers

and datatypes may be specified as a range or separated by commas; the layers are merged. For example:

```
-layer 2-7,16-23  
-layer 40-41:3-7
```

- -depth {all | top-only} — Optional argument set that specifies whether all of the layer should be read or just the parts of the layer that belong to the specified primary *cell*. The default is that all of the layer geometry is read. Specify -depth top-only to only import geometry on this layer that belongs to the specified primary *cell*.
- -svrf '{' *layer_derivation* '}' [-show]

Argument and value set that specifies a layer derivation using SVRF commands. Either -layer or -svrf must be specified for each -layer_info argument. Only one -svrf argument can be used within a -layer_info argument set.

The -svrf option is similar in function to that of a rule check statement. The *layer_derivation* must consist of at least one standalone layer operation, which is the output layer. If two output layers are specified, they are merged. Local layer definitions can be used in the layer derivation.

Note

 The output layer from the SVRF operation is global for use within other -svrf layer derivations for the same die.

The -svrf argument set has the following option:

- -show — Optional argument that writes derived layers to the generated assembly view of your stack. If this option is not specified, no derived layers are included in the assembly layout view.
- -text '{' *layer_numbers* [-depth {*number* ...}] [-no_update] '}'
Optional argument and value set that specifies the text number, datatype, and depth. You must enclose the arguments for each text layer within the {} braces. It is valid to specify the -text argument set by itself, without a -layer or -svrf argument, but this is not generally considered useful. This argument cannot be specified with -net_text.
 - *layer_numbers* — Required argument that specifies the layer numbers and datatypes used for the text layer in the specified die layout. Multiple layer numbers and datatypes may be specified as a range or separated by commas; the layers are merged. For example:

```
-text 2-7,16-23  
-text 40-41:3-7
```

- -depth {*number* ...} — Optional argument set that specifies the level of design hierarchy from which to retrieve the text labels. If this argument set is not specified, text labels are retrieved from all levels of design hierarchy. You can specify any number of depth numbers, for example “-depth {0 1 2}”.

- -no_update — Optional argument that specifies the layer is not included in the extracted layout netlist. Similar to the -no_update argument for attach_text.

You can specify multiple text layers by repeating the -text option as follows:

```
die ... \
-layer_info {
-type INTERP_via_rdl1
-name INTERP_via_rdl1
-layer 19
-text 19
-text 50
...}
```

- **-net_text *layer_numbers***

Optional argument set that specifies that the text attached to the specified layers represents net names instead of pin names (the text labels of this layer are not interpreted as pins of the die or package). This option allows Calibre 3DSTACK to extract layout net names from the text labels of whitebox connectivity layers and check them against net names in source netlists. This argument cannot be specified with -text.

To perform checking on net layers, apply the -net_mismatch option of the [connected](#) command.

- **-ext_connect [*die_name* ...]**

Optional argument that specifies the die's external interface layer for connectivity purposes. This option is used to explicitly define the off-die interface layers for connectivity tracing. At least one layer on each die must be defined as an external connectivity error to perform connected checks, otherwise the tool issues the following error:

```
3DSTACK_WARNING_1312: No "-ext_connect" defined for [DIE1] and
[DIE2] dies. Check <check_name> will not be performed.
```

The *die_name* argument is optional and specifies an interactive die or component in the package to define the exact die-to-die interaction with this layer. This is particularly useful in the case of multiple dies interacting in different tiers to explicitly define die-to-die connectivity.

You can specify multiple values in a space-separated Tcl list format. If you specify a list of names (for example, -ext_connect {DIE CAP1}, the tool generates connectivity between that particular placement layer and all possible placement layers from the specified dies.

- **-ircx *file***

Optional argument set that specifies the path to a file that contains parasitic information in iRCX format. Use this option with the [export_layout](#) -enable_rc_deck option to automate RC extraction rule generation. If this option is specified in both the die and process commands, the file in the die command takes precedence. Encrypted iRCX files are supported. This option is mutually exclusive with the -mipr option.

- **-texted**

Optional argument set that can only be used with the -lefdef argument set. When specified, the tool explicitly attaches text objects from that layer to the corresponding Calibre 3DSTACK layer. If you do not specify this option, then the text is only attached to layers for which the -text option is applied.

- **-mipt *file***

Optional argument set that specifies the path to a file that contains parasitic information in MIPT format. Use this option with the export_layout -enable_rc_deck option to automate RC extraction rule generation. If this option is specified in both the die and process commands, the file in the die command takes precedence. This option is mutually exclusive with the -ircx option.

- **-texttype**

Optionally defines the derived layer as a TEXTTYPE layer. For example:

```
die -die_name die1 \
    -layout {
        -primary TOP
        -path ./layout.gds
        -type gdsii
    } \
    -layer_info {
        -type M3_text \
        -name M3_text \
        -layer 15 -texttype \
    } \
    -layer_info {
        -type M3_mark \
        -name M3_mark \
        -texttype \
        -svrf { EXPAND TEXT "?" M3_text BY 0.1 } \
        -show \
        -top
    }
```

- **-rc_model**

Optional argument that specifies to generate additional PEX extraction statements for TSVs in the extraction file, as follows:

```
PEX NETLIST 3DIC <placement-layer-name> FILE <PATH> SUBNODE <sub>
PEX 3DIC COUPLING <placement-layer-name> ANALOG MAXDISTANCE <max_distance>
PEX 3DIC COUPLING <placement-layer-name> DIGITAL <frequency> MAXDISTANCE <max_distance>
```

Where the <PATH>, <sub>, <max_distance>, and <frequency> fields must be completed manually after generating the file and are defined as follows:

- PATH — Specifies the path to the netlist.
- sub — Specifies the “sub” node in the TSV sub-circuit.

- max_distance — Specifies the maximum distance (in microns) beyond the extents of the TSV for which the coupling is calculated. Anything beyond the maximum distance is not considered in the coupling calculation.
- frequency — Specifies the frequency in Hz.
- -top
Optional argument that indicates that the layer is physically placed on the top side of the die. This argument is needed to correctly connect the interface layers in the stack.
- -bottom
Optional argument that indicates that the layer is physically placed on the bottom side of the die. This argument is needed to correctly connect the interface layers in the stack.

Note

 If the -top or -bottom keywords are not specified and the die has more than one layer, then the layer is considered internal to the die and cannot be identified as an interacting layer. As a result, the layer cannot be used in checks with two layer types.

If the die only has one layer, the layer is automatically treated as interacting and you do not need to specify either -top or -bottom.

- -via
Optional argument that specifies that the layer is a via layer. This is used for annotated GDS generation when creating the extraction rule file.
The extraction engine requires certain layers to be defined as conductors in the generated extraction rule file. The conductors are defined using the BY keyword of the Connect statement. For example, in this statement:

```
CONNECT A C BY B
```

the B layer is a conductor. The -via option controls the layers that serve as conductors when the Connect statements are generated in the extraction rule file. Port Layer statements are not generated for conductors.

- -virtual
Optional argument that specifies that an interposer layer can be involved in connectivity checking with neighboring dies, even when the layer does not directly interact with them. In other words, this option allows you to define checks for “inner” layers of the dies that are not marked as interacting with the -bottom or -top options. The -virtual option can only be used if the die uses the -interposer option. The layer must be part of the white-box connectivity of the die (see the -wb_connect option for more details).

If you specify the -top, -bottom, or -virtual options for the layer, then the layer can participate in geometrical checks.

The specified virtual layers in an interposer can be checked with the following layers:

- Top layers of a die in the bottom tier of the interposer.

- Bottom layers of a die in the top tier of the interposer.

As an example, this option allows you to check connectivity between C4 bumps on one die and BGA pads on an interposer die by marking the BGA layer with the -virtual option.

- `-pex_map layer_name`

Optional argument set that maps the current physical layer to a calibrated layer name. The *layer_name* value must not contain spaces and can be used in different layer definitions.

Calibre 3DSTACK uses this information to write an SVRF file when exporting an annotated GDS file with the config -export_connectivity command.

The SVRF file is named *<gds_name>.map.svrf*, where *gds_name* is the name you specified for the annotated GDS. The calibrated *layer_name* and assembly layer pairings are written to the generated SVRF file with [PEX Map](#) statements.

Description

The -layer_info -name argument references layers in the LEF/DEF designs by name. The -layer option does not apply to LEF/DEF layouts.

Calibre 3DSTACK automatically translates the LEF/DEF die to OASIS format using the fdi2oasis application. If an error occurs during translation, view the *<topcell>.calfdi.log* file.

component

Optional in 3DSTACK+ extended syntax file.

Defines a single component in the assembly. A component can be an object, such as a resistor, capacitor, or inductor, but it cannot be a die.

Usage

```
component -component_name
  {-layout '{' arguments '}' | -lefdef '{' arguments '}'}
  [-thickness value]
  [-process process_name]
  {-layer_info '{' arguments '}' } ...
  [-anchor '{' -name anchor_name {-placement x_offset y_offset
    | -layer number -text label}'}'] ...
  [-interposer | -package | -lamine | -substrate]
  [-rename_text {"expression" ... | {-file file}}]
  [-import_text_labels file
    [-order column_list]
    [-xsi {NET_NAME | PIN_NUMBER | PIN_NAME}]
    [-attach_to {-layer layer_name -component component_name} ...]
    [-top_level_coords]]
  [-source_netlist spice_file]
  [-wb_connect layer1_name layer2_name [BY layer3_name] ] ...
  [-swappable pin_list]
```

Arguments

- **-component_name name**

Required argument that specifies the beginning of a component definition.

- **-layout '{' arguments '}'**

Required argument and value set that specifies the path, layout system type, and top cell of a layout. See “[-layout](#)” on page 82 for details.

- **-lefdef '{' arguments '}'**

Argument and value set that specifies the paths to LEF and DEF files, if a LEF/DEF layout system is used for the component. See “[-lefdef](#)” on page 83 for details.

- **-thickness value**

Optional argument set that defines the thickness of the die in microns. Calibre 3DSTACK uses the thickness values to generate a cross-sectional assembly view of the stack. See “[Assembly Views](#)” on page 35 for details.

If this option is not specified, you can use the

CALIBRE_3DSTACK_DEFAULT_THICKNESS environment variable to set a global thickness for all dies. If neither the environment variable nor the -thickness option is

specified, the thickness of the placement level is calculated based on the minimum width and height of the dies in the same level.

- -process *process_name*

Optional keyword and name for process layer information defined with the “[process](#)” command.

The process layer information is usually provided by the foundry that manufactures your IC. The advantage of applying this command is that you do not have to redefine layer information for each die or component in the 3D-IC. Instead, you can refer to a process that is already defined in the component command.

Note

 If you specify layer information in addition to a process in the component command, the layer information overwrites the process specification if the same information is defined in the process. If items are not defined in the layer information, the process data is merged with the layer information.

- **-layer_info '{' *arguments* '}'**

Required argument and value set that specifies a layer in the component. You can specify more than one layer by applying the -layer_info argument set multiple times. Note that you must enclose the arguments for each layer within the {} braces.

See “[-layer_info](#)” on page 84 for details.

- **-anchor '{'-name *anchor_name* {-placement *x_offset* *y_offset* | -layer *number* -text *label*} '}'**
Optional argument and value set that creates an anchor with a specified name at an x and y coordinate specified in microns. You can specify multiple anchors per component. The anchor is a labeled coordinate that can be referenced in assembly operations for alignment purposes. The options are defined as follows:

- name *anchor_name* — specifies the name of the anchor.
- placement *x_offset* *y_offset* — specifies the coordinates of the anchor in microns.
- layer *number* — specifies the anchor layer.
- text *label* — specifies the label name to identify the anchor location on the layer.

- **-interposer | -package | -laminate | -substrate**

Optional arguments that specify that the layout serves the purpose of an interposer. The -interposer, -package, -laminate, and -substrate arguments are interchangeable; they are all equivalent to specifying -interposer. The different names serve as a label that more closely identifies the component in your package. Apply one of these options if you want to generate a hierarchical netlist or create specific rules for an interposer. You can apply this option more than once in your assembly. If you specify more than one interposer, Calibre 3DSTACK generates a flat assembly netlist.

- `-rename_text {"expression" ... | {-file file}}`

Optional argument set that renames text in the die layout. The expression is a GNU regular expression. Multiple expressions are supported, but the entire set of *expressions* must be enclosed in quotes or braces. For example:

```
-rename_text "/</\[/ />/\]/"
```

Instead of in-line expressions, you can specify the path to a file that contains a list of expressions. The file contains one expression per line. For example:

```
/A/B/  
/A.*B/R/  
/^VDD.*/VCC/  
/A/B/g
```

- `-import_text_labels file {xsi_arguments}`

Optional argument set that specifies a text label file. See “[import_text_labels](#)” on page 402 for more details. If you specify an Xpedition Substrate Integrator® (XSI) netlist as the *file*, then the tool uses this file to generate and attach text labels in the layout. The following *xsi_arguments* only apply to XSI files:

`-order column_list`

Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF_DES, PIN_NUMBER, PIN_X, PIN_Y, PIN_NAME, NET_NAME, COMPONENT_NAME, and IGNORE. The default order is as follows:

```
NET_NAME IGNORE IGNORE COMPONENT_NAME PIN_NUMBER REF_DES
```

`-xsi {NET_NAME | PIN_NUMBER | PIN_NAME}`

Optionally specifies the column in the XSI netlist from which to retrieve text labels. The default is the NET_NAME column.

`-attach_to {-layer layer_name -component component_name} ...`

Specifies the layer number and datatype of the layout to which the labels are attached. The -component *component_name* argument set specifies to generate text labels for the given component in the assembly. The following example demonstrates text labeling for a XSI netlist.

```
die -die_name top -interposer \  
-lefdef {...} \  
-import_text_labels {./labels.csv -xsi PIN_NAME \  
-order { PIN_NAME NET_NAME IGNORE COMPONENT_NAME PIN_NUMBER \  
REF_DES PIN_X PIN_Y } \  
-attach_to {-layer bmp -component L1} \  
-attach_to {-layer p -component L2} \  
-attach_to {-layer ub -component A} } \  
...
```

`-top_level_coords`

Specifies that the coordinates for text labels in the *file* belong to the top level.

- **-source_netlist *spice_file***

Optional keyword and path to a SPICE netlist that contains the subcircuit definition for the specified layout. If you specify this option, Calibre 3DSTACK automatically calls the System Netlist Generator tool to build a complete netlist during a verification run.

- **-wb_connect *layer1_name* *layer2_name* [BY *layer3_name*]**

Optional argument set that defines the connectivity of the materials inside of the component. This is only required for hierarchical (white-box) connectivity extraction of the assembly.

- **-swappable *pin_list***

Optional argument set that specifies a list of pins that are physically equivalent. For example, in the case of a two pin resistor, both of the pins are considered equivalent and can be marked as swappable for LVS purposes.

Description

The component command usage and syntax is the same as the [die](#) command. Use this command to define items in your assembly that are part of the connectivity or assembly of the stack, but are not dies. The only argument unique to this command is the -swappable option.

Examples

Define a new component.

```
component \
    -component_name RESISTOR \
    -layout {-path ./resistor.gds} \
    -layer_info {-type pad -name R1 -layer 40 -top} \
    -layer_info {-type pad -name R2 -layer 41 -top} \
    -layer_info {-type route -name res_layer -layer 42 -top} \
    -wb_connect pad1 pad2 BY res_layer \
    -swappable {r1 r2}
```

stack

Required in 3DSTACK+ extended syntax file.

Defines the locations of objects in the assembly.

Usage

stack

```
-stack_name stack_name
{die_stack | tier_spec | stack_ref}...
```

die_stack Usage

```
{ -die | -component | -package | -laminate | -substrate } '{'
    -name name
    [-anchor anchor_name from_anchor_name to_anchor_name | -placement x y]
    [-mag factor] [-rotate angle] [-flip {x | y}] [-invert]
    [-rename_text {"expression" ... | {-file file} }]
    [-ignore_pin {"expression" ... | {-file file} }]
    [-source source_name]
}' [-z_origin value]
```

tier_spec Usage

```
-tier '{'
    {die_stack | stack_ref}...
'}
```

stack_ref Usage

```
-stack stack_name
```

Arguments

- **-stack_name stack_name**

Required argument set that specifies a stack name used in rule check commands. If this command is specified multiple times, each **stack_name** value must be unique.

- **-die | -component | -package | -laminate | -substrate {' arguments '}**

Argument set that specifies a placement of a die, component, package, laminate, or substrate. The *arguments* are described as follows:

- **-name name**

Required argument and value set that specifies a placement. The object must be defined in the file.

- **-anchor anchor_name from_anchor_name to_anchor_name**

Optional argument and value set that specifies the anchoring information.
anchor_name specifies the name of the object already being placed in the stack.
from_anchor_name specifies the name of an anchor defined for the *anchor_name*

object. *to_anchor_name* specifies the name of an anchor defined for the current object *name*. -anchor and -placement are mutually exclusive. If both options are not specified, the object is placed at the location of the last specified die.

- -placement *x y*

Optional argument and value set that specifies the location to place the object. The *location* is defined by *x* and *y* coordinates (*x y*) and should be specified in microns. -anchor and -placement are mutually exclusive. If both options are not specified, the object is placed at the location of the last specified object. If there is no previously placed object, the default placement location is (0 0).

- -mag *factor*

Optional argument and value set that specifies a magnification factor by which to expand or shrink the placement. Coordinate data in the object is multiplied by the specified *factor*.

Caution

 If you specify -mag, geometries can shift by 1 dbu due to grid snapping. This can affect the rule check results. See “[Magnification](#)” on page 98 for more details.

- -rotate *angle*

Optional argument and value set that specifies the rotation angle of the current placement. The *angle* must be an integer between 0 and 360.

- -flip {*x* | *y*}

Optional argument and value set that specifies the axis around which the current placement is flipped. If you are viewing the layout from above, flipping the layout around the *y* axis means that the shapes on the left side are moved to the right side, and the shapes on the right side are moved to the left side. Flipping around the *x* axis means that the shapes on the top side are moved to the bottom side and the shapes on the bottom side are moved to the top side.

The -flip option does not affect the -top and -bottom layer definitions for the die, but it does affect the die placement coordinates.

- -invert

Optional argument that indicates that the layers with the top connections become bottom connections, and vice versa in the current placement. In this case, if viewing the layout from the side (cross-section view), the layers on the top are moved to the bottom and the layers on the bottom are moved to the top. The die placement coordinates do not change, but the -top and -bottom layer definitions are affected.

- -rename_text {"*expression*"... | {-file *file*} }

Optional argument and value set that specifies the rule for text renaming. The *expression* is a GNU regular expression.

Instead of in-line expressions, you can specify the path to a file that contains a list of expressions. The file contains one expression per line. For example:

```
/A/B/  
/A.*B/R/  
/^VDD.*/VCC/  
/A/B/g
```

- -ignore_pin {"*expression*"... | {-file *file*} }

Optional argument and value set that specifies an expression that matches one or more pin names to be ignored in layout extraction. Corresponding ports in the source netlist that match the ignored layout pins are also ignored. The expression is a case-insensitive GNU regular expression.

Instead of in-line expressions, you can specify the path to a file that contains a list of expressions for pins to ignore. The file contains one pin or expression per line, for example:

```
VSS  
VDD  
^A1$  
A2
```

The *expression* and -file options are mutually exclusive.

Note

 To list all successfully ignored pins in the report, include “-report_ignored_pins YES” in the config -report argument set. See “[-report](#)” on page 111 for details.

- -source *source_name*

Optional argument and value set that specifies the corresponding source placement of a cell, similar to the [map_placement](#) command in standard Calibre 3DSTACK.

Hierarchical names are supported if the source netlist is hierarchical; use the forward slash (/) as the hierarchy separator. For example:

- A source netlist has a top cell named 3DSTACK_TOP that contains one placement, named INTERPOSER, of the sub-circuit INTERPOSER_CHIP.
- The INTERPOSER_CHIP has two placements, named DIE1 and DIE2.

In order to map to DIE1, you specify the hierarchical *source_name* as INTERPOSER/DIE1.

- **-tier** '{' {*die_stack* | *stack_ref*}... '}'

Argument and value set that specifies a tier consisting of dies and stacks placed at the same z coordinate. A tier provides a method to place elements side-by-side. Additional connectivity checking is performed between the die placements within the same tier.

- **-stack** *stack_name*

Argument and value set that specifies the name of a stack defined in the Calibre 3DSTACK+ rule file. The same stack cannot be referenced more than once.

- **-z_origin** *value*

Optional argument set that specifies the height of the die from the bottom of the stack, where the *value* is a distance in microns. This option enables the generation of a cross-section assembly view. See “[Assembly Views](#)” on page 35 for details.

Description

The stack statement defines the assembly operations that construct your 3D package. It defines the locations of die placements and interface layers that connect die placements. The stack represents any set of vertical alignments. A true 3D stack may consist of multiple object placements. In a 2.5D interposer approach, each stack consists of a single die placement and the interposer. It is also possible to have an interposer with multiple stacks, each consisting of more than one die (or equivalent object) placement.

For Calibre 3DSTACK, all stack definitions are combined into a single assembly.

At least one die, stack, or tier must be specified. The elements can be specified multiple times. The order of stack arguments determines the stack’s structure. The order in the stack is from bottom to top and does not always indicate whether die placements interact with one another. Each subsequent element in the stack is placed in a layer with the z coordinate greater than the z coordinate of the previous element.

The -tier option is used to group elements side by side—all elements in a tier are at the same z location. The -tier option has special meaning during connectivity checking (see “[connected](#)” on page 147 for more details).

Note

 Connectivity between die layers is determined implicitly by the layer definitions and die locations. If you need to explicitly define connectivity, use the [connect](#) command.

Placement Naming Conventions

The generated placement names use the following format (the <> brackets are not part of the format):

```
<stack-name>_tier<tier-index>_<die-name><placement-id>
```

where:

- *stack-name* is the name of the stack.
- *tier-index* is a generated number for the tier's position in the stack. The numbering starts from 1 and increases by 1 for each tier from the bottom to the top of the stack.
- *die-name* is the name of the die.
- *placement-id* is an identification number for each unique tier and die pair.

Magnification

Specifying the -mag option applies a magnification factor to the placed layout. Depending on the geometry, layout precision, and applied magnification factor, this can lead to unintended consequences caused by grid snapping. During the magnification process, the coordinate values are rounded up or down to the nearest database unit. After magnification, the actual placement of cells may be affected by snapping, causing gaps and jogs. This may not be noticeable until you run certain rule checks.

For example, assume two layouts are placed on top of each other using the stack command and one of the layouts has an applied magnification factor. You might have an overlap rule that checks for sufficient layer coverage between the pads. You also might have a centers check that verifies that the center points of each pad are aligned. When running a verification, the overlap check returns no results, but the centers check returns hundreds or thousands of errors. The root cause of the errors may be difficult to diagnose until you compare the original layout to the layout in the assembly. In this example, the magnified pad geometry can shift by 1 dbu in different directions due to rounding in grid snapping, which affects the alignment and the extent calculations.

Examples

```
stack
-stack_name w_tier
-tier {
    -die { -name RAM -mag 1.1 -flip x -placement 0 0}
    -die { -name RAM -mag 1.1 -rotate 90 -flip y -placement 0 750}
    -die { -name RAM -mag 1.1 -rotate 180 -placement 750 0}
    -die { -name RAM -mag 1.1 -rotate 270 -placement 750 750}
}
-die { -name INT -invert -rotate 90}
-tier {
    -die { -name MEM -anchor INT anchor_MEMORY1 anchor_INT -source pRam1}
    -die { -name MEM -anchor INT anchor_MEMORY2 anchor_INT -source pRam2}
    -die { -name MEM -anchor INT anchor_MEMORY3 anchor_INT -source pRam3}
}
-die { -name MEM -placement 100 100}
```

The preceding example defines a stack named w_tier with the following die placements (for simplicity, the height of each die placement is 100):

- The first four dies form a tier:
 - RAM die placement at position (0, 0, 0) magnified by 1.1 factor and flipped across the x axis.
 - RAM die placement at position (0, 750, 0) magnified by 1.1 factor, rotated 90 degrees and flipped across the y axis.
 - RAM die placement at position (750, 0, 0) magnified by 1.1 factor and rotated 180 degrees.
 - RAM die placement at position (750, 750, 0) magnified by 1.1 factor and rotated 270 degrees.
- INT die placement at position (0, 0, 100) rotated 90 degrees and inverted top and bottom layers of it.
- The next three dies form another tier and the -source option is used:
 - MEM die placement at position (0, 0, 200) calculated by anchor definition.
 - MEM die placement at position (0, 500, 200) calculated by anchor definition.
 - MEM die placement at position (0, 1500, 200) calculated by anchor definition.
- MEM die placement at position (100, 100, 300).

Ignoring Pins

You can also specify to ignore certain pins as follows:

```
stack -die { -name die -ignore_pin "VSS" }
```

Related Topics

[config](#)

[die](#)

[process](#)

connect

Optional in 3DSTACK+ extended syntax file.

Defines explicit connectivity between die layers in a stack. This enables you to manually define connectivity between layers instead of using only the implicit connectivity provided by the physical assembly.

Usage

connect

```
-stack stack_name
{-die {'die_name -layer die_layer_name'}}
{-die {'die_name -layer die_layer_name'}}
[-die {'die_name -layer die_layer_name''} ...]
```

Arguments

- **-stack *stack_name***

Specifies the name of a stack that has been defined in the [stack](#) command.

- **-die {'*die_name*-layer *die_layer_name*'}**

Specifies a die name and a layer on that die to connect. The -die and -layer arguments must be specified together. The first specified -die and -layer argument set is connected to any number of following -die and -layer pairs. You must specify at least two sets of -die and -layer pairs. The arguments are defined as follows:

die_name — Specifies a die that belongs to the named stack.

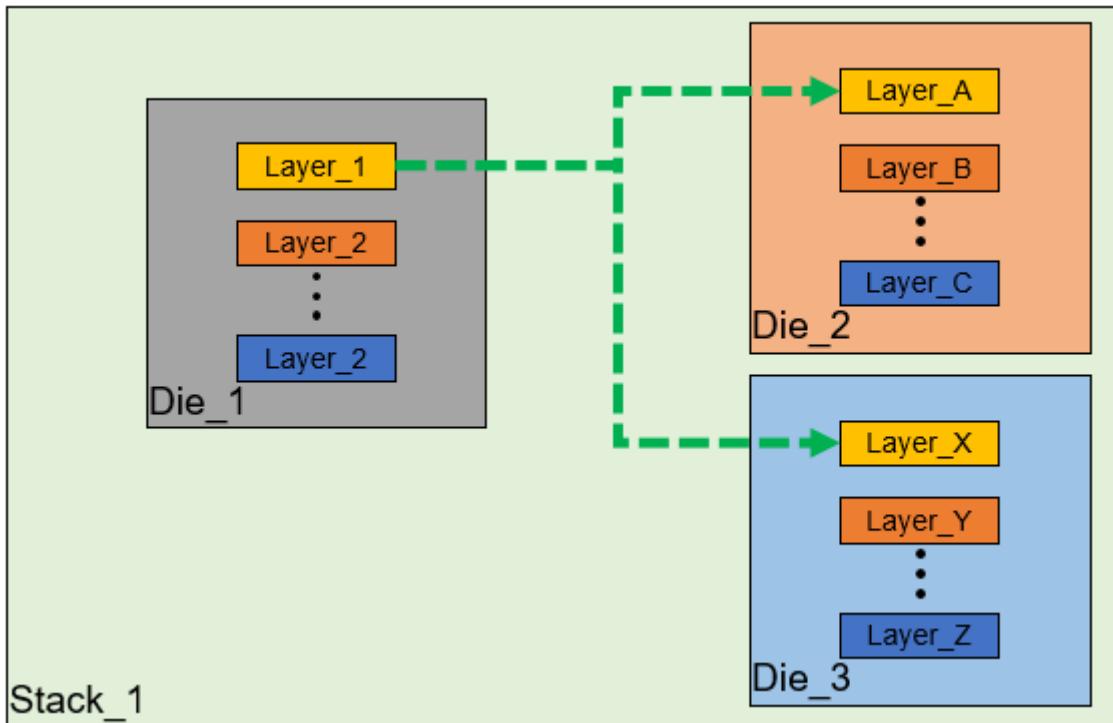
-layer *die_layer_name* — Specifies a single layer in the die.

Examples

The following example demonstrates how to define explicit layer connectivity between a single layer that is present in three dies within the same stack.

```
stack -stack_name Stack_1 \
{
    -die {-name Die_1 -placement 0 0 -source Die1} \
    -die {-name Die_2 -placement 0 0 -source Die2} \
    -die {-name Die_3 -placement 0 0 -source Die3} \
}

connect -stack Stack_1 \
-die Die_1 -layer Layer_1 \
-die Die_2 -layer Layer_A \
-die Die_3 -layer Layer_X
```



config

Specifies configuration information for the 3DSTACK+ syntax format rule file. The top cell name, source netlist, report file, connectivity, and other information can be specified.

Usage

config

```
[ -layout_primary name ]
[ -layout_case {no | yes} ]
[ -layer_props_file props_file ]
[ -source_netlist '{' -file file_name -format {SPICE | MGC | CSV | VERILOG
    [-version {2001 | 1995}] } [-case {YES | NO}]
    [-hier [-wrap interposer_name]] [-order {column_list}] [-subckt_pins {pin_type}]
    [-apply_bboxing cell_list] '}' ]
[ -report '{' -file report_path [-max_results value] [-child_rdbs {NO | YES}]
    [-report_ignored_pins {NO | YES}] '}']
[ -ignore_trailing_chars char_list ]
[ -export_connectivity '{' -file output_file [-format {VERILOG | AIF | MGC | SPICE
    | XSI}]} [-hier [-no_top]] [-pkg package_name] '}' ]...
[ -net_map "-nets list_of_nets -to net_name" ]
[ -import_net_map -file filename ]
[ -pin_map "-pins list_of_pins -to pin_name" ]
[ -import_pin_map -file filename ]
[ -set_auto_rve_show_layers {NO | YES} ]
[ -set_rve_cto_file -set_rve_cto_file cto_file ]
[ -svrf_specs -svrf_specs specification_file ]
[ -units [-distance {um | mm | nm}] [-power {W | mW | uW}] [-time {s | hr | min | ms | us}]]
[ -warning_severity -warning {message_id | *} -severity {0 | 1 | 2} [-count count]]
```

Arguments

The descriptions for the config command options are contained in each of the linked topics.

Description

This command provides a way to include the configuration information provided with certain standard Calibre 3DSTACK commands in the 3DSTACK+ extended syntax rule file. You can use the [3dstack_block](#) command to include standard Calibre 3DSTACK commands that are not specified with the config command.

Examples

The example specifies the name of the top cell for the 3D assembly as BGA, and generates the file *3dstack.report* containing a report of the run. It also specifies to import connectivity information hierarchically from *netlist.spice*.

```
config -layout_primary BGA \
    -source_netlist {
        -file netlist.spice
        -format SPICE -hier
    } \
    -export_connectivity {
        -file layout.spice
        -format SPICE
    } \
    -report {
        -file 3dstack.report
    } \
    -units {-distance nm}
```

-layout_primary

Option for the **config** command.

Defines the name of the top-level cell for the 3D assembly.

Usage

```
config other_arguments
    -layout_primary name
```

Arguments

- ***name***

Required value that specifies the name of the top-level cell for the 3D assembly. The default name is “TOPCELL_3DI” if this command does not appear in the chip stack rule file.

Examples

```
config ... layout_primary TOPCELL_3DSTACK
```

-layer_props_file

Option for the [config](#) command.

Specifies a layer properties file for Calibre DESIGNrev that sets custom layer visibility options.

Usage

```
config other_arguments
    -layer_props_file props_file
```

Arguments

- *props_file*

Required path to a layer properties file. See the “Description” section for details.

Description

The specified *props_file* defines the properties of the layers used in Calibre DESIGNrev and follows this format:

```
layer_name layer_color layer_fill layer_visibility layer_width
```

where:

- *layer_name* — is an alphanumeric string specifying the name of the layer.
- *layer_color* — is any valid Tcl color name.
- *layer_fill* — is one of the following fill types:

clear	diagonal_1	diagonal_2
wave	brick	circles
speckle	light_speckle	solid

- *layer_visibility* — Specifies whether the layer is visible. The *layer_visibility* must be one of the following:
 - 0 — the layer is not visible.
 - 1 — the layer is visible.
- *layer_width* — an integer greater than 1 that specifies the line width of the polygon outline on that layer.

Examples

```
config ... -layer_props_file assembly.layerprops
```

where *assembly.layerprops* is written as follows:

```
controller1 blue speckle 1 1  
ram_1 yellow diagonal_1 1 1  
ram_2 red wave 1 1  
...
```

-layout_case

Option for the [config](#) command.

Controls the case-sensitivity for the pins in layout. This option also controls case-sensitivity for pin mapping.

Note

 The config -source_netlist case option controls the case-sensitivity for the pins in the source.

Usage

```
config other_arguments
    -layout_case {no | yes}
```

Arguments

- no | yes

Optional value that controls the case-sensitivity handling of the assembly layout. If you specify yes, the layout net and pin names are handled in case-sensitive manner. The equivalent standard syntax command is layout_case. The default is no.

Examples

```
config ... -layout_case yes
```

-source_netlist

Option for the [config](#) command.

Imports a source netlist file used for connectivity comparison. It is highly recommended that you use a source netlist during the verification of your assembly.

Note



The config -layout_case option controls the case-sensitivity for the pins in the layout.

Usage

config other_arguments

```
-source_netlist -file file_name
-format {SPICE | MGC | CSV | {VERILOG [-version {2001 | 1995}]} }
[-apply_bboxing cell_list]
[-case {NO | YES}]
[-hier]
[-order {column_list}]
[-subckt_pins {NET_NAME | PIN_NUMBER | PIN_NAME}]
[-wrap interposer_name]
```

Arguments

- **-file *file_name***

Required argument and value set that specifies the name of the source netlist file.

- **-format {SPICE | MGC | CSV | {VERILOG [-version {2001 | 1995}]} }**

Required argument and value set that specifies the format of the input file.

The MGC format contains a list of add_connection directives that describe single connections in the 3D assembly:

```
add_connection -connection1 {placement1 cell1 net1 pin_instance_name1} \
-connection2 {placement2 cell2 net2 pin_instance_name2}
```

If you specify to read a Verilog netlist, apply the -version argument to specify the version of the input netlist. The default version is Verilog 2001.

The CSV keyword supports comma-separated value input files in AIF, XSI, or spreadsheet format. Calibre 3DSTACK also writes a SPICE file with the extension *.spi* to your working directory for debugging purposes. See “[AIF Converter Reference](#)” on page 278 and “[Spreadsheet Converters](#)” on page 285 for details on these files.

- **-apply_bboxing *cell_list***

Optional argument set that automatically enables black boxes for specified cells in the source netlist. The tool internally generates LVS Box SVRF statements for specified cells for use in LVS checks and the SPICE netlist import.

- **-case {NO | YES}**
Optional argument and value set that controls whether the source netlist file is handled as case-sensitive or not. Note that this option only applies to the source netlist file. The default value is NO (the file is handled regardless of case).
- **-hier**
Optional argument that specifies the source netlist is hierarchical. The source netlist format must be SPICE if -hier is specified.
- **-order *column_list***
Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF_DES, PIN_NUMBER, PIN_X, PIN_Y, PIN_NAME, and NET_NAME. This option can only be specified with the CSV keyword.
- **-subckt_pins {NET_NAME | PIN_NUMBER | PIN_NAME}**
Optional argument set that specifies the type of information specified for the pin column of the CSV file. The default is NET_NAME. This option can only be specified with the CSV keyword.
- **-wrap *interposer_name***
Optional argument that specifies to automatically wrap the specified interposer subcircuit with a top-level circuit for use in Calibre 3DSTACK. This option only applies to source netlists of format SPICE. When you specify -wrap, Calibre 3DSTACK does the following at runtime:
 - a. Creates a new source netlist file in your run directory named <*source_netlist_file*>.wrapped. The new file contains the following:
 - An include statement for the specified source netlist file.
 - A top subcircuit definition named the same as the config -layout_primary option or a default if no value is given.
 - A placement inside the top subcircuit for the specified *interposer_name* and its nets.
 - b. Automatically changes placement mapping as follows:
 - Ignores the interposer placement mapping and issues a warning message.
 - Adds the interposer placement name to mapped placements.

Description

Imports connectivity information from the specified *file_name*.

If there are connected checks specified in the chip stack rule file, the imported connectivity information is checked against the netlist information extracted from the chip stack. If your source netlist file is case-sensitive, specify the -case YES option.

Specify the -hier option if the source netlist is hierarchical. The standard syntax `source_filter` command is not supported when the -hier argument is used. See the [sng](#) command for information on generating a hierarchical source netlist.

You can create a source netlist using the System Netlist Generator. See “[System Netlist Generator Flow Example](#)” on page 327.

-report

Option for the [config](#) command.

Generates a report file that contains information about the verification run. This is useful for debugging the design.

Usage

```
config other_arguments
    -report -file report_file [-max_results value] [-child_rdbs {NO | YES}]
        [-report_ignored_pins {NO | YES}]
```

Arguments

- **-file *report_file***

Required argument set that specifies the name of the output file.

- **-max_results *value***

Optional argument set that limits the number of reported results (per check) to the number specified with the *value* keyword. The max_results argument applies to each check in the rule file. The *value* keyword must be a positive integer, unless “all” is specified. If *value* is set to “all”, no limitation is applied to the report command. The default is 50.

Note

 If you do not specify the report command in your rule file, then there is no limit to the number of results written to the results database.

- **-child_rdbs {NO | YES}**

Optional argument set that specifies whether to generate results databases for each chip in the stack. Specify yes to generate child RDBs for all chips in the design. The default is NO.

- **-report_ignored_pins {NO | YES}**

Optional argument set that specifies to list all ignored pins in the report. Ignored pins are specified in the [stack](#) command. The default is not to report any ignored pins. The following is an example of an ignored pin in the report:

```
IGNORE PIN STATEMENTS:
    Chip Placement: assembly_tier1_interposer1
    Expression: {*clk*}
    Matched Pins:
        controller_clk
        clk
```

Description

Generates a file containing a report of the run. The format of the report is similar to the reports for Calibre nmLVS and Calibre PERC runs. Refer to [Report File Format](#) for an example.

The following sections are only generated when a source netlist is specified in the chip stack rule file:

- Layout pads missing source ports.
- Source ports missing layout pads.
- Incorrect nets.
- Connectivity rulecheck results.

Errors are reported only for [connected](#) checks. Errors for other verification commands ([internal](#), [external](#), and [copy](#)) are not included in the report.

Examples

Use the following command to generate a report file with no reported result limitations:

```
config ... -report -file report.txt -max_results all
```

Use the following command to limit the results for each rule check to a maximum of 30:

```
config ... -report -file report.txt -max_results 30 -child_rdbs yes
```

-ignore_trailing_chars

Option for the `config` command.

Removes specified characters from the end of text labels.

Usage

```
config other_arguments
    -ignore_trailing_chars {character...}
```

Arguments

- *character...*

Required list of characters to remove from the end of all text labels.

Description

Removes up to one character from the end of text labels during connectivity extraction. Trailing characters, such as “:”, are sometimes used on text labels to indicate implied connections between polygons that are not physically connected.

In the 3DSTACK assembly, it is possible for placement labels to inherit trailing characters. For example, the VSS and CLK nets may also be VSS: and CLK:. These are interpreted as different nets, even though they are the same.

Use this command to ignore trailing characters so that the nets are connected during connectivity comparison.

Examples

The following text label pairs are used in a design:

Table 3-4. Nets With Trailing Characters In Design

Net	Standard Text Label	Label With Implied Connectivity
VDD	VDD	VDD:
VSS	VSS	VSS.
CLK	CLK	CLK::
ADR	ADR	ADR,

Use the following command to merge the desired nets:

```
config ... -ignore_trailing_chars ": , "
```

The following nets are extracted during connectivity analysis as a result of the command:

```
VDD VSS CLK CLK: ADR
```

Note, that only the last character (“：“) was removed from the CLK net. Even though that character is specified, Calibre 3DSTACK only ignores the last character in the string, so the CLK and CLK: text labels are still considered different nets.

-export_connectivity

Option for the [config](#) command.

Exports the connectivity information into a netlist file.

Usage

```
config other_arguments
    -export_connectivity -file file_name
        [-format {VERILOG | AIF | MGC | SPICE | XSI}]
        [{-property number} | -text}] [-flat]
        [-hier [-no_top]]
        [-pkg package_name]]
```

Arguments

- **-file *file_name***

Required argument and value set that specifies the name of the output file.

- **-format {VERILOG | AIF | MGC | SPICE | XSI}**

Optional argument set that specifies the format for the output connectivity file (the default is Verilog).

Note



Use the [export_layout](#) command to write out annotated GDS files.

- **-hier**

Optional argument that specifies to generate a hierarchical netlist for 2.5D ICs. This option only works with -format SPICE. You must also specify die -interposer for one of the imported chips.

When the -interposer argument is applied to a layout and the -hier option is used, the generated netlist instantiates all dies within the layout instance specified by -interposer. The top cell of the assembly only contains the instantiated interposer die. The net names are generated from the interposer pin names.

- **-no_top**

Optional argument that specifies not to create a new top cell for the entire design. This option requires -hier to be specified.

- **-pkg *package_name***

Optional argument that specifies a package layout in the design. The layout must not be an interposer. The specified *package_name* must have at least one layer defined in the connectivity stack. This option requires -hier to be specified.

Description

Exports connectivity to a file in the specified format (Verilog, AIF, SPICE, MGC, or XSI).

The MGC format contains a list of add_connection directives that describe single connections in the 3D assembly:

```
add_connection -connection1 {placement1 cell1 net1 pin_instance_name1}  
-connection2 {placement2 cell2 net2 pin_instance_name2}
```

Warnings are issued for single-port connections (nets connected to one port only) in the extracted netlist.

Caution

 There should not be spaces in cell or pin names as this causes the exported netlist to be incorrect. For pins, the tool creates multiple pins in such a case. The export_connectivity command issues a warning for these cases.

AIF Format

See “[AIF Export File Format](#)” on page 280 for details on the AIF netlist exported by Calibre 3DSTACK.

XSI Format

Calibre 3DSTACK generates the XSI CSV file using the pins from all placements in the following columns:

Signal Name, Instance Level Name, Design Name, Component Name, Pin Number, Ref Des

The following is an example.

```
685,685,TOPCELL_3DI,assembly1_tier1_BGA1,BS_SW1_GPIO_40,BGA
```

Examples

This example demonstrates the config -export_connectivity -hier option. The two netlists are shown side-by-side for comparison:

Table 3-5. config -export_connectivity -hier

3dstack_hier.sp	3dstack_no_hier.sp
<pre>.SUBCKT TOP_CELL XpInterposer ... interposer .ENDS TOP_CELL .SUBCKT controllerENDS controller .SUBCKT ramENDS ram .SUBCKT interposer ... XpController ... controller XpRam0 ... ram XpRam1 ... ram .ENDS interposer</pre>	<pre>.SUBCKT TOP_CELL XpInterposer ... interposer XpController ... controller XpRam0 ... ram XpRam1 ... ram .ENDS TOP_CELL .SUBCKT controllerENDS controller .SUBCKT ramENDS ram .SUBCKT interposerENDS interposer</pre>

Without -hier, all chips in the assembly are instantiated in the TOP_CELL. With -hier, the chips that are not interposers are instantiated within the interposer.

-net_map

Option for the [config](#) command.

Specifies nets that can use a different name during connectivity checking. This command is useful for waiving known connectivity errors by adjusting net names at runtime.

Usage

```
config other_arguments
    -net_map -nets list_of_nets -to net_name
```

Arguments

- **-nets *list_of_nets***
Required list of nets to rename.
- **-to *net_name***
Required name for the list of specified nets. All nets in the *list_of_nets* argument are renamed to the specified value.

Description

The net mapping operation only occurs in memory during connectivity checking. Net mapping is written to the Calibre 3DSTACK report file.

Multiple net mapping statements are allowed (many-to-one):

```
config ... -net_map -nets {VDD_a} -to VDD
config ... -net_map -nets {VDD_m} -to VDD
```

However, you cannot apply the -net_map statement multiple times to the same net (one-to-many). For example, the following is *not* allowed:

```
config ... -net_map -nets {VDD_a} -to VDD
config ... -net_map -nets {VDD_a} -to VDD_analog
```

Note

 Case-sensitivity is determined by the config -source_netlist -case setting.

Examples

In this example, chip_1 and chip_2 have nets called “global_reset”. These nets are connected to a net named “reset” on chip_3. This is logically correct, but this could result in misleading LVS errors. In this case, you can map the nets during connectivity checking by specifying the following command:

```
config ... -net_map -nets {global_reset} -to reset
```

-import_net_map

Option for the `config` command.

Imports a text file that contains a list of net mapping statements.

Usage

```
config other_arguments
    -import_net_map -file filename
```

Arguments

- **-file *filename***

Required argument that specifies the path to a file that contains a list of net mapping statements.

Description

The file specified by the `-file` argument set contains one net mapping statement per line. Each line contains two space-separated entries as follows:

```
original_net_name new_net_name
...
...
```

Where the `original_net_name` argument is an existing net name in the design and the `new_net_name` is the desired name.

Note

 The same net cannot be renamed to different values.

Note

 Case-sensitivity is determined by the config `-source_netlist -case` setting.

Examples

In this example, `chip_1` and `chip_2` have nets called “`global_reset`”. These nets are connected to a net named “`reset`” on `chip_3`. This is logically correct, but this could result in misleading LVS errors. In this case, you can rename the nets by specifying a file that contains the correct mapping:

```
config ... -import_net_map -file ./import_nets.txt
```

where the contents of `import_nets.txt` is the following:

```
global_reset reset
```

-pin_map

Option for the [config](#) command.

Specifies pins that can use a different name during connectivity checking. This command is useful for waiving known connectivity errors by adjusting pin names in both the layout and source at runtime.

Note

 This operation does not rename pins in the layout. If you want to change text labels in the design, see “[rename_text](#)” on page 424.

Usage

```
config other_arguments
    -pin_map -pins list_of_pins -to pin_name
```

Arguments

- **-pins *list_of_pins***
Required list of pins to rename.
- **-to *pin_name***
Required name for the list of specified pins. All pins in the *list_of_pins* argument are renamed to the specified value.

Description

This command behaves similar to [rename_text](#), except that the pin names in the layout and source are not physically altered. The pin mapping operation only occurs in memory during connectivity checking. The operation is implicitly applied to the source netlist. If you want to apply net-name based mapping to the source, then the [-net_map](#) option should be used.

Note

 If the specified pin name does not exist in the layout, then the mapping operation is ignored and the tool issues the following warning:

```
3DSTACK_WARNING_401: The pin mapping from <name> to <name> has not been
applied for placement <name> as pin <name> doesn't exist in placement
<name>
```

The nets of the pins that are mapped are merged, forming a logical connection that is later checked against source. In other words, during connectivity checking, the mapped pins are treated as connected even if they are not physically connected.

Multiple pin mapping statements are allowed (many-to-one):

```
config ... -pin_map -pins {VDD_a} -to VDD
config ... -pin_map -pins {VDD_m} -to VDD
```

However, you cannot apply the `-pin_map` statement multiple times to the same pin (one-to-many). For example, the following is *not* allowed:

```
config ... -pin_map -pins {VDD_a} -to VDD
config ... -pin_map -pins {VDD_a} -to VDD_analog
```

Circular pin mapping is also not allowed. For example:

```
config ... -pin_map -pins {A} -to B
config ... -pin_map -pins {B} -to C
config ... -pin_map -pins {C} -to A
```

The above statements end up mapping A to B, and then B to C, and then C back to A. This is not possible and will cause problems during a verification run. To avoid circular mapping, you must carefully review your mapping statements in both the direct `-pin_map` statement and any pin mapping files that were specified.

Note

 Case-sensitivity is determined by the config `-source_netlist` `-case` and `-layout_case` settings.

Examples

In this example, `chip_1` and `chip_2` have pins called “`global_reset`”. These pins are connected to a pin named “`reset`” on `chip_3`. This is logically correct, but this could result in misleading LVS errors. In this case, you can map the pins during connectivity checking by specifying the following command:

```
config ... -pin_map -pins {global_reset} -to reset
```

-import_pin_map

Option for the [config](#) command.

Imports a text file that contains a list of pin mapping statements.

Usage

```
config other_arguments
    -import_pin_map -file filename
```

Arguments

- **-file filename**

Required argument that specifies the path to a file that contains a list of pin mapping statements.

Description

The file specified by the -file argument set contains one pin mapping statement per line. Each line contains two space-separated entries as follows:

```
original_pin_name new_pin_name
...
...
```

Where the **original_pin_name** argument is an existing pin name in the design and the **new_pin_name** is the desired name.

Note

 The same pin cannot be renamed to different values.

Case-sensitivity is determined by the config -source_netlist -case setting.

The nets of the pins that are mapped are merged, forming a logical connection that is later checked against source. In other words, during the connectivity checking these mapped pins are treated as connected even if they are not physically connected.

Circular pin mapping is not allowed. For example:

```
config ... -pin_map -pins {A} -to B
config ... -pin_map -pins {B} -to C
config ... -pin_map -pins {C} -to A
```

The above statements end up mapping A to B, and then B to C, and then C back to A. This is not possible and will cause problems during a verification run. To avoid circular mapping, you must carefully review your mapping statements in both the direct -pin_map statement and any pin mapping files that were specified.

Examples

In this example, chip_1 and chip_2 have pins called “global_reset”. These pins are connected to a pin named “reset” on chip_3. This is logically correct, but this could result in misleading LVS errors. In this case, you can rename the pins by specifying a file that contains the correct mapping:

```
config ... -import_pin_map -file ./import_pins.txt
```

where the contents of *import_pins.txt* is the following:

```
global_reset reset
```

-set_auto_rve_show_layers

Option for the [config](#) command.

Specifies for all rule checks (except tvf_block) whether Calibre RVE layer highlights show only layers used to derive the rule check.

Usage

```
config other_arguments
    -set_auto_rve_show_layers {NO | YES}
```

Arguments

- **NO**
Specifies not to apply -set_rve_show_layers AUTO to any checks that do not have an existing Calibre RVE layer specification. This is the default.
- **YES**
Applies -set_rve_show_layers AUTO to all rule checks that do not have an existing Calibre RVE layer specification.

Description

Use this optional rule file command to control the “-set_rve_show_layers {layer_list | AUTO}” option for all rule checks. When this command is specified with YES in the Calibre 3DSTACK rule file, then -set_rve_show_layers AUTO is applied to all rule checks that do not already specify the -set_rve_show_layers option. The AUTO keyword instructs Calibre RVE to only show the input layers that were used to derive the rule checks.

Note

 In order to use this functionality, you must enable the Calibre RVE option “Only show check-dependent layers while highlighting results (hides other layers)” on the **Setup > Options > Highlighting** pane in the DRC/DFM Highlighting area. See “[RVE Show Layers](#)” in the *Calibre RVE User’s Manual* for complete details.

Examples

```
config ... -set_auto_rve_show_layers YES
```

-set_rve_cto_file

Option for the [config](#) command.

Specifies the path to a check text override file that controls how results are highlighted in Calibre RVE.

Usage

```
config other_arguments
    -set_rve_cto_file -file cto_file
```

Arguments

- **-file cto_file**

Required argument and value that specifies the path to a check text override file.

Description

Use this optional rule file command to specify the path to a check text override (CTO) file. The *cto_file* contains DRC RVE check text comments that control how results are highlighted in Calibre RVE. Only one CTO file is allowed.

Note

 The `-set_rve_*` arguments in your Calibre 3DSTACK rule file take precedence over Calibre RVE options in your CTO file specified for the same rules.

The following CTO file specifies two check text comments for a rule named `pads_on_grid`:

```
# DRC RVE check text override file for a Calibre 3DSTACK rule file
#
pads_on_grid
RVE Highlight color: red
RVE Show Layers: backside_rdl
```

For complete information on the CTO file and how to load it, see “[DRC RVE Check Text Override File \(CTO File\)](#)” in the *Calibre RVE User’s Manual*.

Caution

 Calibre RVE searches your working directory and loads a CTO file named `3dstack.rdb.cto` if it exists. Do not use `-set_rve_cto_file` to specify this file if it exists in your working directory as it is automatically loaded. You may use `3dstack.rdb.cto` as the CTO filename if it exists outside of your working directory.

Examples

```
config ... -set_rve_cto_file -file ./custom_rve_settings.cto
```

-svrf_specs

Option for the [config](#) command.

Includes a file containing SVRF Layout Input Exception Severity statements. If a line in the file does not contain a Layout Input Exception Severity statement, then the line is ignored.

Usage

config other_arguments
 -svrf_specs svrf_file

Arguments

- **svrf_file**

Required path to a file containing Layout Input Exception Severity specification statements.

Examples

```
config ... -svrf_specs ./3d_severity.svrf
```

-units

Option for the [config](#) command.

Specifies the physical units used in the 3DSTACK+ rule file.

Note

 The -units option has no effect on statements inside the 3dstack_block command.

Usage

config *other_arguments*

-units [-distance {um | mm | nm}] [-power {W | mW | uW}] [-time {s | hr | min | ms | us}]

Arguments

- **-distance {um | mm | nm}**

Specifies the units for distance. The value must be microns, millimeters, or nanometers. The default is microns.

- **-power {W | mW | uW}**

Specifies the units for power. The value must be watts, milliwatts, or microwatts. The default is watts.

- **-time {s | hr | min | ms | us}**

Specifies the units for time. The value must be seconds, hours, minutes, milliseconds, or microseconds. The default is s.

Examples

```
config -units {-distance nm}
```

-warning_severity

Option for the [config](#) command.

Specifies the severity level for warning messages.

Usage

config other_arguments

-warning_severity -warning {message_id | *} -severity {0 | 1 | 2} [-count count]

Arguments

- **-warning {message_id | *}**

Specifies the warning message by the ID. You can also specify * to include all warning messages.

- **-severity {0 | 1 | 2}**

Sets the severity level of the message. The following keywords are supported:

0 — Disables the warning message. The tool does not print this message.

1 — Enables the warning message (the default).

2 — Configures the warning message as an error. The tool immediately exits from arun after printing the contents of the message.

- **-count count**

Sets an upper limit on the number of messages printed when the -severity is set to 1. This must be an integer greater than 0.

Examples

```
config ... \
    -warning_severity -warning 3DSTACK_WARNING_108 -severity 1 -count 20
```

process

Defines technology kit layer information for the dies, interposers, packages, and other components used in a 3D-IC.

Usage

```
process -name process_name
  {-layer_info '{' arguments '}' } ...
  [-wb_connect layer1_name layer2_name [BY layer3_name] ] ...
```

Arguments

- **-name *process_name***
Required keyword and name for the process technology.
- **-layer_info '{' *arguments* '}'**
Required argument and value set that specifies a layer in the process. You can specify more than one layer by applying the -layer_info argument set multiple times. Note that you must enclose the arguments for each layer within the {} braces.
See “[-layer_info](#)” on page 84 for details.
- **-wb_connect *layer1_name* *layer2_name* [BY *layer3_name*]**
Optional argument set that defines the connectivity of the process layers. This is only required for hierarchical (white-box) connectivity extraction of the assembly.

Description

The process data is usually provided by the foundry that manufactures your IC. The advantage of applying this command is that you do not have to redefine layer information for each die in the 3D-IC. Instead, you can refer to a process that is already defined in the die command.

Note

 If you specify layer information and a process in the die command, the layer information overwrites the process specification if the same information is defined in the process. If items are not defined in the layer information, the process data is merged with the layer information.

Examples

```
process -name N10
  -layer_info {-type pad -name pad1 -layer 40 -top} \
  -layer_info {-type pad -name pad2 -layer 41 -top} \
  -layer_info {-type merge -name merge \
  -svrf {MERGE (EXTENT DRAWN pad1 pad2)}}
```

After defining a process, you can apply it in the die command as follows:

```
die -die_name INT \
-path ./int.gds \
-process N10 \
-anchor {-name mem -placement 0 0 } \
-anchor {-name log -placement 10 10 } \
-interposer \
-import_text_labels texts
```

Related Topics

[die](#)

export_layout

Generates an annotated GDS and extraction data for a specified portion of the assembly.

Note

 You can use a different xCalibrate version than your current Calibre version. If you set the XCALIBRATE_HOME environment variable to a custom path, Calibre 3DSTACK uses that version instead of the xCalibrate version specified by MGC_HOME.

Usage

export_layout

```
-file output_file
[-output_dir directory]
[-stack stack_name]
[-die '{'-die_name die_name
       [-layer_info {'-name layer_name
                     [-pex_map calibrated_layer_name]
                     [-rc_model]
                     [-bottom_rc_interface | -top_rc_interface]
                     '}']...
       [-rename_text {"expression"... | {-file expr_file}]
       [-second_ground {'-name layer_name
                         {-vertices {x1 y1 x2 y2} | -extent }
                         -text "ground_name"
                         [-pex_map layer_name] '}']
       '}']
     [-cci [-ground_name name] [-power_name name]]
     [-enable_rc_deck | -enable_rc_map]
     [-flat] [-netlist_format {[SPICE] [XSI] [VERILOG] [MGC] [AIF]}]
     [-pex_map_file file]
     [-property property_number]
     [-rename_text "expression"]
     [-text [-use_net_text]]
     [-use_lvs_names]
     [-virtual_connect {net ... }]
```

Arguments

- **-file *output_file***

Required argument set that specifies the path to output the generated layout file. The tool creates a DFM database in *3dstack.dfmdb/pex/<output_file>*, where *output_file* is your specified layout file without the extension. For example, if you specified the following:

```
export_layout -file my_output.agds ...
```

the tool creates a directory named *my_output* in the DFM database and saves all output files from the **export_layout** command in that directory.

```
3dstack.dfmdb/pex/my_output/  
    my_output.agds  
    ...
```

If you want to output the files to a directory instead of a DFM database, apply the **-output_dir** option.

- **-output_dir *directory***

Optional argument set that specifies a directory in which to write all output files from the **export_layout** command. If you do not specify this option, the tool generates all output from the **export_layout** command to *3dstack.dfmdb/pex/<output_file>* in your working directory.

- **-stack *stack_name***

Optional argument set that specifies to export the stack of dies as a new layout file.

- **-die {*arguments*}**

Optional argument that specifies to export one or more interacting dies.

The arguments inside the **-die** option must be enclosed in brackets {} and are described as follows:

-die_name *die_name*

Optional argument that specifies the name of the die to export.

-layer_info {*arguments*}

Optional argument set that specifies layers to include in the exported layout from the die.

The **-layer_info** arguments must be enclosed in brackets {} and are described as follows:

- **-name *layer_name***

Optional argument set that specifies a layer to export on the die.

- **-pex_map *calibrated_layer_name***

Optional argument set that maps calibrated layer names to the physical layers in the assembly. This argument set only applies for -format GDS.

The Calibre 3DSTACK tool uses this information to write an SVRF file when exporting an annotated GDS file. The SVRF file is named *<gds_name>.map.svrf*, where *gds_name* is the name you specified for the annotated GDS. The calibrated layer names and assembly layer pairings are written to the generated SVRF file with **PEX Map** statements. The following is an example:

```
-layer_info {-name M1 -pex_map PM1_M1}
```

- **-rc_model**

Optional argument that specifies to generate additional PEX extraction statements for TSVs in the extraction file, as follows:

```
PEX NETLIST 3DIC <placement-layer-name> FILE <PATH> SUBNODE <sub>
PEX 3DIC COUPLING <placement-layer-name> ANALOG MAXDISTANCE <max_distance>
PEX 3DIC COUPLING <placement-layer-name> DIGITAL <frequency> MAXDISTANCE <max_distance>
```

Where the <PATH>, <sub>, <max_distance>, and <frequency> fields must be completed manually after generating the file and are defined as follows:

- PATH — Specifies the path to the netlist.
 - sub — Specifies the “sub” node in the TSV sub-circuit.
 - max_distance — Specifies the maximum distance (in microns) beyond the extents of the TSV for which the coupling is calculated. Anything beyond the maximum distance is not considered in the coupling calculation.
 - frequency — Specifies the frequency in Hz.
- **-bottom_rc_interface**

Optional argument that indicates that the specified layer represents the interface layer on the bottom of the die. You can only apply this argument to a single layer in the export_layer -die_info argument set. If you apply this argument, you must also specify the -top_rc_interface argument for one of the other layers in the die. This option is used to identify the interface layers for a virtual die between dies or from dies to interposers for extraction rule generation.

- **-top_rc_interface**

Optional argument that indicates that the specified layer represents the interface layer on the top of the die. You can only apply this argument to a single layer in the export_layer -die_info argument set. If you apply this argument, you must also specify the -bottom_rc_interface argument for one of the other layers in the die. This option is used to identify the interface layers for a virtual die between dies or from dies to interposers for extraction rule generation.

- **-rename_text “*expression* ...”**

Optional argument set that renames text in the die layout. The expression is a GNU regular expression. Multiple expressions are supported, but the entire set of *expressions* must be enclosed in quotes or braces. For example:

```
-rename_text "/</\ [ /_ /> /\] /"
```

```
-second_ground {'arguments'}'
```

Optional argument set that enables you to create an additional ground layer in the generated annotated GDS, create a rectangular shape on the layer, and attach a text

label. The tool also writes the information to the *.map.svrf* file. To use this option, you must specify the following *arguments*:

- -name *layer_name* — Specifies the name of the layer on which the second ground shape is generated.
 - -vertices *x1 y1 x2 y2* — Specifies a set of coordinates for the ground pin shape. This option is mutually exclusive with -extent.
 - -extent — Specifies that the extent of the die should be used as the shape for the second ground. This option is mutually exclusive with -vertices.
 - -text “*ground_name*” — Specifies the text label for the second ground.
 - -pex_map *layer_name* — Specifies the calibrated layer name for the second ground.
- -cci [-ground_name] [-power_name]

Optional argument that generates a set of standard files that can be used in third-party extraction flows. This option cannot be specified with the -text, -flat, -enable_rc_deck, -use_lvs_names, or -netlist_format options. The -cci argument includes the following options:

-ground_name *name*

Specifies a ground name used for ground nets in the generated Layer Net Specs (LNS) file.

-power_name *name*

Specifies a power name used for power nets in the generated LNS file.

When you specify the -cci argument set, the tool creates a Layer Net Specs file in your working directory. The LNS file is an SVRF file that contains the following sections:

- Layer definitions.
- Connectivity statements.
- Power and ground name statements, if specified using the -power_name and -ground_name options.

The tool generates a flat spice netlist that contains a top subcircuit without any instances and with sorted net numbers as pins.

If you set the CALIBRE_3DSTACK_ENABLE_HIER_CCI environment variable to a non-null value, the following items include placement information:

- lnn (Layout Netlist Names)
- ixr (Instance Xref File)
- nxr (Net Xref File)
- lnxr (Layout Net Xref File)

- **-enable_rc_deck**

Optional argument that specifies to generate an extraction rule file for interface layers between two specified dies. You must specify the extraction files for each die with the -ircx or -mipt options in the [-layer_info](#) option of the die or process commands. The tool does the following when you specify this option:

- a. Retrieves interacting layers in each of the specified dies. The order of the layers is read from the specified stack.

The dies that contain the interacting layers must have a MIPT or iRCX file, otherwise the tool issues an error.

- b. Generates a mapping file and MIPT file for the interacting layers between the specified dies in the `export_layout` output directory.
- c. Uses the mapping and MIPT file to generate the final RC rules file for the interface layers.
- d. Generates a log file named `xcalibrate.log` in the `export_layout` output directory.

You can specify the `export_layout` command with this option for each of the interacting dies. This option cannot be specified with `-enable_rc_map`.

- **-enable_rc_map**

Optional argument that specifies to only generate the `xcalibrate_map` file. This argument only applies to `-enable_rc_deck` and cannot be specified with `-layout_only`. If the MIPT files were not specified, the tool creates placeholders in the map file with the following format:

```
<DIE_NAME>_mipt
```

This option cannot be specified with `-enable_rc_deck`.

- **-flat**

Optional argument that specifies to export a flat layout. If this is not specified, then the output is hierarchical.

- **-netlist_format {[SPICE] [XSI] [VERILOG] [MGC] [AIF]}**

Optional argument set that specifies the netlist format to output. You can specify a single format or a space-separated list containing multiple formats.

- **-pex_map_file file**

Optional argument set that specifies a mapping file that corresponds LVS layer names to RC layer names. The tool uses the RC names and defined mapping to generate PEX MAP statements for the LVS rule file. The mapping information read from the file takes precedence over the information specified in the `-pex_map` argument set.

The file is formatted in a two column, space-separated text file as follows:

```
#   column 1           column 2
    RC_LAYER_NAME     LVS_LAYER_NAME
```

For example:

```
#RC layer names      LVS layer names
BOT_TOPMETAL_MINUS1 AP
TOP_TOPMETAL        P_die1
BOT_TOPMETAL        P_die2
BOT_TOPVIA          C
```

The tool skips a line if it contains less than two space-separated fields or if the line starts with a comment character (#).

- **-property *property_number***

Optional argument that specifies the property number to which to write connectivity information in the annotated GDS file. The default property number is 1. The *number* argument must an integer between 0 and 65535. This option only applies to -format GDS and is mutually exclusive with -text.

- **-rename_text {“*expression*”... | {-file *expr_file*}**

Optional argument set that renames text in the exported layout. The expression is a GNU regular expression. Multiple expressions are supported, but the entire set of *expressions* must be enclosed in quotes or braces. For example:

```
-rename_text "/</\[/ />/\]/"
```

Instead of in-line expressions, you can specify the path to a file that contains a list of expressions. The file contains one expression per line. For example:

```
/A/B/
/A.*B/R/
/^VDD.*/VCC/
/A/B/g
```

- **-text**

Optional argument that specifies to include text in the generated layout file. If this argument is specified, the -flat argument is also applied automatically.

- **-use_net_text**

Optional argument that specifies to use the net text for the pin names on all layers. When specified, text layers containing net name labels are treated as pin text layers and are used in Port Layer Text and Attach statements in the generated extraction rule file. If you do not specify this option, the tool uses the text and pin layers defined in the [-layer_info](#) option.

- **-use_lvs_names**

Optional argument that specifies to use original layer names from the [-layer_info](#) option in the generated LVS rules file. If this argument is not specified, the tool uses the layer names from the generated Calibre 3DSTACK assembly. This argument cannot be specified with -cci.

- **-virtual_connect {net ... }**

Optional argument set that specifies a list of nets to which the virtual connect statement is applied. When the virtually connected nets are written to the *.map.svrf* file, Calibre xACT treats the virtually connected bumps on the same net as if they are connected, even if they are physically disconnected. The *net* value can be a regular expression.

Description

The *export_layout* command also generates a template Calibre xACT rule file that contains all possible SVRF statements and placeholders for statements that need to be modified. The generated template file is written to the specified annotated GDS directory with the extension *.template.svrf*.

Statements enclosed in brackets “<>” are placeholders that must be completed. The following is an example:

```
LAYOUT PATH "anno.gds"
PEX NETLIST <SPEF-file-name> SPEF LAYOUTNAMES CLAYER CLOCATION //MAPNAMES
//NOINSTANCEX RLAYER CLAYER
PEX NETLIST CAPACITANCE UNIT ff
PEX NETLIST ESCAPE CHARACTERS OFF
MASK SVDB DIRECTORY "svdb" XACT
LAYOUT SYSTEM GDS //LEFDEF
LAYOUT CASE YES
SOURCE CASE YES

PEX EXTRACT TEMPERATURE 27

LAYOUT USE DATABASE PRECISION YES
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 0
LAYOUT PRECISION 2000
PRECISION 2000

PEX EXTRACT EXCLUDE LAYOUTNAMES RECURSIVE "?VSS?" "?VDD?"
UNIT CAPACITANCE ff

//ignore self-capacitance of the layers for which the coupling capacitance
//is desired
PEX IGNORE CAPACITANCE ALL SUBSTRATE assembly_tier1_info1_PM0
assembly_tier1_info1_PM0
PEX IGNORE CAPACITANCE ALL SUBSTRATE assembly_tier2_FCCC_N10_top1_AP
assembly_tier2_FCCC_N10_top1_AP

INCLUDE ./path_to_lvs_deck
INCLUDE <path_to_C_rules>INCLUDE <path_to_R_rules>
INCLUDE <path_to_xact_rules>
```

Examples

Export the assembly stack and files to a directory instead of a DFM database:

```
export_layout -file stack.agds -output_dir ./extraction_output \
-stack assembly
```

Export the layout interaction between the controller die and the interposer:

```
export_layout -file cont_to_interposer.gds \
    -die {-die_name controller -layer_info {-name cont_c4}} \
    -die {-die_name interposer -layer_info {-name interposer_pads}} -text
```

Rule Check Commands

Rule check commands perform specific checks on your design during a verification run.

Rule checks are specified as follows:

- Checks in the 3DSTACK+ file operate on placement layer types. The layer type is a custom string that is defined in the die -layer_info, component -layer_info, or process -layer_info commands.
- You can specify stack names in rule checks to apply rules to certain stacks.
- The -direction argument allows you to specify the vertical direction in which a check is applied.
- The connected check has options to specify white box and black box checking. See “[connected](#)” on page 147 for details.
- You can define a custom check using TVF statements with the [custom_check](#) command.

Table 3-6. Rule Check Commands

Command	Description
centers	Checks for misaligned pads.
connected	Checks for connectivity between connected shapes when there is a source netlist present. When no source netlist is present, it checks for matching text strings on connected shapes.
copy	Exports specified layer contents.
custom_check	Defines custom checks using TVF format for the 3DSTACK+ syntax.
dangling_ports	Checks for placed ports that do not have physical connectivity on a specified placement or layer type.
dangling_no_text	Checks for orphan ports that are missing text labels.
density	Checks for density constraints on specified placements.
enclosure	Checks for sufficient separation distance between the exterior-facing sides of the first placement’s edges and the interior-facing sides of the second placement’s edges.
external	Checks for sufficient separation distance between the exterior-facing sides of the first placement’s edges and the exterior-facing sides of the second placement’s edges.
extra_ports	Checks for missing ports in the source (or extra ports in the layout) on a specified chip, placement, die or layer.

Table 3-6. Rule Check Commands (cont.)

Command	Description
floating_pads	Checks for pads of a layer type or stack that do not overlap any other pads from other dies or interposers in the stack.
floating_texts	Checks for text labels that are not attached to any geometry on a specified chip, placement, die or layer.
internal	Checks for sufficient separation distance between the interior-facing sides of the first placement's edges and the interior-facing sides of the second placement's edges.
locations	Checks the location of pads and that the shape and text match between source and layout.
missing_ports	Checks for ports in the source netlist that have no corresponding layout pad on a specified chip, placement, die or layer.
multi_texts	Checks for multiple text labels attached to the same pad on a specified chip, placement, die or layer. In the case where multiple text labels overlap a pad in the layout, the tool generates a multi-text error and chooses to use one of the text labels attached to the pad for the connectivity analysis. Because the chosen label may not match your design intent, it is important to review and resolve all multi-text errors to ensure that the layout is correct. If you do not resolve the multi-text errors, your connectivity analysis may not be correct.
no_texts	Checks for missing text labels for pads on a specified chip, placement, die or layer.
offgrid_centers	Checks for pad centers that are not aligned to a specified grid.
overlap	Checks for sufficient overlap between two placed layers.
same_size	Compares the sizes of interacting pads and reports mismatches. Use this command to ensure that the pads of two connected dies are of the same size.
select_checks	Selects specified checks to execute during a Calibre 3DSTACK verification.
unconnected_ports	Reports ports that are both not connected to any other port in the layout and in the source netlist. A source netlist is required.
unselect_checks	Selects specified checks not to execute during a Calibre 3DSTACK verification.

Table 3-6. Rule Check Commands (cont.)

Command	Description
<code>3dstack_block</code>	Defines a code block in which standard Calibre 3DSTACK rule file format commands can be specified. This command allows you to directly pass standard 3DSTACK syntax commands to the Calibre 3DSTACK run.

centers

Checks for misaligned pads.

Usage

centers

```
-check_name check_name
{{{-layer_type1 placed_layer_type1 [-same_die]} [-layer_type2 placed_layer_type2]}}
-constraint "constraint_value"
[-alignment {octagonal_only | orthogonal_only}]
[-direction {up | down | both}]
[-overlapping]
[-square]
[-stack '{' stack_name_list '}']
[-comment "comment"]
[rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing output results. If centers is specified multiple times, each ***check_name*** must be unique.

- **{-layer_type1 *placed_layer_type1* [-same_die]} [-layer_type2 *placed_layer_type2*]**

Argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers to the layer of the die placement.

The arguments behave as follows:

-layer_type1 *placed_layer_type1* — Required argument set that specifies a layer type on which to check the centers alignment.
-same_die — Optional argument set that specifies to only perform the centers check on a single layer type. You must include this option if you do not specify -layer_type2.
-layer_type2 *placed_layer_type2* — Optional argument set that specifies a second layer type. The centers alignment is compared between polygons on the first layer type and the second layer type

For example, if a check is defined between layer_type1 and layer_type2, all the placement layers of layer_type1 that interact with the placement layer of layer_type2 (as defined by the stack) are checked against each other.

If you do not specify -layer_type2, you must also include the -same_die option. In this case, the check is done on all placement layers of ***placed_layer_type1*** type.

- **-constraint "*constraint_value*"**

Required argument and constraint that specifies the dimensions of the measurement region around a pad of the specified layer type, which must have an upper bound. The

constraint_value must conform to the constraint notation described under “[Constraints](#)” in the [Standard Verification Rule Format \(SVRF\) Manual](#).

The expression may not be of the form ">", ">=", or "!=".

- **-alignment {octagonal_only | orthogonal_only}**

Optional argument and keyword that defines the alignment of pad centers for distance measurement. The options are defined as follows:

- octagonal_only — the distance between the center points of the pad extents is defined to satisfy the constraint only if the center-points are aligned at multiples of 45 degrees.
- orthogonal_only — the distance between the center points of the pad extents is defined to satisfy the constraint only if the center-points are aligned in either the x- or y-direction.

- **-direction {up | down | both}**

Optional argument and value set that specifies the direction of the check. If the -direction argument is specified, the checks are performed only in the specified direction.

up — The check is only performed from the bottom to the top of the stack. This is the default.

down — The check is only performed from the top to the bottom of the stack.

both — The check is performed in both directions

- **-overlapping**

Optional argument that checks the specified constraint between two overlapping placements. Floating pads are ignored if -overlapping is specified. This option must be specified with two layers using the **placed_layer_type1** and **placed_layer_type2** arguments.

- **-stack '{' stack_name_list '}'**

Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the -stack argument is specified, the checks are applied only to specified stack(s).

- **-square**

Optional argument that specifies that the distance measurement is performed using a square metric, rather than a Euclidean metric.

- **-comment “comment”**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- *rve_option ...*

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

This command checks whether the pad centers are aligned, or if the centers are spaced at a specified distance. Or in other words, the **centers** check flags pads that are not within a specified distance of other pads. The measurement is performed using the centers of the original geometries (not the extents) of the pads. Its functionality is similar to the [Not With Neighbor](#) SVRF operation using the **CENTERS** option.

If you want to check the center alignment of overlapping pads, specify the **-overlapping** argument. This option only checks the specified constraint between two overlapping pads. If this option is not specified, the **centers** command checks for neighboring and overlapping pads.

Note

 The **centers** command operates with rectilinear shapes. If the geometries that you want to check include non-rectilinear shapes, then it is recommended to generate layer derivations without the non-rectilinear shapes and then use the derived layer in the check.

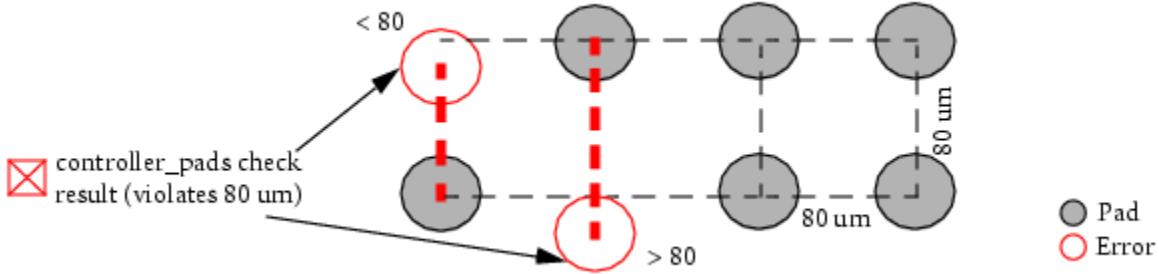
Examples

Example 1

This example reports any pads on a layer placement that are not exactly within a center-to-center distance of 80 microns.

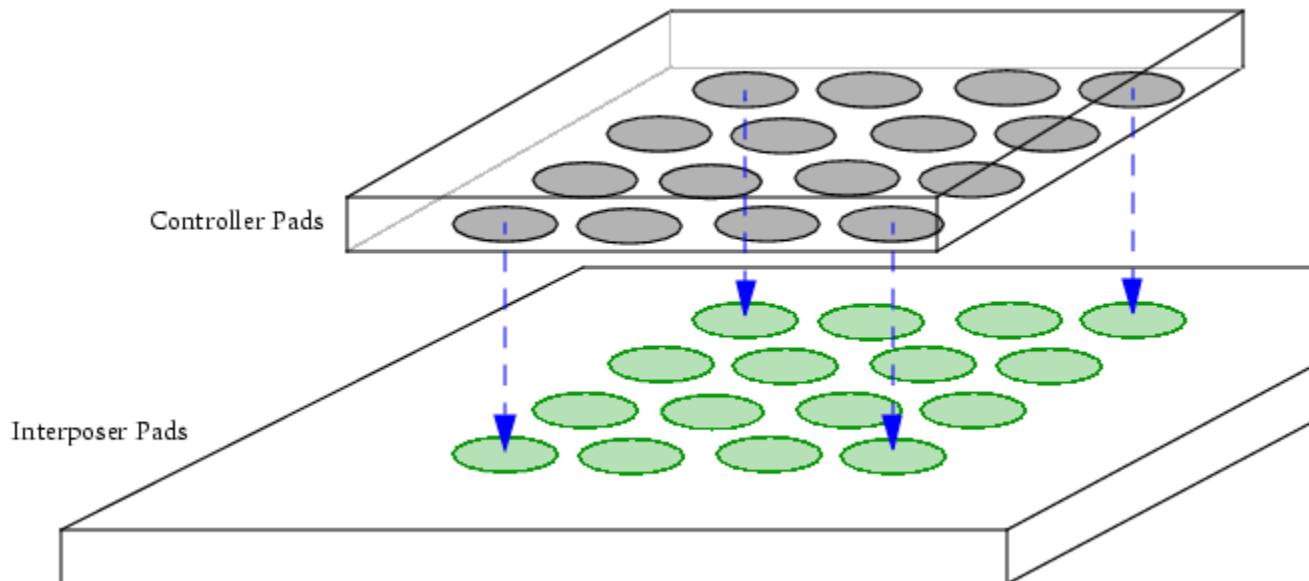
```
centers -check_name controller_pads -layer_type1 pad -same_die \
    -constraint "== 80" -alignment orthogonal_only \
    -comment "Pads spaced 80 um in x or y from all pad centers"
```

If any pads on all pad type layers on the same die are not exactly 80 um away from each pad’s center, Calibre 3DSTACK flags them by selecting the offending pads, similar to this:



Example 2

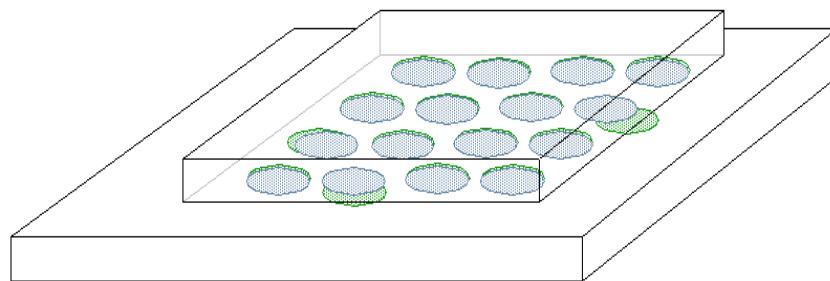
This example demonstrates how to check the alignment between two overlapping pad placements. In this case, a chip is stacked on an interposer and you want to ensure that the pads do not misalign more than 0.1 um.



The following rule check defines a maximum center-to-center deviation between the cont_placement and inter_placement pads of 0.1 um. It also specifies that the pad placements are orthogonally aligned and the detection region is square shaped.

```
centers -check_name centers_controller_interposer \
-layer_type1 pad -stacks {controller interposer} \
-constraint "<= 0.1" -square -alignment orthogonal_only -overlapping \
-comment "Center-to-center spacing for the controller to \
interposer_controller landing pads must be within 0.1 um in the \
x or y direction (ortho)."
```

In this example, the check selects any interposer pads with centers more than 0.1 um away from the centers of the controller pads:



Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

connected

Checks for connectivity between connected shapes when there is a source netlist present. When no source netlist is present, it checks for matching text strings on connected shapes.

Usage

connected

```
-check_name check_name
{[-layer_type1 placed_layer_type1 [-layer_type2 placed_layer_type2]]
 | [-die1 die_name {[-standalone [-short_only | -open_only]] | [-die2 die_name]}}
 [-stack '{' stack_name_list '}']
 [-detailed]
 [-black_box]
 [-white_box]
 [-isolate_path]
 [-net_mismatch {ALL | {[MULTI_NAME] [MISMATCH] [MISSING_NAME]}]}
 [-nothrutier]
 [-no_dangling_ports]
 [-no_extra_ports]
 [-no_missing_ports]
 [-pin_list list_of_pins]
 [-text_checks {'ALL | NONE | [type ...]'}]
 [-comment "comment"]
 [rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each **check_name** must be unique.

- **-layer_type1 *placed_layer_type1* [-layer_type2 *placed_layer_type2*]**

Argument and value set that specifies the type(s) of the layer for which the check is applied.

If you do not specify a second layer type with the -layer_type2 option, Calibre 3DSTACK performs connectivity checks on *placed_layer_type1* layer inside the die. See “[Internal Connectivity Checking](#)” on page 153 for details.

If you do not specify a layer type or a die, the tool performs a connectivity check on the entire assembly.

- **-die1 *die_name* {[-standalone [-short_only | -open_only]] | [-die2 *die_name*]}**

Argument and value set that checks connectivity for one or more dies. This argument cannot be specified with the -layer_type1 or -layer_type2 arguments.

The benefit of this die-name to die-name connectivity checking is that pin text layers can be attached to any one (or multiple) layers in the die and Calibre 3DSTACK automatically constructs the pins for this die. If you perform layer-type to layer-type connectivity

checking, pin text layers must be attached to the layer type that represent the pins. This argument set has the following two modes:

- -die1 *die_name* -standalone — Checks the internal connectivity of the specified die. This option checks the internal connectivity of the die in isolation, ignoring any external connections from the assembly operations. If you use this option and your assembly only contains one die, then you do not need to include any stack definitions because the tool generates the stack internally.

Note



Dangling port checks are automatically disabled for standalone connectivity checking.

- -short_only — Optional argument that specifies to exclusively reports connectivity errors due to shorted nets. This option can only be used on a standalone die using the “connected -die1 *die* -standalone” argument set.
- -open_only — Optional argument that specifies to exclusively reports connectivity errors due to open nets. This option can only be used on a standalone die using the “connected -die1 *die* -standalone” argument set.
- -die1 *die_name* -die2 *die_name2* — Checks the connectivity between the two specified dies.

- **-stack ‘{’ *stack_name_list* ‘}**

Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the -stack argument is specified, the checks are applied only to specified stack(s).

- **-detailed**

Argument that enables detailed reporting of connectivity errors. This option is always enabled. Additional properties are reported for each result:

- LNC (Layout Net Components) — the names of the ports connected to the layout net.
- SNC (Source Net Components) — the names of the ports connected to the source net.

This feature has no effect if [source_netlist](#) is not present in the chip stack rule file.

- **-isolate_path**

Optional argument that enables path isolation for connected results that involve shorts. This feature enables you to highlight paths between shorted pins or pads in your design that can help debug the cause of connectivity errors.

When you enable this option, the shorts in connectivity results are filtered under a *<connect_check_name>*_isolated result to display only the ports that are physically

involved in the short. This option has no effect on open circuit results. See “[Using Path Isolation to Debug Shorts](#)” on page 49 for an example.

- `-net_mismatch {ALL | {[MULTI_NAME] [MISMATCH] [MISSING_NAME]} }`
Optional argument that specifies to check for net text issues on the specified layer. You must specify either [attach_text -net_text](#) or [die -layer_info -net_text](#) on the layer, depending on your rule file syntax. The tool issues a warning and skips net text checks if these options are not specified for the layer. MISMATCH and MISSING_NAME errors are reported by default.

See “[Net Text Mismatch Checking](#)” on page 154 for details.

The options behave as follows:

ALL

Report all net mismatch problems.

MULTI_NAME

Report multiple net names in the layout that belong to a single net name in the source.

MISMATCH

Report net names in the layout that are different than net names in the source. The tool performs this check by default.

MISSING NAME

Report missing net names in the layout for net names found in the source. The tool performs this check by default.

The following is an example:

```
connected ... -net_mismatch {MULTI_NAME MISMATCH}
```

- `-no_dangling_ports`

Optional argument that specifies to exclude DanglingPort errors from the verification results. DanglingPort errors are reported when port placements have no physical connectivity with connectivity layers. To check for this type of error without performing a full connectivity check, use the [dangling_ports](#) command.

- `-no_extra_ports`

Optional argument that specifies to exclude ExtraPort errors from the verification results. If you want this option to apply to an entire layer, then it must be specified for each connected check in which the layer is specified. To check for this type of error without performing a full connectivity check, use the [extra_ports](#) command.

Note

 If you apply this option, extra port errors are listed in the report within the context of each connected command.

- **-pin_list *list_of_pins***

Optional argument set that specifies a list of pins for which to report shorts or opens. This argument set can only be applied when you are checking a single layer type with the **-layer_type1** argument set. If this option is not specified, shorts and opens are reported for all pins. See Example 3 later in this section.

- **-no_missing_ports**

Optional argument that specifies to exclude MissingPort errors from the verification results. If you want this option to apply to an entire layer, then it must be specified for each connected check in which the layer is specified. To check for this type of error without performing a full connectivity check, use the [missing_ports](#) command.

- **-text_checks ‘{ALL | NONE | [*type* ...]}’]**

Optional argument and keyword list that specifies the types of text checks that you want to execute. The results of these checks are reported for each specified connected command. You can specify either of the following options to enable or disable all text checks:

- ALL — All text checks are performed. This is the default.
- NONE — No text checks are performed.

Otherwise, you can specify one or more of the following text *type* options:

- NO_TEXTS — Missing text is checked.
- FLOATING_TEXTS — Floating text is checked.
- MULTI_TEXTS — Multiple text labels on one object are checked.

Note

 To check for these types of errors without performing a full connectivity check, use the [floating_texts](#), [multi_texts](#), and [no_texts](#) commands.

- **-comment “*comment*”**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- **rve_option ...**

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

- **-black_box**

Optional argument to specify black box connection checking. As an example, even if internal routing layers for the interposer are defined in the assembly file, block box checking is performed. If tiers are defined with the stack command, connectivity checks are applied between the interposer and the dies of the interacting tier but not between dies within a tier.

- **-white_box**

Optional argument to specify white box connection checking. As an example, routing layers are checked through the interposer. If tiers are defined with the stack command, connectivity checking is applied between dies within the tier that interact with the interposer. A compiler error occurs if the internal routing layers of the interposer are not defined in the assembly file and -white_box only is specified.

Note

Both -black_box and -white_box may be specified. If neither is specified, both are used.

- **-nothrutier**

Optional argument that specifies not to perform through-tier connectivity checking in the connected check. This option ignores internal layers that pass through dies from the connected check.

Description

The connected check is applied in the following manner in a 3DSTACK+ rule file:

- Checks between interfacing placement layers defined by the stack(s).
- Checks between placement layers within the same tier that directly interface with the interposer die placement.
- Checks between dies by specifying two die names.
- Both -black_box and -white_box may be specified. If neither is specified, both are used.

Note

 If you specify multiple layer types in different connected checks for the same dies, then the tool ignores the duplicated connected rule checks and only executes one of the checks. In this case, the tool issues the following warning:

```
3DSTACK_WARNING_1311: Connectivity check conn_check is a duplicate of conn2 . Ignoring...
```

This command has two modes, depending on the presence of a source netlist in the rule file:

- **Mode 1** — Verifies that connected shapes on the specified placed layers form valid connections, and outputs results to a check named **check_name**. This mode is enabled when config -source_netlist is present in the chip stack rule file. (See “[Creating a Calibre 3DSTACK Rule File](#)” on page 25 for more information.)
- **Mode 2** — Verifies that connected shapes on the specified placed layers contain identical text strings, and outputs results in a check named **check_name**. This mode is enabled when config -source_netlist is not present in the chip stack rule file. (See “[Creating a Calibre 3DSTACK Rule File](#)” on page 25 for more information.)

More information on each mode is given later in the section.

Note

 If you do not specify a second layer or layer type (depending on the syntax), Calibre 3DSTACK performs connectivity checks on the first specified layer inside the die. See “[Internal Connectivity Checking](#)” on page 153 for details.

Typically, this command appears more than once in the chip stack rule file.

Caution

 Calibre 3DSTACK issues an error message if placements involved in a `connected` check are not part of a connectivity stack. A connectivity stack is defined by explicitly issuing the die `-wb_connect` command in the rule file or through implicit interactions generated by Calibre 3DSTACK. To avoid this error, the connectivity should be properly defined.

Mode 1

If the `config -source_netlist` command is present in the chip stack rule file, this command verifies that connected shapes on the specified placed layers form valid connections, and outputs results to a check named `check_name`.

Connectivity data is extracted from the chip stack based on placement layer connections (either defined explicitly using a die `-wb_connect` statement, or implicitly if there are no `connect` statements) and text layers. You can also define implicit connectivity between die layers using the `connect` command. This connectivity data is compared with the connectivity data defined in the source netlist. All mismatches are considered invalid connections and are reported as errors. Connected shapes that do not contain text are output in a separate text checks.

By default, three properties are reported for each result:

- **Net** — The 3D assembly node number that the pin is connected to.
- **Port** — The 3D assembly port name in the form `placement_name:net_name`.
- **SourceNet** — The netlist node number that the pin is connected to. If the netlist node number cannot be determined, “UNKNOWN” is used for the value.

Mode 2

If the `config -source_netlist` command is not present in the chip stack rule file, this rule check verifies that connected shapes on the specified layers contain identical text strings, and outputs results in a check named `check_name`. Connected shapes that do not contain text are output in a separate text checks. If there are no `connect` statements in the rule file, a `connections` are executed implicitly for the specified layers.

Internal Connectivity Checking

If you do not specify a second placed layer or layer type with the `-layer_type2` option, Calibre 3DSTACK performs connectivity checks inside the die using only the first layer type. No source netlist is required for these checks. In this mode, the following checks are performed:

- **Shorts** — Checks that all pins connected to the same net have the same name. If the names are different, Calibre 3DSTACK reports a short for the associated geometry.
- **Opens** — Checks that one or more pins that share the same name are physically connected. If they are not connected, Calibre 3DSTACK reports an open for the associated geometry.

The shorts and opens are reported in sections of the Calibre 3DSTACK report file:

```
RuleCheck: CONNECT_1
-----
-----  
OPENS
-----
1
Port: pRAM_stack1:vdd
Net: 68
LNC: pRAM_stack1:vdd
...
-----
SHORTS
-----
200
Port: pRAM_stack1:vdd
Net: 68
SNC: pRAM_stack1:vdd
...
```

Text Checks

All text-related verification checks are reported by the type of text result and the chip it belongs to (if the placement is part of a connectivity check). The text results are organized into the following categories:

- **Floating text** — Text labels that do not overlap with a pad. Apply the [floating_texts](#) command outside of connected statements.
- **No text** — Pads that are missing text labels. Apply the [no_texts](#) command outside of connected statements.
- **Multi text** — Pads that have more than one text label (overlapping labels). In certain cases, pads can overlap and create overlapping text labels. In these cases, the following warning message is issued:

```
WARNING: there are multiple text-labels overlapping with the pads on
layer layer.
"connected" check may produce incorrect results!
```

Multi text warnings are not issued if the overlapping text labels are identical. Apply the [multi_texts](#) command outside of connected statements.

In the case where multiple text labels overlap a pad in the layout, the tool generates a multi-text error and chooses to use one of the text labels attached to the pad for the connectivity analysis. Because the chosen label may not match your design intent, it is important to review and resolve all multi-text errors to ensure that the layout is correct. If you do not resolve the multi-text errors, your connectivity analysis may not be correct.

Note

 If there are multiple placements for one chip and the placements do not use [rename_text](#) operations, the text results are reported for the first placement only (the other affected placements are mentioned in the generated Calibre RVE check comments).

- **Extra Port** — Missing ports in the source (or extra ports in the layout). The [source_netlist](#) command must be included in order for these results to be produced. Calibre RVE allows you to highlight and cross-reference LNC and SNC ports from the details pane. This result can occur if one or more of the following conditions are true:
 - The subcircuit for a placement is missing a pin definition in the source netlist.
 - When multiple text labels are found on one pad, Calibre 3DSTACK chooses one of the labels and associates it with that pad during connectivity extraction. As a result, an Extra Port result can occur because the placement port may not be found in the source netlist.

You can disable this check by specifying the [-no_extra_ports](#) option.

- **Missing Ports** — A port in the source netlist has no corresponding layout pad. The [source_netlist](#) command must be included in order for these results to be produced. When the result is highlighted, there is no layout object to highlight, so the result properties are displayed in the bottom left corner of the top cell in the assembly.

The result includes the properties SourceNet and MissingPort and these properties are displayed as links in the Result Data Pane—click the links to highlight the source net and port. The Calibre 3DSTACK DFM Database and the source netlist must be open in Calibre RVE; see “[Debugging Connectivity Errors in Calibre 3DSTACK](#)” on page 44.

You can disable this check by specifying the [-no_missing_ports](#) option.

Net Text Mismatch Checking

To set up checking of layout net text and source nets, you must specify [die -layer_info -net_text](#) on the layer, depending on your rule file syntax.

To perform net text checking, you must specify the connected [-net_mismatch](#) option on the given layers.

A net is considered matching (no issues) when there is only one net text label in a given path and that net text label matches the source net name in the source netlist. If this is not the case, all layout pin instance geometries on the path are marked with a net mismatch error.

The `<check_name>_net_mismatch` result reports the following issues:

- **Missing Name** — There is no name associated with the current layout net. All pads of the net are reported.
- **Mismatch** — The source net name is not among the associated net names.
- **Multi Name** — Multiple net names are attached to the same layout net. All text pads are reported.

The result also contains the following formation:

- **Net** — All associated net names.
- **SourceNet** — The source net name.

Examples

Example 1

This example defines two dies, INT and MEM, and a stack of two placements, where the MEM die is placed on top of one INT die.

Based on the defined stack, the connectivity check conn1 derives three separate connectivity checks:

- Between the bump type of placement layers of the INT die placement and the first placement of the MEM die
- Between the bump type of placement layers of the INT die placement and the second placement of the MEM die
- A separate check between the bump type of placement layers of two MEM die placements, since they are defined within the same tier.

```
die -die_name INT -interposer -layout {-path ./INT.gds} \
    -anchor {-name mem1 -placement 0 0} \
    -anchor {-name mem2 -placement 1000 0} \
    -layer_info {-type bump -bottom -layer 43 -text 44}
die -die_name MEM -layout {-path ./MEM.gds} \
    -anchor {-name int -placement 0 0} \
    -layer_info {-type bump -bottom -layer 40 -text 40}
stack -stack_name w_tier \
    -die {-name INT -invert -rotate 90} \
    -tier {
        -die {-name MEM -anchor INT mem1 anchor_INT} \
        -die {-name MEM -anchor INT mem2 anchor_INT} \
    }
connected -check_name conn1 -layer_type1 bump -layer_type2 bump
```

Both white box and black box checking is performed (the default).

Example 2

The following example performs black box connectivity checks on a 2.5D design. This can be performed even if the internal routing layers of the interposer are defined in the assembly file.

```
connected -check_name connect_bb -layer_type1 pad -layer_type2 pad \
           -black_box \
           -comment "BB CONNECT between interposer and interacting dies"
```

The following check performs white box checking. It checks the physical connection of the pad layers through the interposer.

```
connected -check_name connect_wb -layer_type1 pad -layer_type2 pad \
           -white_box \
           -comment "WB CONNECT between interposer and interacting dies"
```

Example 3

If you specify only one layer type, Calibre 3DSTACK checks the connectivity of that layer inside the die. For example, the following rule checks for opens and shorts on geometry connected to the bump layer type.

```
connected -check_name internal_open_short \
           -stack assembly \
           -layer_type1 bump -isolate_path \
           -comment "Opens or Shorts detected on dies"
```

For this same example, you can specify a list of pins to check on the layer type as follows:

```
connected -check_name internal_open_short \
           -stack assembly \
           -layer_type1 bump \
           -pin_list {VDD VSS} \
           -comment "Opens or Shorts detected on dies for VSS/VDD"
```

In this case, the tool only reports shorts or opens on the VDD and VSS pins.

Example 4

The following example checks the connectivity between two dies:

```
connected -check_name cont_to_ram \
           -die1 controller -die2 ram -isolate_path \
           -comment "CONT_CHECK: Die to Controller"
```

Example 5

The following example checks the internal connectivity of a single die:

```
connected -check_name interp_conn_internal \
           -die1 interposer -standalone \
           -comment "CONT_CHECK: INTERNAL CONNECTIVITY CHECK"
```

Example 6

The following example verifies the connectivity between all dies in the assembly using a single check:

```
connected -check_name CONNECT_ALL -comment "FULL CHECK"
```

Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

copy

Exports specified layer contents.

Usage

```
copy
  -check_name check_name
  -layer_type placed_layer_type
  [-direction {up | down | both}]
  [-stack '{' stack_name_list '}']
  [-comment "comment"]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing output results. If copy is specified multiple times, each *check_name* must be unique.

- **-layer_type *placed_layer_type***

Argument and value set that specifies the type of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers to the layer of the die placement.

- **-direction {up | down | both}**

Optional argument and value set that specifies the direction of the check. If the -direction argument is specified, the checks are performed only in the specified direction.

up — The check is only performed from the bottom to the top of the stack. This is the default.

down — The check is only performed from the top to the bottom of the stack.

both — The check is performed in both directions

- **-stack '{' *stack_name_list* '}'**

Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the -stack argument is specified, the checks are applied only to specified stack(s).

- **-comment "*comment*"**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

Description

Use this command to copy and export the layer contents. This command saves the layer placement in DFM database format and exports it into the *3dstack.rdb* and corresponding auxiliary RDB.

Examples

```
copy -check_name enc_check -layer_type bump \
      -comment "EXPORT OPERATION\n Writing layer type bump to DFM database"
```

Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

custom_check

Used in the 3DSTACK+ extended syntax file. Cannot be used in the standard rule file.

Defines custom checks using TVF format for the 3DSTACK+ syntax.

Usage

```
custom_check
  -check_name check_name
  -layer_type1 '{' placed_layer_type1... [-merge] '}'  
  [-layer_type2 '{' placed_layer_type2 ... [-merge] '}']  
  -tvf '{' tvf_body '}'  
  [-direction {up | down | both}]  
  [-stack {stack_list}]
```

Arguments

- **-check_name *check_name***

Required argument set that specifies the check name. The check name must be unique in the rule file. The check name is used in the output result database.

- **-layer_type1 '{' placed_layer_type1 ... [-merge] '}'**

A required argument set that specifies the layer type(s) the check applies to. If -merge is specified all layers of the specified type(s) are merged before the check is applied.

- **-layer_type2 '{' placed_layer_type2 ... [-merge] '}'**

An optional argument set that specifies one or more layer types used for a second layer argument in the rule check. The check is applied to layers of layer_type1 and layer_type2 that interact. If -merge is specified all layers of the specified type(s) are merged before the check is applied.

- **-tvf '{' tvf_body '}'**

Required argument set that specifies a block of TVF statements that define a rule check. See the “Description” section for more information on writing the rule check body.

If a list is used within the command body, braces ({...}) should be used to enclose the list to ensure correct evaluation. Do not use the list syntax ([list ...]).

- **-direction {up | down | both}**

Optional argument set specifying the direction of the check if two layer types are specified.

- **up** — Apply check from bottom to top (default).
- **down** — Apply the check from top to bottom.
- **both** — Apply the check in both directions.

- -stack *stack_list*

Optional argument set that specifies the stack(s) the check is applied to. The check is applied to all stacks if this argument is not specified.

Description

The custom_check command defines a custom rule check using TVF statements. Internally, the tool generates a set of TVF blocks with the correct layer arguments depending on the layer types specified with -layer_type1 and -layer_type2.

If custom_check is defined with only the *-layer_type1* argument, then a set of TVF rule checks is generated for all layers corresponding to *placed_layer_type1*. If -merge is specified all matching layers are merged and only one rule check is generated for the merged layer.

If both *-layer_type1* and *-layer_type2* are specified, then a set of TVF rule checks is generated using interacting layer pairs of type *placed_layer_type1* and *placed_layer_type2*. If -merge is specified all matching layers are merged and only one rule check is generated for the merged layer.

Rule Check Construction

The rule check body (*tvf_body*) is constructed with tvf::SETLAYER and tvf::OUTLAYER statements. Each rule check body must have at least one TVF::OUTLAYER statement defining the output layer of the rule check. Use multiple tvf::OUTLAYER statements to output more than one layer.

The TVF::OUTLAYER statement can have only one argument, the output layer name, when used in Calibre 3DSTACK. Use the TVF::SETLAYER statement for layer derivations. For example, the following construction causes an error:

```
# INCORRECT, multiple arguments to OUTLAYER cause error
TVF::OUTLAYER EXT via < 0.35 ABUT < 90 SINGULAR
```

The output layer for the preceding example is correctly specified as follows:

```
# CORRECT, use SETLAYER for derivation and OUTLAYER with one argument
TVF::SETLAYER out = EXT via < 0.35 ABUT < 90 SINGULAR
TVF::OUTLAYER out
```

Comments may be specified with tvf::COMMENT; check text comments are specified with tvf::@. See “[Rule checks in Compile-Time TVF](#)” for details.

Tcl Coding Guidelines for custom_check

The tool performs command substitution and evaluates Tcl variables before evaluating the code within the rule check body. Use the following guidelines when using Tcl within the *tvf_body*:

- Do not change the value of global Tcl variables. Global variables may be used within the rule check body, but any changes are local in scope. Therefore, modification of such Tcl variables within the *tvf_body* may result in undefined or undesired behavior.

- Do not declare Tcl variables within the *tvf body*. Usage of locally declared variables results in an error.
- Use the {A B ...} list construction. Do not use the list command because command substitution takes place and the list command is evaluated.

For example, the command [list A B] is evaluated and replaced by “A B” (without quotes), which is generally not the intent, as it is no longer a list. Use the {A B} list construction for proper evaluation.

The following actions take place when the custom_check code body is evaluated:

1. Command substitution and Tcl variable evaluation takes place. Use the preceding coding guidelines to ensure correct evaluation.
2. The evaluated custom_check code body is passed to the generated 3DSTACK rule file, along with other commands converted from 3DSTACK+ to standard 3DSTACK.

No syntax checking is done for the 3DSTACK commands within the custom_check, therefore any errors are reported during the 3DSTACK rule file compilation or at runtime, not during the 3DSTACK+ rule file compilation step.

Examples

Example 1

A custom check between two interacting pad layers.

```
custom_check -check_name c1 -layer_type1 pad -layer_type2 pad -tvf {  
    tvf:@ Internal between layers $_layer_type1 and $_layer_type2  
    tvf:@ RVE Highlight Color: red  
    tvf::SETLAYER c1 = INT $_layer_type1 $_layer_type2 < 0.4  
    tvf::OUTLAYER c1  
}
```

Example 2

A custom check that checks the ratio of the die area to package area.

```
die -die_name die1 -layout {-path ./die1.gds} \  
    -layer_info {-type pad1 -layer 1}  
die -die_name die2 -layout {-path ./die2.gds} \  
    -layer_info {-type pad2 -layer 1}  
die -die_name pack -layout {-path ./pack.gds} \  
    -layer_info {-type ext -layer 2}  
  
stack -die {-name pack} \  
    -tier {  
        -die {-name die1 -invert}  
        -die {-name die2 -placement 1000 0 -invert}  
    }
```

```
custom_check -check_name area_die_pack \
-layer_type1 ext \
-layer_type2 {pad1 pad2 -merge} \
-tvf {
    tvf::@ ratio of interacting tiers greater than 60 percent
    tvf::SETLAYER out = \
        DFM PROPERTY $_layer_type1 $_layer_type2 \
        OVERLAP [= AREA($_layer_type2) / AREA($_layer_type1)] <= 0.6
    tvf::OUTLAYER out
}
```

dangling_ports

Checks for placed ports that do not have physical connectivity on a specified placement or layer type.

Note

 This rule check identifies problems with physical connectivity in the layout that are not reported using netlist-based connectivity checks. The `dangling_ports` check is not intended to highlight missing text. For missing text, use the [no_texts](#) check.

Usage

```
dangling_ports
  -check_name check_name
  -layer_types layer_types_list
  [-comment “comment”]
  [rve_option ...]
```

Arguments

- **-check_name** *check_name*

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each *check_name* must be unique.

- **-layer_types** *layer_types_list*

Argument and value set that specifies a Tcl-formatted list of layer types.

- **-comment** “*comment*”

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- ***rve_option* ...**

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

Use this command to verify that the port placements in the layout have physical connectivity with connectivity layers.

The tool issues 3DSTACK_WARNING_122 for all port objects that are not connected in either the source or layout. The tool also lists all unconnected ports in the log file.

Note

- ❑ If all port placements of the same pin are physically unconnected (for example, they are all are dangling) in the layout and the corresponding port is not connected in the source netlist, they are not highlighted as an error because the layout versus netlist is technically correct.
-

Port-related verification checks are reported by the type of result and the chip, placement, die, or layer to which it belongs. This command allows you to perform port checks without running a full connectivity analysis with the [connected](#) command.

Note

- ❑ If you do not specify a source netlist, Calibre 3DSTACK uses the layout connectivity to check for dangling ports.
-

Examples

This command verifies that all of the ports on the pad layer type are connected to physical geometry.

```
dangling_ports -check_name dangling_pad_check \
    -layer_types pad \
    -comment "ERROR: Unconnected ports detected!"
```

dangling_no_text

Checks for orphan ports that are missing text labels.

Usage

```
dangling_no_text
  -check_name check_name
  -dies die_name_list
  [-comment “comment”]
  [rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.

- **-dies *die_name_list***

Specifies a list of dies in the stack. The tool checks for orphan ports on all interacting layers on the specified dies.

- **-comment “*comment*”**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- ***rve_option* ...**

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

When you specify this command, the tool removes results for the following:

- Die ports that do not have physical connectivity to interacting ports from another die.
- Results with no pins on both sides of the die.
- All standard dangling port results.

For the remaining ports that were not removed, the tool checks the port geometries for text labels. If no labels are found, the tool reports the port as dangling with no text.

Examples

Check for orphan port geometries with missing text labels on the controller die.

```
dangling_ports -check_name orphan_port_cont \
  -dies controller \
```

density

Checks for density constraints on specified placements.

Usage

density

```
-check_name check_name
{-layer_types {placed_layer_type ...}}
-constraint "constraint_expression" [-expression "density_expression"]
[-centers value]
[-direction {up | down | both}]
[-inside { extent | placed_layer}]
[-stack '{' stack_name_list '}']
[-window {wxy | wx wy} [-step {sxy | sx sy}]]
[-window_type {truncate | backup | ignore | wrap}]
[-comment "comment"]
[rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing output results.

- **-layer_types {*placed_layer_type* ...}**

Argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers the layer of the die placement.

- **-constraint "*constraint_expression*"**

Required argument and value set that specifies a constraint. The *constraint_expression* “< 0” is not allowed.

When used without a *density_expression*, data capture windows whose area ratio meets the *constraint_expression* are output. In this case, if the ratio is intended as a percentage, the *constraint_expression* should contain numbers between 0 and 1, where division by 100 has already occurred.

If a *density_expression* is provided, then data capture windows whose expression values meet the *constraint_expression* are output.

- **-expression "*density_expression*"**

Optional argument and value set that performs a calculation involving layer data. If no *density_expression* is provided, then the density is calculated as the ratio of the area of the input layers in a data capture window to the area of the data capture window itself. If you supply a *density_expression*, then the value (based in user units) of the *density_expression* is

computed within each data capture window. If the value of the expression satisfies the *constraint_expression*, then the window is output as usual.

The *density_expression* may involve numbers (including numeric variables), operators, parentheses (), and the functions given in the following table:

Table 3-7. Density Math Functions

Function	Definition
AREA(<i>input_layer</i>)	Area of <i>input_layer</i> in user units of length squared. AREA() gives the area of the data capture window.
SQRT(x)	Square root of x
EXP(x)	Exponential (base e) of x
LOG(x)	Natural logarithm of x
SIN(x)	Sine of x in radians
COS(x)	Cosine of x in radians
TAN(x)	Tangent of x in radians
MIN(<i>expression1,expression2</i>)	Returns the lesser value of the two expressions. The expressions must be of the same form as the primary <i>density_expression</i> , without the square brackets.
MAX(<i>expression1,expression2</i>)	Returns the maximum value of the two expressions. The expressions must be of the same form as the primary <i>density_expression</i> , without the square brackets.

In **Table 3-7**, x is a numeric argument (including numeric variables and other numeric-valued expressions). There is no compile-time or runtime exception checking on the arguments of these functions, and unreasonable or undefined values may result from certain arguments.

The expression must not result in strictly negative values because such values cannot be checked by a *constraint_expression*. Division by zero is defined *not* to satisfy any *constraint_expression*.

Parentheses have the highest precedence for the calculation of numeric values and may be used for grouping of terms.

Operators — Unary operators (+, -, !, ~) require only one numerical argument and all such operators have the same precedence. The unary + and - operators are the usual positive and negative signs. The other unary operators do the following:

- ! operator — Returns 0 (or false) if its argument is non-zero and 1 (or true) if its argument is 0.
- ~ operator — Returns 0 (or false) if the argument is positive and 1 (or true) if the argument is non-positive.

The following table shows some useful combinations of the ~ and ! operators.

Table 3-8. ! and ~ Operator Combinations

x	!(x)	!!(x)	!(x-1)	!!(x-1)	~(x)	~~(x)	~(x-1)	~~(x-1)
3	0	1	0	1	0	1	0	1
2	0	1	0	1	0	1	0	1
1	0	1	1	0	0	1	1	0
0	1	0	0	1	1	0	1	0
-1	0	1	0	1	1	0	1	0
-2	0	1	0	1	1	0	1	0
-3	0	1	0	1	1	0	1	0

Here is a summary of interpretations of the ! and ~ operators:

- !(x) detects whether something is absent (here, 0 means absent).
- !!(x) detects whether something is present (not 0 means present).
- !(x-r) detects the value r.
- !!(x-r) detects everything except r.
- ~(x) detects whether something is absent (here, $x \leq 0$ means absent).
- ~~(x) detects whether something is present ($x > 0$ means present).
- ~(x-r) detects everything $\leq r$.
- ~~(x-r) detects everything $> r$.

[Table 3-7](#) does not show the following relations explicitly, but they can be verified from the table:

$$\sim(x) = \sim\sim(x) \text{ and } \sim\!(x) = \!\!(x).$$

The binary operators $^$, $*$, $/$, $+$, $-$ require two numerical arguments. The $^$ operator is the same as the C language `pow()` function, where $x ^ y$ means x raised to the y power.

The $*$, $/$, $+$, and $-$ operators are multiplication, division, addition, and subtraction, respectively, and have the customary precedence. The $^$ operator has the same precedence as $*$ and $/$.

The binary operators $\|$ (OR) and $\&\&$ (AND) are the same as the C language operators of the same type, except that they have the same precedence as binary $+$ and $-$. They expect numeric-valued inputs. For example:

`AREA(via) && AREA(met)` returns 1 if both inputs are non-zero and 0 otherwise.

`AREA(via) \| AREA(met)` returns 1 if either input is non-zero and 0 otherwise.

- **-centers value**

Optional argument and positive floating number in user units that causes the geometric output of **density** to be squares of value ' value dimensions located at the centers of rectangles that would be output by default.

For further details, refer to the [Density](#) command described in the [Standard Verification Rule Format \(SVRF\) Manual](#).

- **-inside { extent | placed_layer}**

Optional argument and value set that defines a data capture extent within which data capture windows are tiled.

The extent keyword specifies that the data capture extent is the rectangular extent of the input *placed_layer_type* parameters.

The *placed_layer* replaceable specifies that the output of data capture windows is coincident with polygons from *placed_layer*. The *placed_layer* parameter must be an original or derived polygon layer. The data capture extents are the rectangular extents of polygons from *placed_layer*.

If you do not specify any of these keyword sets in the operation, the default data capture extent is the database extent read in at run time. *Calibre 3DSTACK does not read in database layers if they are not required for calculating outputs during the run*. Layers that are not read in do not contribute to the Calibre 3DSTACK database extent.

For further details, refer to the [Density](#) command described in the [Standard Verification Rule Format \(SVRF\) Manual](#).

- **-direction {up | down | both}**

Optional argument and value set that specifies the direction of the check. If the -direction argument is specified, the checks are performed only in the specified direction.

up — The check is only performed from the bottom to the top of the stack. This is the default.

down — The check is only performed from the top to the bottom of the stack.

both — The check is performed in both directions

- **-stack '{ stack_name_list '}**

Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the -stack argument is specified, the checks are applied only to specified stack(s).

- **-window {wxy | wx wy}**

Optional argument and value set that specifies the dimensions of the data capture window within which the density check is to occur. The choices are:

-window *wxy* — Specifies a square window with a height and width of *wxy* user units. The parameter *wxy* must evaluate to a positive real number.

-window *wx wy* — Specifies a rectangular window with a height of *wy* user units and a width of *wx* user units. The parameters *wx* and *wy* must evaluate to positive real numbers.

The windows are tiled across a data capture extent, which is determined by the keywords specified with the operation. Windows that meet the ***constraint_expression*** are output. Windows that are written to the DRC Results Database are merged.

If you do not specify this keyword set in the statement, the default window size is either the extent of the layout database read in at runtime, or is defined by the -inside condition when one is specified. Calibre 3DSTACK does not read in database layers if they are not required for calculating outputs during the run. If the window is larger than the boundary in the x-direction or y-direction, then the window is truncated to the boundary's dimensions in the appropriate directions.

For further details, refer to the [Density](#) command described in the *SVRF Manual*.

- **-step {*sxy | sx sy*}**

Optional argument and value set that is specified with the -window argument. Specifies the grid increment for which data capture windows are tiled within the data capture extent. The choices are:

sxy — Specifies that the windows are tiled to the right and up in increments of *sxy* user units. The parameter *sxy* must evaluate to a positive real number.

sx sy — Specifies that the windows are tiled incrementally to the right *sx* user units and up *sy* user units. The parameters *sx* and *sy* must evaluate to a positive real numbers.

If you specify both -step and -window, then the values *sxy*, *sx*, and *sy* must evenly divide into the window values *wxy*, *wx*, and *wy*, respectively.

Default behavior — If you do not specify -step in the statement, the default -step size is defined by the -window dimensions.

Note that -step 3 -1 is interpreted as -step 2, and -step 3(-1) is a compiler error.

The stepped data capture windows start in the lower-left corner of the data capture extent and initially are tiled in *sxy* or *sx* increments to the right. Upon arriving at the right edge of the data capture extent, the capture windows are tiled upward by the *sxy* or *sy* increment and are tiled again starting from the left edge of the data capture extent.

For further details, refer to the [Density](#) command described in the [Standard Verification Rule Format \(SVRF\) Manual](#).

- **-window_type {truncate | backup | ignore | wrap}**

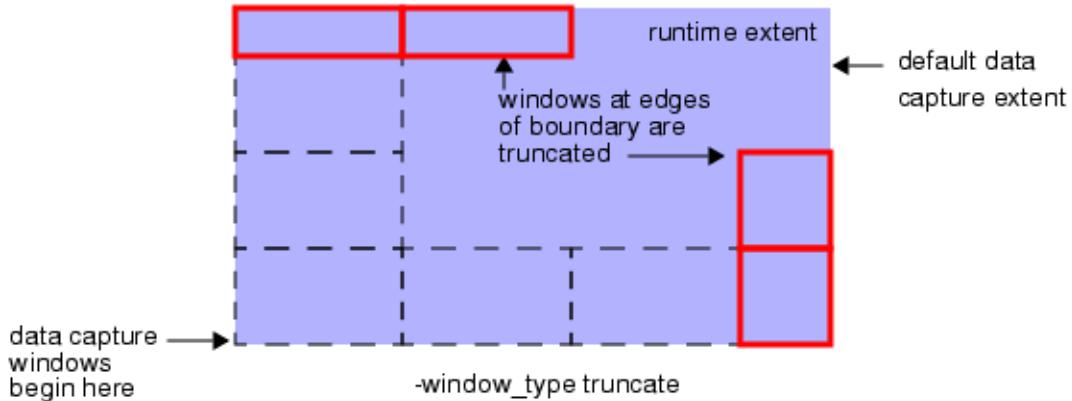
Optional argument and value set that specifies the type of window.

- truncate

Optional keyword that specifies a data capture window is truncated at the limits of the data capture extent. The density calculation in a truncated window is based upon the truncated dimensions of the window. This is the default behavior if you do not specify this keyword. The algorithms for this and related keywords are discussed in

the “Functional Details” section of the [Density](#) command in the *Standard Verification Rule Format (SVRF) User’s Manual*.

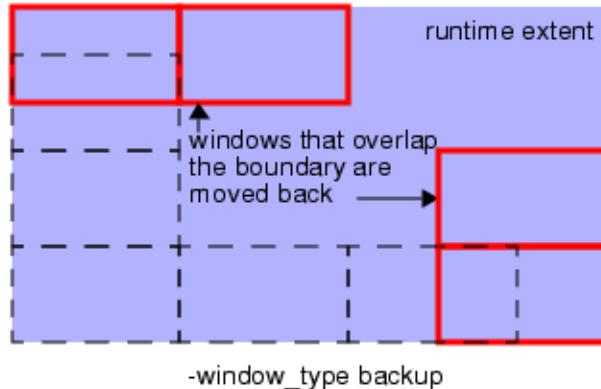
Figure 3-1. density -window_type truncate



- o **backup**

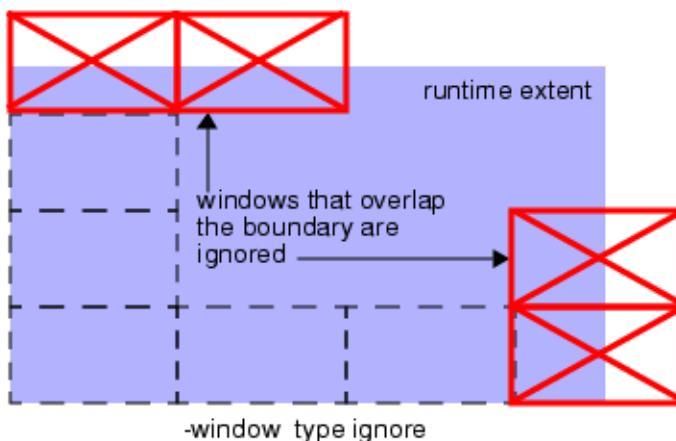
Optional keyword that specifies if a window overlaps the right or top edges of the data capture extent, the window is shifted left or down until it is no longer overlapping the data capture extent. The **density** calculation is then performed after the window has been shifted.

Figure 3-2. density -window_type backup



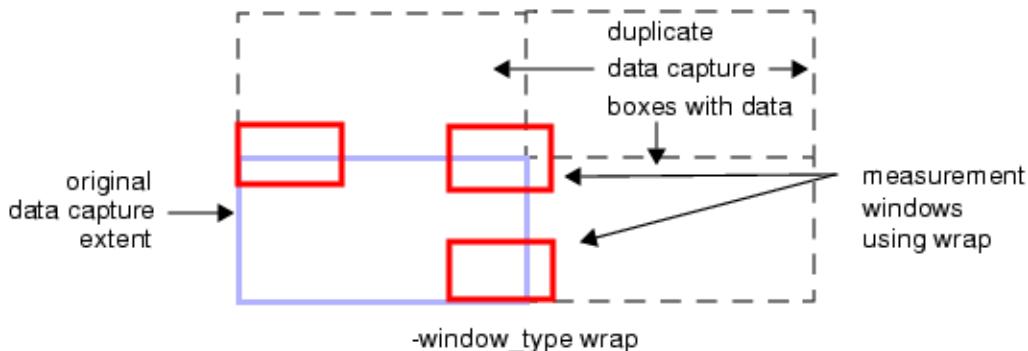
- o **ignore**

Optional keyword that specifies if a window overlaps the right or top edges of the data capture extent, then the window is ignored and no data for that window location is output.

Figure 3-3. density -window_type ignore

- wrap

Optional keyword that specifies if a window overlaps the right or top edges of the data capture extent, then the data capture extent and its data are duplicated and added to the right side or top side of the original bounding box. The density measurement is then taken in the window that intersects the duplicated data capture extent regions.

Figure 3-4. density -window_type wrap

- -comment “*comment*”

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- *rve_option* ...

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

The **density** verification check is typically used to compute the density of an input layer within a specified data capture window, such as in this design rule: “The density of metal2 in every 50 x 50 area of the layout must exceed 25%.” The default **density** function is defined as the

ratio of the total area of the input layers within the data capture window to the area of the window itself. The density is calculated as the window is tiled over a specified data capture extent. Each window placement that meets the constraint value is output as part of a merged layer.

The size of the data capture window, the data capture extent, and how the data capture window is tiled across the capture extent are controlled with options. The *density_expression* option is used to provide a custom calculation instead of the default density calculation.

For further details, refer to the [Density](#) command described in the [Standard Verification Rule Format \(SVRF\) Manual](#). Also see “[Density Best Practices](#)” in the *Calibre Solutions for Physical Verification* manual.

Examples

This example check computes the density of an input layer’s route and bump types:

```
density -check_name den1 -layer_types {route bump} -constraint "<0.1"
```

Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

enclosure

Checks for sufficient separation distance between the exterior-facing sides of the first placement's edges and the interior-facing sides of the second placement's edges.

Usage

enclosure

```
-check_name check_name
{-layer_type1 placed_layer_type1 -layer_type2 placed_layer_type2}
-constraint "constraint_value"
[-direction {up | down | both}]
[-stack '{' stack_name_list '}']
[-comment "comment"]
[rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each *check_name* must be unique.

- **-layer_type1 *placed_layer_type1* -layer_type2 *placed_layer_type2***

Argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers to the layer of the die placement.

For example, if a check is defined between layer_type1 and layer_type2, all the placement layers of layer_type1 that interact with the placement layer of layer_type2 (as defined by the stack) are checked against each other.

- **-constraint "*constraint_value*"**

Specifies the checking distance, which must have an upper bound. The *constraint_value* must conform to the constraint notation described under “[Constraints](#)” in the [Standard Verification Rule Format \(SVRF\) Manual](#).

- **-direction {up | down | both}**

Optional argument and value set that specifies the direction of the check. If the -direction argument is specified, the checks are performed only in the specified direction.

up — The check is only performed from the bottom to the top of the stack. This is the default.

down — The check is only performed from the top to the bottom of the stack.

both — The check is performed in both directions

- **-stack '{' *stack_name_list* '}'**

Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack

with the given name should be specified. If the `-stack` argument is specified, the checks are applied only to specified stack(s).

- `-comment "comment"`

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “`\n`” escape sequence.

- `rve_option ...`

Optional argument and value set that controls how Calibre RVE displays the rule check results. Multiple options are allowed. The permitted values for `rve_option` are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

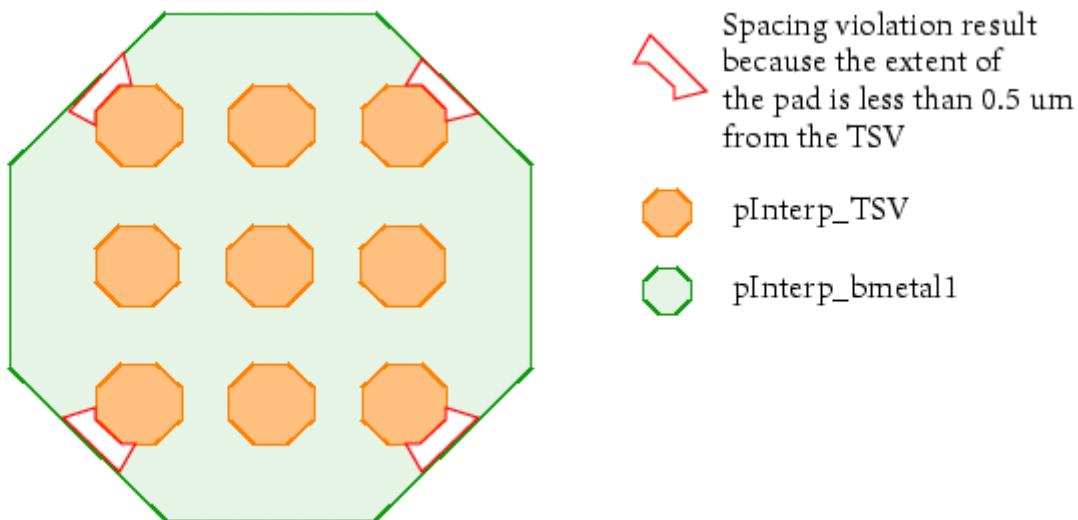
Measures the separation distance between the exterior-facing sides of `placed_layer_type1` edges and the interior-facing sides of `placed_layer_type2` edges, and outputs edge pairs that satisfy the specified `constraint_value`.

The `placed_layer_type1` and `placed_layer_type2` values must be a pair of placed polygon layers.

Examples

This rule example applies to all interacting pad-bump layers in the upward direction (the default) for all stacks. It measures the separation distance between the exterior-facing sides of pad layer edges and the interior-facing sides of bump layer edges.

```
enclosure -check_name enc -layer_type1 pad -layer_type2 bump \
           -constraint "<0.5"
```



Related Topics

- [System and Miscellaneous Commands](#)
- [Assembly Commands](#)
- [Rule Check Commands](#)

external

Checks for sufficient separation distance between the exterior-facing sides of the first placement's edges and the exterior-facing sides of the second placement's edges.

Usage

external

```
-check_name check_name
{-layer_type1 placed_layer_type1 [-layer_type2 placed_layer_type2]}

-constraint "constraint_value"
[-direction {up | down | both}]
[-stack '{' stack_name_list '}']
[-comment "comment"]
[rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each **check_name** must be unique.

- **-layer_type1 *placed_layer_type1* [-layer_type2 *placed_layer_type2*]**

Argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers to the layer of the die placement.

For example, if a check is defined between layer_type1 and layer_type2, all the placement layers of layer_type1 that interact with the placement layer of layer_type2 (as defined by the stack) are checked against each other.

If the check has only one layer type specified, the check is done on all placement layers of that type.

- **-constraint "*constraint_value*"**

Required argument and value set that specifies the checking distance, which must have an upper bound. The **constraint_value** must conform to the constraint notation described under “[Constraints](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

- **-direction {up | down | both}**

Optional argument and value set that specifies the direction of the check. If the -direction argument is specified, the checks are performed only in the specified direction.

up — The check is only performed from the bottom to the top of the stack. This is the default.

down — The check is only performed from the top to the bottom of the stack.

both — The check is performed in both directions

- `-stack '{' stack_name_list '}'`
Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the -stack argument is specified, the checks are applied only to specified stack(s).
- `-comment "comment"`
Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.
- `rve_option ...`
Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

Measures the separation distance between the exterior-facing sides of *placed_layer_type1* edges and the exterior-facing sides of *placement2* edges. If -placement2 is not specified, this command measures the separation distance between the exterior-facing sides of *placed_layer_type1* edges. Edge pairs are output that satisfy the specified *constraint_value*.

The *placement1* and *placement2* values must be a pair of placed polygon layers.

Examples

Example 1

This example rule measures the separation distance between the exterior-facing sides of pad layer edges and the exterior-facing sides of edges of the same layer.

```
external -check_name ext -stack {st1 st2} -layer_type1 pad \
    -constraint "<0.5"
```

Example 2

The following example demonstrates how assemble dies and check for sufficient spacing between chip placements.

First, define the dies used in the assembly:

```
die -die_name Interposer \
    -layout { -path ./design/int.gds -type gdsii -primary ram } ...

die -die_name Mem_1 \
    -layout { -path ./design/ram1.gds -type gdsii -primary ram } \
    -layer_info { -type outline -name ram_extent -layer {267} } ...

die -die_name Mem_2 \
    -layout { -path ./design/ram2.gds -type gdsii -primary ram } \
    -layer_info { -type outline -name ram_extent -layer {267} } ...
```

```
die -die_name Controller \
    -layout { -path ./design/controller.gds -type gdsii -primary top } \
    -layer_info { -type outline -name con_extent -layer {267} } ...
```

Next, arrange the die locations in the stacked assembly:

```
stack -stack_name assembly \
    -die { \
        -name Interposer \
        -placement 0 0 \
        -invert \
    } \
    -z_origin 0 \
    -tier { \
        -die { -name Controller -placement 40 40 } \
        -die { -name Mem_1 -placement 250 100 } \
        -die { -name Mem_2 -placement 280 40 -rotate 90 } \
    }
```

For this example, we want to ensure that no chip comes within 65 microns of the extents of the chip edge (using the predefined chip extent layer)

Tip

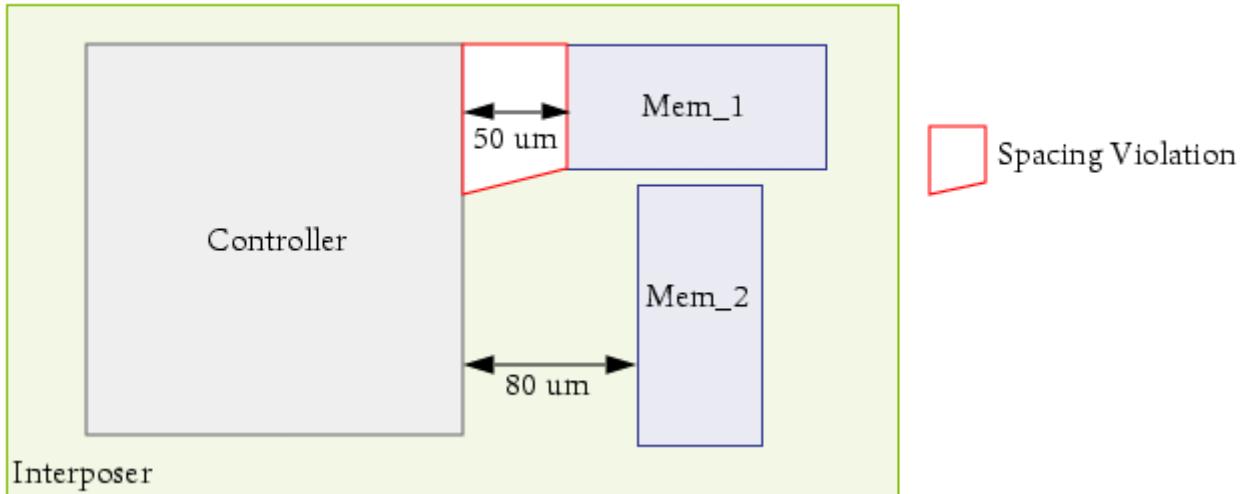
i You can also derive layers to determine the chip extent using the die -svrf option:

```
die -die_name der \
    -layout { -path ./der.gds } \
    -layer_info { -type pad -name pad1 -layer 40 -top } \
    -layer_info { -type pad -name pad2 -layer 41 -top } \
    -layer_info { -type merge -name merge \
    -svrf {MERGE (EXTENT DRAWN pad1 pad2) } }
```

The chip instantiations can be directly referenced in a geometrical check as follows:

```
external -check_name controller_spacing_check -layer_type1 outline \
    -constraint "< 65 REGION" -comment "Chip placement closer than 65 um."
```

Calibre 3DSTACK flags the Mem_1 chip placement because it lies within 65 um of the Controller's extents. Mem_2 is within a safe distance:



Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

extra_ports

Checks for missing ports in the source (or extra ports in the layout) on a specified chip, placement, die or layer.

Usage

```
extra_ports
  -check_name check_name
  -layer_types layer_types_list
  [-comment “comment”]
  [rve_option ...]
```

Arguments

- **-check_name *check_name***
Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.
- **-layer_types *layer_types_list***
Argument and value set that specifies a Tcl-formatted list of layer types.
- **-comment “*comment*”**
Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.
- ***rve_option* ...**
Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

Use this command to verify that the pads in the layout match the ports in the source netlist. Port-related verification checks are reported by the type of result and the chip, placement, die, or layer to which it belongs. This command allows you to perform port checks without running a full connectivity analysis with the [connected](#) command.

The [source_netlist](#) command must be included in order for these results to be produced. Calibre allows you to highlight and cross-reference LNC and SNC ports from the details pane. This result can occur if one or more of the following conditions are true:

- The subcircuit for a placement is missing a pin definition in the source netlist.
- When multiple text labels are found on one pad, Calibre 3DSTACK chooses one of the labels and associates it with that pad during connectivity extraction. As a result, an Extra Port result can occur because the placement port may not be found in the source netlist.

Examples

This command verifies that the layout pads match the source netlist for all pad layer types.

```
extra_ports -check_name pad_text_rule \
             -layer_types pad \
             -comment "ERROR: Extra ports detected!"
```

Related Topics

[missing_ports](#)

[multi_texts](#)

[floating_texts](#)

[no_texts](#)

[connected](#)

floating_pads

Checks for pads of a layer type or stack that do not overlap any other pads from other dies or interposers in the stack.

Note

 The floating_pads command only checks a single interface layer. To check for floating pads for dies with two or more interacting layers, you should apply the overlap command with two layer inputs with an overlap constraint of 0. The additional overlap check will highlight floating interface layers with respect to other layers, which may not be flagged by the standard floating_pads check due to the positioning of multiple interacting layers.

Usage

```
floating_pads
  -check_name check_name
  -layer_type layer_type
  [-stack stack_name_list]
  [-comment “comment”]
  [rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each **check_name** must be unique.

- **-layer_type *layer_type***

Required argument that specifies a single layer type. The layer type is checked with interacting layers from other dies to highlight floating pads.

- **-stack *stack_name_list***

Optional argument and value set that specifies a Tcl-formatted list of stack names. If this option is used, the check only applies to the layer types on the specified list of stacks.

- **-comment “*comment*”**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- ***rve_option* ...**

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

This command enables you to check for unconnected pads on dies and interposers in your layout. If they are not connected to any interacting layers, the tool reports a floating pad error.

Examples

The first command verifies that all pads are connected to interacting geometry on the RAM and IO stacks. The second command verifies floating pads for all pad layer types in the assembly.

```
floating_pads -check_name float_pad_check_stack1_2 \
    -layer_type pad \
    -stack {mem_stack io_stack} \
    -comment "ERROR: Pad not connected!"

floating_pads -check_name float_pad_check_all \
    -layer_type pad \
    -comment "ERROR: Pad not connected!"
```

Related Topics

[offgrid_centers](#)

[overlap](#)

floating_texts

Checks for text labels that are not attached to any geometry on a specified chip, placement, die or layer.

Usage

```
floating_texts
  -check_name check_name
  { -dies die_name_list | -layer_types layer_types_list }
  [-comment “comment”]
  [rve_option ...]
```

Arguments

- **-check_name** *check_name*

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each **check_name** must be unique.

- **-dies** *die_name_list*

Argument and value set that specifies a Tcl-formatted list of die names. This argument set cannot be specified with **-layer_types**.

- **-layer_types** *layer_types_list*

Argument and value set that specifies a Tcl-formatted list of layer types. This argument set cannot be specified with **-dies**.

- **-comment** “*comment*”

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- ***rve_option* ...**

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

Use this command to verify that text labels in the design are attached to physical geometry. If they are not attached, Calibre 3DSTACK reports a floating text error. Text-related verification checks are reported by the type of text result and the chip, placement, die, or layer to which it belongs. This command allows you to perform floating text checks without running a full connectivity analysis with the [connected](#) command.

Examples

The first command verifies that all text is correctly attached to geometry on the controller and interposer die. The second command verifies floating text for all pad layer types.

```
floating_texts -check_name cont_int_pad_text_rule \
-die {controller interposer} \
-comment "ERROR: Text not attached to any geometry!"\n\nfloating_texts -check_name pad_text_rule \
-layer_types pad \
-comment "ERROR: Text not attached to any geometry!"
```

Related Topics

[no_texts](#)

[multi_texts](#)

[connected](#)

internal

Checks for sufficient separation distance between the interior-facing sides of the first placement's edges and the interior-facing sides of the second placement's edges.

Usage

internal

```
-check_name check_name
{-layer_type1 placed_layer_type1 [-layer_type2 placed_layer_type2] }
-constraint "constraint_value"
[-direction {up | down | both}]
[-stack '{' stack_name_list '}']
[-comment "comment"]
[rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.

- **-layer_type1 *placed_layer_type1* [-layer_type2 *placed_layer_type2*]**

Argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers to the layer of the die placement.

For example, if a check is defined between layer_type1 and layer_type2, all the placement layers of layer_type1 that interact with the placement layer of layer_type2 (as defined by the stack) are checked against each other.

If the check has only one layer type specified, the check is done on all placement layers of that type.

- **-constraint "*constraint_value*"**

Specifies the checking distance, which must have an upper bound. The ***constraint_value*** must conform to the constraint notation described under “[Constraints](#)” in the [Standard Verification Rule Format \(SVRF\) Manual](#).

- **-direction {up | down | both}**

Optional argument and value set that specifies the direction of the check. If the -direction argument is specified, the checks are performed only in the specified direction.

up — The check is only performed from the bottom to the top of the stack. This is the default.

down — The check is only performed from the top to the bottom of the stack.

both — The check is performed in both directions

- `-stack '{' stack_name_list '}'`
Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the `-stack` argument is specified, the checks are applied only to specified stack(s).
- `-comment "comment"`
Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.
- `rve_option ...`
Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for `rve_option` are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

Measures the separation distance between the interior-facing sides of *placed_layer_type1* edges and the interior-facing sides of *placed_layer2* edges. If *placed_layer_type2* is not specified, this command measures the separation distance between interior-facing sides of *placed_layer_type1* edges. Edge pairs are output that satisfy the specified *constraint_value*.

Examples

This example rule applies to all interacting pad-to-pad layers in both the upward and downward directions of all stacks. The check measures the separation distance between the interior-facing sides of pad layer edges and the interior-facing sides of interacting pad layer edges.

```
internal -check_name int -direction both \
    -layer_type1 pad -layer_type2 pad -constraint "<0.4"
```

Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

locations

Checks the location of pads and that the shape and text match between source and layout.

Usage

locations

```
-check_name check_name
{-layer_type1 placed_layer_type1 -layer_type2 placed_layer_type2}
[-constraint “constraint_value”]
[-no_case]
[-stack ‘{’ stack_name_list ‘}’]
[-text_only | -overlap_only]
[-comment “comment”] [rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name that is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.

- **-layer_type1 *placed_layer_type1* -layer_type2 *placed_layer_type2***

Argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers to the layer of the die placement.

For example, if a check is defined between layer_type1 and layer_type2, all the placement layers of layer_type1 that interact with the placement layer of layer_type2 (as defined by the stack) are checked against each other.

- **-constraint “*constraint_value*”**

Optional argument and value set that specifies the overlap constraint in terms of a percentage. The constraint must have an upper bound. The *constraint_value* must conform to the constraint notation described under “[Constraints](#)” in the *SVRF Manual*. The default constraint is “<100”.

- **-no_case**

Optional argument that specifies to perform case-insensitive text comparison. For example, when you apply this option, “VDD” in the layout and “vdd” in the source are not considered different.

- **-overlap_only**

Optional argument that specifies to only check for placement geometries that meet the specified *constraint_value*. Geometries that do not overlap or exceed the specified constraint are flagged as “Unmatched Pads” errors. This option cannot be specified with **-text_only**.

- **-text_only**
Optional argument that specifies to only check for placement geometries with text that does not match. If the geometries overlap, but the text does not match, Calibre 3DSTACK reports a “Different Text” error. If the geometries do not overlap, then the geometry is reported as an “Unmatched Pads” error. This option cannot be specified with -overlap_only or -constraint.
- **-stack ‘{’ stack_name_list ‘}’**
Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the -stack argument is specified, the checks are applied only to specified stack(s).
- **-comment “comment”**
Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.
- **rve_option ...**
Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under [“Check Text Override Comments for Calibre 3DSTACK”](#) on page 213.

Description

This command checks that pads overlap correctly, have matching shapes, and have text labels that match between two placement layers. Checking is done in both directions—layout pads are compared to source pads and source pads are compared to layout pads. The source pad information must be given as a placement in the 3DSTACK assembly.

Location checking is only done for pads with attached text, in other words, for placement layers with associated text placement layers specified with the [attach_text](#) command. By default, the [attach_text](#) command causes the placement layer to be part of connectivity extraction, which is only needed for layout placement layers. When using [attach_text](#) with source layers, use the [-no_update](#) argument to specify that the layer is not included in the extracted layout netlist.

The command produces an output layer with result shapes corresponding to the pad. Each result has the following property annotations:

- **Result Type (RT)** — Unmatched Pads or Different Texts.
- **Placement** — The placement name.
- **Overlap** — The percent overlap.
- **Text** — The text for *placed_layer_type1*, if it exists.
- **Text2** — The text for *placed_layer_type2* for Different Texts results.

The center (x,y) coordinates of each reported pad are listed in the Calibre 3DSTACK report file.

Examples

```
locations -check_name locations_check \
-layers_type1 pad \
-layers_type2 bump
```

The results are listed in the report file as follows:

```
*****  
LOCATIONS RULECHECK RESULTS  
*****  
RuleCheck: locations_check ( text only check Selecting unmatched pad  
placements between pCont_CONTROLLER_if and pInterp_INTERP_FRONT_if  
layers.)  
-----  
gnd in placement pCont (-454.49, 10.28)  
wda<4> wda<4>: in placement pCont (-10.79, 490.28)  
adr<6> adr<6> in placement pCont (-10.79, 570.28)
```

missing_ports

Checks for ports in the source netlist that have no corresponding layout pad on a specified chip, placement, die or layer.

Usage

```
missing_ports
  -check_name check_name
  -layer_types layer_types_list
  [-comment “comment”]
  [rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.

- **-layer_types *layer_types_list***

Argument and value set that specifies a Tcl-formatted list of layer types.

- **-comment “*comment*”**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- ***rve_option* ...**

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

Use this command to verify that each pad in the source netlist has a corresponding layout pad. Port-related verification checks are reported by the type of result and the chip, placement, die, or layer to which it belongs. This command allows you to perform port checks without running a full connectivity analysis with the [connected](#) command.

The [source_netlist](#) command must be included in order for these results to be produced. When the result is highlighted, there is no layout object to highlight, so the result properties are displayed in the bottom left corner of the top cell in the assembly.

The result includes the properties SourceNet and MissingPort and these properties are displayed as links in the Result Data Pane (click the links to highlight the source net and port). The Calibre 3DSTACK DFM database and the source netlist must be open in Calibre RVE; see “[Debugging Connectivity Errors in Calibre 3DSTACK](#)” on page 44.

Examples

This command verifies that the layout contains matching pads found in the source netlist for all pad layer types.

```
missing_ports -check_name pad_text_rule \
    -layer_types pad \
    -comment "ERROR: The layout is missing ports!"
```

Related Topics

[extra_ports](#)

[multi_texts](#)

[floating_texts](#)

[no_texts](#)

[connected](#)

multi_texts

Checks for multiple text labels attached to the same pad on a specified chip, placement, die or layer. In the case where multiple text labels overlap a pad in the layout, the tool generates a multi-text error and chooses to use one of the text labels attached to the pad for the connectivity analysis. Because the chosen label may not match your design intent, it is important to review and resolve all multi-text errors to ensure that the layout is correct. If you do not resolve the multi-text errors, your connectivity analysis may not be correct.

Note

 If there are multiple placements for one chip and the placements do not use `rename_text` operations, the text results are reported for the first placement only (the other affected placements are mentioned in the generated Calibre RVE check comments).

Usage

```
multi_texts
  -check_name check_name
  { -dies die_name_list | -layer_types layer_types_list }
  [-comment "comment"]
  [rve_option ...]
```

Arguments

- **-check_name *check_name***
Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each `check_name` must be unique.
- **-dies *die_name_list***
Argument and value set that specifies a Tcl-formatted list of die names. This argument set cannot be specified with **-layer_types**.
- **-layer_types *layer_types_list***
Argument and value set that specifies a Tcl-formatted list of layer types. This argument set cannot be specified with **-dies**.
- **-comment "*comment*"**
Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.
- ***rve_option* ...**
Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for `rve_option` are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

Use this command to verify that pads in the design only contain one text label. Multiple text labels on a single pad may indicate a layout connectivity error. Text-related verification checks are reported by the type of text result and the chip, placement, die, or layer to which it belongs. This command allows you to perform multiple text checks without running a full connectivity analysis with the [connected](#) command.

The check verifies that your design does not have pads that have more than one text label (overlapping labels). In certain cases, pads can overlap and create overlapping text labels. In these cases, the following warning message is issued:

```
WARNING: there are multiple text-labels overlapping with the pads on
layer layer.
"connected" check may produce incorrect results!
```

Multi text warnings are not issued if the overlapping text labels are identical.

Examples

The first command verifies that each pad on the controller and interposer die only contains a single text label. The second command performs the same check, but for all pad layer types.

```
multi_texts -check_name cont_int_pad_text_rule \
-die {controller interposer} \
-comment "ERROR: Pads contain multiple text labels!"

multi_texts -check_name pad_text_rule \
-layer_types pad \
-comment "ERROR: Pads contain multiple text labels!"
```

Related Topics

[floating_texts](#)
[no_texts](#)
[connected](#)

no_texts

Checks for missing text labels for pads on a specified chip, placement, die or layer.

Usage

```
no_texts
  -check_name check_name
  {-dies die_name_list | -layer_types layer_types_list }
  [-comment “comment”]
  [rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.

- **-dies *die_name_list***

Argument and value set that specifies a Tcl-formatted list of die names. This argument set cannot be specified with **-layer_types**.

- **-layer_types *layer_types_list***

Argument and value set that specifies a Tcl-formatted list of layer types. This argument set cannot be specified with **-dies**.

- **-comment “*comment*”**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- ***rve_option* ...**

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

Use this command to verify that all pads in the design are correctly labeled with text objects. Text-related verification checks are reported by the type of text result and the chip, placement, die, or layer to which it belongs. This command allows you to perform missing text checks without running a full connectivity analysis with the [connected](#) command.

Examples

The first command verifies that all pads are correctly labeled on the controller and interposer die. The second command verifies pad text for all pad layer types.

```
no_texts -check_name cont_int_pad_text_rule \
          -die {controller interposer} \
          -comment "ERROR: Pads are missing text labels!"
```

no_texts

```
no_texts -check_name pad_text_rule \
    -layer_types pad \
    -comment "ERROR: Pads are missing text labels!"
```

Related Topics

[multi_texts](#)

[floating_texts](#)

[connected](#)

offgrid_centers

Checks for pad centers that are not aligned to a specified grid.

Usage

offgrid_centers

```
-check_name check_name
-layer_type placed_layer_type
-resolution {resolution_value | {x_resolution_value y_resolution_value}}
[-direction {up | down | both}]
[-hint]
[-stack '{' stack_name_list '}']
[-comment "comment"]
[rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.

- **-layer_type *placed_layer_type***

Argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers to the layer of the die placement.

- **-resolution {*resolution_value* | {*x_resolution_value* *y_resolution_value*}}**

Required argument and value set that specifies the grid resolution in microns. Floating-point numbers are allowed. The ***resolution_value*** is applied in both the x and y directions. The ***x_resolution_value*** and ***y_resolution_value*** arguments enable you to specify different resolutions in the respective dimensions.

- **-direction {up | down | both}**

Optional argument and value set that specifies the direction of the check. If the -direction argument is specified, the checks are performed only in the specified direction.

up — The check is only performed from the bottom to the top of the stack. This is the default.

down — The check is only performed from the top to the bottom of the stack.

both — The check is performed in both directions

- **-hint**

Optional keyword that specifies the output objects provide locations of nearby on-grid points. These can be useful for correcting the locations of off-grid objects. The locations of the HINT markers are simply suggestions for on-grid locations.

See “[Offgrid](#)” in the *SVRF Manual* for complete details on this option’s operation.

- **-stack ‘{’ stack_name_list ‘}’**
Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the -stack argument is specified, the checks are applied only to specified stack(s).
- **-comment “comment”**
Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.
- **rve_option ...**
Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

This command detects whether the center points of pad shapes are aligned to a specified grid. The grid used for checking objects is defined by either the **resolution_value** or **x_resolution_value** and **y_resolution_value** parameters. For example, if you specify the resolution as 45, then all relevant points (vertices, endpoints, or center points) of objects are checked to see if their x and y coordinates are on a 45 micron grid.

The functionality is similar to the SVRF [Offgrid](#) operation with the CENTERS option as described in the *Standard Verification Rule Format (SVRF) User’s Manual*.

Examples

This example checks for pads on the “route” layer that are not aligned to a five micron grid:

```
offgrid_centers -check_name off_mem -layer_type route -resolution 5
```

Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

overlap

Checks for sufficient overlap between two placed layers.

Note

 This rule check is not recommended for pad alignment checking if the two pad shapes are of a different size or different shape type (for example, a circle versus a square). Instead, use the centers check.

Usage

overlap

```
-check_name check_name
{-layer_type1 placed_layer_type1 -layer_type2 placed_layer_type2}
-constraint “constraint_value”
[-by_area]
[-direction {up | down | both}]
[-exact]
[-intersection]
[-merge]
[-stack ‘{’ stack_name_list ‘}’]
[-comment “comment”]
[rve_option ...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.

- **-layer_type1 *placed_layer_type1* -layer_type2 *placed_layer_type2***

Argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers to the layer of the die placement.

For example, if a check is defined between ***layer_type1*** and ***layer_type2***, all the placement layers of ***layer_type1*** that interact with the placement layer of ***layer_type2*** (as defined by the stack) are checked against each other.

- **-constraint “*constraint_value*”**

Required argument and value set that specifies the overlap constraint, which must have an upper bound. The ***constraint_value*** must conform to the constraint notation described under “[Constraints](#)” in the *SVRF Manual*.

- **-by_area**

Optional argument that specifies that the **overlap** operation outputs the overlapping area, rather than the percentage of overlap. The constraint must also be an area value. The -exact option cannot be used with -by_area.

- **-direction {up | down | both}**

Optional argument and value set that specifies the direction of the check. If the -direction argument is specified, the checks are performed only in the specified direction.

up — The check is only performed from the bottom to the top of the stack. This is the default.

down — The check is only performed from the top to the bottom of the stack.

both — The check is performed in both directions

- **-exact**

Optional argument that specifies that the percentage **constraint_value** is *not* rounded to the nearest integer. If this argument is not applied, then the constraint is always rounded. For example, if you specify the following:

```
overlap ... -constraint 98.6
```

The tool checks for pads that do not overlap by at least 99%. However, if you apply the -exact option as follows:

```
overlap ... -constraint 98.6 -exact
```

The tool checks for pads that overlap by at least 98.6%.

The -exact option cannot be used with the -by_area option.

- **-intersection**

Optional argument that specifies that the intersecting regions are reported.

- **-merge**

Optional argument that specifies to merge all polygons on the specified layer types. This option is useful for layers with non-manhattan geometries where there are multiple overlapping polygons and 100% overlap matching may not occur due to internal modeling. Applying this option decreases the performance of the overlap command.

- **-stack '{' stack_name_list '}'**

Optional argument and value set that specifies the stack(s) for which the rule check is applied. If this argument set is not specified, the check is applied to all stacks. The stack with the given name should be specified. If the -stack argument is specified, the checks are applied only to specified stack(s).

- **-comment "comment"**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- *rve_option* ...

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

This command measures the percentage of the polygons on *placed_layer_type1* that are common with the polygons of the layers specified in the *placed_layer_type2* value list. The geometry of *placed_layer_type1* that meets the specified *constraint_expression* is the output of the overlap command.

If you specify the -by_area option, the overlap value is the area overlap between the first and secondary placement layers instead of the percentage overlap.

To report the intersecting region between the placement layers, use the -intersection keyword.

Examples

```
overlap -check_name pad_overlap \
-layer_type1 die_bmp \
-layer_type2 int_pad \
-constraint "<100"
-comment "Pad overlap is less than 100%"
```

Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

same_size

Compares the sizes of interacting pads and reports mismatches. Use this command to ensure that the pads of two connected dies are of the same size.

Usage

```
same_size
  -check_name check_name
  -layer_type1 placed_layer_type1
  -layer_type2 placed_layer_type2
  [-comment “comment”]
  [rve_options...]
```

Arguments

- **-check_name *check_name***

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each ***check_name*** must be unique.

- **-layer_type1 *placed_layer_type1* -layer_type2 *placed_layer_type2***

Required argument and value set that specifies the type(s) of the layer for which the check is applied. All geometrical checks are applied on interfacing (interacting) placement layers defined by the stack(s). The placement layer refers the layer of the die placement.

For example, if a check is defined between ***layer_type1*** and ***layer_type2***, all the placement layers of ***layer_type1*** that interact with the placement layer of ***layer_type2*** (as defined by the stack) are checked against each other.

- **-comment “*comment*”**

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- ***rve_option ...***

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for ***rve_option*** are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Description

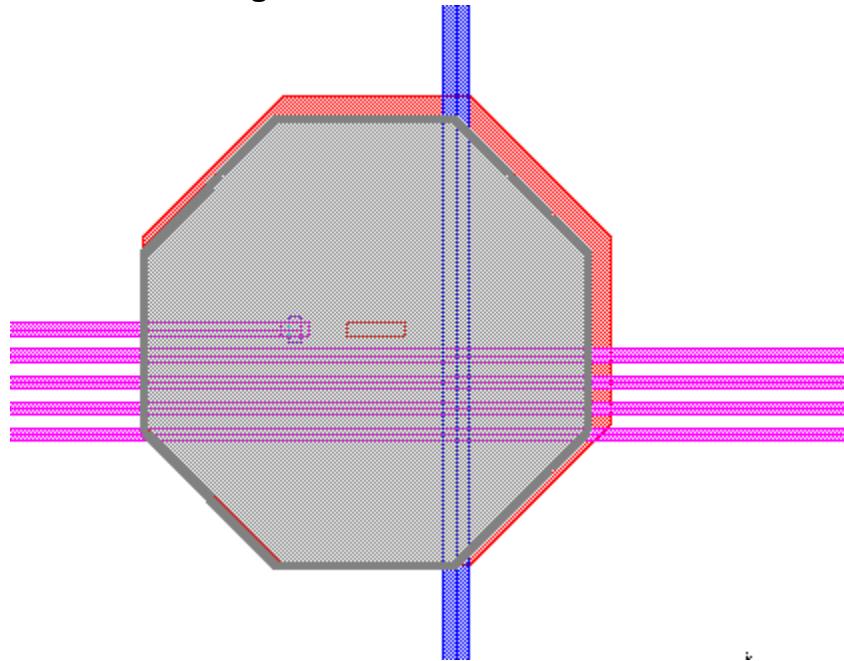
This command measures the size of each of the polygons on ***placed_layer_type1*** that are common with the polygons of the layers specified in the ***placed_layer_type2*** value list. If the size of the polygons are not exactly the same for overlapping polygons on the two layer types, the tool issues an error result.

Examples

Check that the pads are the same size for two stacked dies:

```
same_size -check_name pad_size_check
    -layer_type1 mem_pads
    -layer_type2 interposer_pads
    -comment "Interacting pads are not the same size!"
```

Figure 3-5. same_size Error



select_checks

Selects specified checks to execute during a Calibre 3DSTACK verification.

Usage

select_checks

```
{-check_names {'check_name_pattern [check_name_pattern ...]}'  
| -stack_names {'stack_name_pattern [stack_name_pattern ...]}'  
| -layer_types {'layer_type_pattern [layer_type_pattern ...]} }
```

Arguments

- **-check_names** {'*check_name_pattern* [*check_name_pattern* ...]}'
Argument and value set that specifies a list of rule checks that are executed. The list of checks are specified as Tcl regular expressions. Multiple *check_name_pattern* values are allowed.
- **-stack_names** {'*stack_name_pattern* [*stack_name_pattern* ...]}'
Argument and value set that specifies a list of stacks for which rule checks are not executed. Any checks that apply to the specified stacks are executed. The list of stacks are specified as Tcl regular expressions. Multiple *stack_name_pattern* values are allowed. Must not be specified with the **-check_names** or **-layer_types** argument.
- **-layer_types** {'*layer_type_pattern* [*layer_type_pattern* ...]}'
Argument and value set that specifies a list of layer types for which rule checks are not executed. Any checks that apply to the specified layer types are executed. The list of layer types are specified as Tcl regular expressions. Multiple *layer_type_pattern* values are allowed. Must not be specified with the **-stack_names** or **-check_names** argument.

Description

This command executes only those rule checks that are defined by the *check_name_pattern* argument list. Checks can also be selected using the **-stack_names** and **-layer_types** arguments.

Multiple **select_checks** commands are allowed in the Calibre 3DSTACK rule file. If multiple **select_checks** statements are specified, the list is accumulated as shown in Example 2.

The **-stack_names**, **-layer_types**, and **-check_names** arguments are mutually exclusive.

Examples

Example 1

Specify a single check to run:

```
select_checks -check_names {"int_1"}
```

Example 2

Specify multiple checks to run using either of the following methods:

```
select_checks -check_names { ".*_1" "ext_2" "enc_2" }
```

Alternatively, enter multiple commands:

```
select_checks -check_names { ".*_1" }
select_checks -check_names { "ext_2" }
select_checks -check_names { "enc_2" }
```

Example 3

Specify a single layer type to run (this can result in multiple rule check selection):

```
select_checks -layer_types { "pad" }
```

Example 4

Specify multiple layer types to run:

```
select_checks -layer_types { "interposer" "ram_*" }
```

Example 5

Alternatively, enter multiple commands:

```
select_checks -layer_types { "pad.*" }
select_checks -layer_types { ".*bump.*" }
select_checks -layer_types { "route" }
```

Example 6

Specify single stack to run (this can result in multiple rule check selection):

```
select_checks -stack_names { "st1" }
```

Related Topics

[unselect_checks](#)

unconnected_ports

Reports ports that are both not connected to any other port in the layout and in the source netlist. A source netlist is required.

Usage

```
unconnected_ports
  -check_name check_name
  { {-dies die_name_list} | {-layer_types layer_types_list} }
  [-comment "comment"]
  [rve_option ...]
```

Arguments

- **-check_name** *check_name*

Required argument and value set that specifies a check name, which is used when writing out results. If this command is specified multiple times, each **check_name** must be unique.

- **-dies** *die_name_list*

Argument and value set that specifies a Tcl-formatted list of die names. This argument set cannot be specified with **-layer_types**.

- **-layer_types** *layer_types_list*

Argument and value set that specifies a Tcl-formatted list of layer types. This argument set cannot be specified with **-dies**.

- **-comment** “*comment*”

Optional argument and value set that specifies a rule check comment. You can specify multi-line comments using the “\n” escape sequence.

- ***rve_option* ...**

Optional argument and value set that controls how Calibre RVE displays rule check results. Multiple options are allowed. The permitted values for *rve_option* are described in detail under “[Check Text Override Comments for Calibre 3DSTACK](#)” on page 213.

Examples

```
unconnected_ports -check_name check_name dies cont_1
```

unselect_checks

Selects specified checks not to execute during a Calibre 3DSTACK verification.

Usage

unselect_checks

```
{-check_names {'check_name_pattern [check_name_pattern ...]}'  
| -stack_names {'stack_name_pattern [stack_name_pattern ...]}'  
| -layer_types {'layer_type_pattern [layer_type_pattern ...]} }
```

Arguments

- **-check_names** {'*check_name_pattern* [*check_name_pattern* ...]}'
Required argument and value set that specifies a list of rule checks that are excluded from the verification run. The list of checks are specified as Tcl regular expressions. Multiple *check_name_pattern* values are allowed.
- **-stack_names** {'*stack_name_pattern* [*stack_name_pattern* ...]}'
Argument and value set that specifies a list of stacks for which rule checks are not executed. Any checks that apply to the specified stacks are executed. The list of stacks are specified as Tcl regular expressions. Multiple *stack_name_pattern* values are allowed. Must not be specified with the **-check_names** or **-layer_types** argument.
- **-layer_types** {'*layer_type_pattern* [*layer_type_pattern* ...]}'
Argument and value set that specifies a list of layer types for which rule checks are not executed. Any checks that apply to the specified layer types are executed. The list of layer types are specified as Tcl regular expressions. Multiple *layer_type_pattern* values are allowed. Must not be specified with the **-stack_names** or **-check_names** argument.

Description

This command removes specified rule checks from the Calibre 3DSTACK verification run that are defined with the *check_name_pattern* value list.

Checks can also be unselected using the **-stack_names** and **-layer_types** arguments.

The syntax and usage is similar to the [DFM Unselect Check](#) command described in the [Standard Verification Rule Format \(SVRF\) Manual](#). Multiple **unselect_checks** commands are allowed in the rule file. If multiple **unselect_checks** statements are specified, the list is accumulated as shown in Example 2.

The **-stack_names**, **-layer_types**, and **-check_names** arguments are mutually exclusive.

Examples

Example 1

Unselect a single check:

```
unselect_checks -check_names {"int_1"}
```

Example 2

Unselect multiple checks using either of the following methods:

```
unselect_checks -check_names {".*_1" "ext_2" "enc_2"}
```

Alternatively, enter multiple commands:

```
unselect_checks -check_names {".*_1"}  
unselect_checks -check_names {"ext_2"}  
unselect_checks -check_names {"enc_2"}
```

Example 3

Specify a single layer type to run (this can result in multiple rule check selection):

```
unselect_checks -layer_types {"pad"}
```

Example 4

Specify multiple layer types to run:

```
unselect_checks -layer_types {"interposer" "ram_*"}
```

Example 5

Alternatively, enter multiple commands:

```
unselect_checks -layer_types {"pad.*"}  
unselect_checks -layer_types {".*bump.*"}  
unselect_checks -layer_types {"route"}
```

Example 6

Specify single stack to run (this can result in multiple rule check selection):

```
unselect_checks -stack_names {"st1"}
```

Related Topics

[select_checks](#)

3dstack_block

Used in the 3DSTACK+ extended syntax file.

Defines a code block in which standard Calibre 3DSTACK rule file format commands can be specified. This command allows you to directly pass standard 3DSTACK syntax commands to the Calibre 3DSTACK run.

Usage

3dstack_block '{' *body* '}'

Arguments

- '{' *body* '}'

The **body** is a required set of arguments that consist of commands from the standard Calibre 3DSTACK syntax. You must enclose the commands in braces {}.

If a list is used within the command body, braces ({...}) should be used to enclose the list to ensure correct evaluation. Do not use the list syntax ([list ...]).

Description

Use this command to define custom commands. The 3dstack_block command allows you to specify commands from the standard 3DSTACK syntax language in the extended 3DSTACK+ rule file. See “[Standard Calibre 3DSTACK Syntax Commands](#)” on page 357

Multiple 3dstack_block commands are allowed in the rule file.

Some configuration information can be specified with the [config](#) command rather than by using standard Calibre 3DSTACK commands in a 3dstack_block statement; this includes the layout_primary, source_netlist, report, ignore_trailing_chars, and export_connectivity commands.

Evaluation Details

The tool performs command substitution and evaluates Tcl variables before evaluating the code within the 3dstack_block. Use the following guidelines when using Tcl within the 3dstack_block:

- Do not change the value of global Tcl variables. Global variables may be used within the code body, but any changes are local in scope. Therefore, modification of such Tcl variables within the **3dstack_block** may result in undefined or undesired behavior.
- Do not declare Tcl variables within the **3dstack_block**. Usage of locally declared variables results in an error.
- Use the {A B ...} list construction. Do not use the list command because command substitution takes place and the list command is evaluated.

For example, the command [list A B] is evaluated and replaced by “A B” (without quotes), which is generally not the intent, as it is no longer a list. Use the {A B} list construction for proper evaluation.

The following actions take place when the 3dstack_block code body is evaluated:

1. Command substitution and Tcl variable evaluation takes place. Use the preceding coding guidelines to ensure correct evaluation.
2. The evaluated 3dstack_block code body is passed to the generated 3DSTACK rule file, along with other commands converted from 3DSTACK+ to standard 3DSTACK.

No syntax checking is done for the 3DSTACK commands within the 3dstack_block, therefore any errors are reported during the 3DSTACK rule file compilation or at runtime, not during the 3DSTACK+ rule file compilation step.

Examples

```
3dstack_block {
    export_template -chip c2.oas -system OASIS
    run -drc -directory ./DRC -rule_deck ./n45_m.rules \
        -run_options "-turbo -hier"
    source_filter -chip interposer -subckt tsv -short_pins {bottom top}
    map_placement -placement Memory_1 -source memory_block_1
    ignore_pin -placement p1 -pin {VSS}
    import_text_labels -chip stack_die -file lvsText_dieWrapper.txt
    tvf_block creating_logic_to_memory_connects {
        tvf::SETLAYER l_logic_ltsv_to_memory = l_ltsv NOT i_out_M1_tsv;
    } -export_layers {l_logic_ltsv_to_memory} -attach_to_placement l
    set_auto_rve_show_layers YES
    set_rve_cto_file -file ./custom_rve_settings.cto
}
```

Check Text Override Comments for Calibre 3DSTACK

Check text override comments allow additional control over how results are displayed in Calibre RVE. Check text override comments can be specified as options to certain rule check commands.

Usage

`rule_check check_arguments [rve_option ...]`

Note

 The `rve_option` argument can be replaced by one or more of the argument and value sets described under Arguments. The `rule_check` and `check_arguments` parameters are one of the following verification commands which support check text override comments. For example, [connected](#), [centers](#), [density](#), [enclosure](#), [external](#), [internal](#), [offgrid_centers](#), and [overlap](#).

Arguments

- `-set_rve_highlight_index index`

Applies a Calibre RVE Highlight Index comment to the check. The `index` value is an integer that specifies the highlight layer index used for the rule check. See “[RVE Highlight Index](#)” in the *Calibre RVE User’s Manual* for more details.

- `-set_rve_highlight_color color`

Applies a Calibre RVE Highlight Color comment to the check. The `color` value (for example, blue) specifies the layer color used when the rule check is highlighted in supported layout viewers from Calibre RVE. See “[RVE Highlight Color](#)” in the *Calibre RVE User’s Manual* for more details.

- `-set_rve_priority priority`

Applies a Calibre RVE Priority comment to the check. The `priority` value is an integer that sets the display priority for a rule check. The rule check priority is displayed in the tree view of Calibre RVE and can be used to sort the rule checks. See “[RVE Priority](#)” in the *Calibre RVE User’s Manual* for more details.

- `-set_rve_show_layers {layer_list | AUTO}`

Specifies the layers that are shown in the layout editor when highlighting a result from the rule check. The `layer_list` parameter is a list of layers that are visible when highlighting a rule check result in the layout editor (unspecified layers are hidden).

The AUTO keyword instructs Calibre RVE to only show the input layers that were used to derive the rule check.

This command is only supported by Calibre DESIGNrev, Calibre WORKbench, and Cadence Virtuoso.

In order to use this functionality, you must enable the Calibre RVE option “Only show check-dependent layers while highlighting results (hides other layers)” on the

Setup > Options > Highlighting pane in the DRC/DFM Highlighting area. See “[RVE Show Layers](#)” in the *Calibre RVE User’s Manual* for more details.

- `-set_rve_link doc [-anchor_tag tag] [-display_text text]`

Applies a Calibre RVE Link comment to the check. This option creates a hyperlink to a filename specified by the *doc* value. The link is displayed in the check text pane of Calibre RVE for the specified rule check. The `-set_rve_link` command may be specified with the following optional arguments:

- `-anchor_tag tag`

An optional keyword and parameter that specifies a named destination tag within the document.

- `-display_text text`

An optional keyword and parameter that specifies the display text for the hyperlink. If this keyword and parameter is not included, the hyperlink text displays the full path.

See “[RVE Link](#)” in the *Calibre RVE User’s Manual* for more details.

Description

The usage of check text override comments is described in detail in the section “[DRC Rule Check Comments for Calibre RVE](#)” in the *Calibre RVE User’s Manual*.

The CTO commands can be used to set the priority of text related checks during connectivity extraction.

Examples

Example

If you want to change the default Calibre RVE highlight color for a rule check, you may specify the `set_rve_highlight_color` option as follows:

```
external -check_name spacing_check -placement1 ${place1}_RAM \
    -placement2 ${controller}_CONTROLLER -constraint "< 70 REGION" \
    -comment "Spacing 70 RAM-CONTROLLER." -set_rve_highlight_color red
```

When this check is selected in the Calibre 3DSTACK results database using Calibre RVE, the new highlight color option is listed in the check comments section. When the result is highlighted, the result layer in the layout editor is colored red.

Example

Multiple check comment options may also be specified as follows:

```
centers -check_name centers_interposer -placement1 m_mtsv \
-constraint "<=5" -comment "Pad centers 5um" \
-set_rve_highlight_color yellow \
-set_rve_priority 1 \
-set_rve_link ./docs/spec_1.1.5.txt -display_text "Pad Specification"
```

Example

To assign priorities to text-related checks, include a CTO in your 3DSTACK rule file as follows:

```
set_rve_cto_file -file 3dstack.cto
```

Where the *3dstack.cto* file contains statements similar to the following:

```
my_connected_check_1
RVE Priority: 1

No_Text
RVE Priority: 2

Floating_Text
RVE Priority: 2

Multi_text
RVE Priority: 2
```

Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

Chapter 4

Calibre 3DSTACK Utilities

Calibre 3DSTACK includes utilities that allow you to convert other layout or CSV files into supported formats.

The spreadsheet conversion utilities and AIF conversion utilities are commands within the Calibre YieldServer tool.

System Netlist Generator	218
AIF Converter Reference	278
Spreadsheet Converters	285

System Netlist Generator

The System Netlist Generator is included in the Calibre installation package and is used to create a source netlist for a multi-chip design. The tool is invoked separately to Calibre 3DSTACK and includes its own set of commands and a graphical interface.

System Netlist Generator Flow and Invocation	220
Workflow	220
sng.....	222
System Netlist Generator Graphical User Interface	227
System Netlist Generator GUI Tips	227
Main Window	230
Properties Panel	233
Set Net Name Template	234
System Netlist Generator Configuration File Format	235
PLACEMENTS	236
CONNECTIVITY.....	237
EXPORTS.....	239
System Netlist Generator Commands	240
sng::create_journal_file	242
sng::export_db	243
sng::export_netlist.....	244
sng::new_db	245
sng::open_db.....	246
sng::save_db	247
sng::filter.....	248
sng::get_begin.....	250
sng::get_buses.....	252
sng::get_chips	253
sng::get_nets	254
sng::get_pins	255
sng::get_placements	256
sng::get_ports	257
sng::get_property	259
sng::get_property_names	260
sng::incr	262
sng::set_property.....	263
sng::sort	264
sng::add_pin	266
sng::connect	267
sng::create_chip	269
sng::create_placement.....	270
sng::import_chip	271
sng::remove_chip	273
sng::remove_pin	274

sng::remove_placement	275
sng::set_net_name_template.....	276
sng::unconnect	277

System Netlist Generator Flow and Invocation

The System Netlist Generator allows you to easily create source netlists for complex stacked systems. It supports batch and interactive modes.

Calibre Interactive can invoke the System Netlist Generator to create a source netlist before a verification run. For more information on Calibre Interactive, see “[Creating a Source Netlist](#)” on page 54.

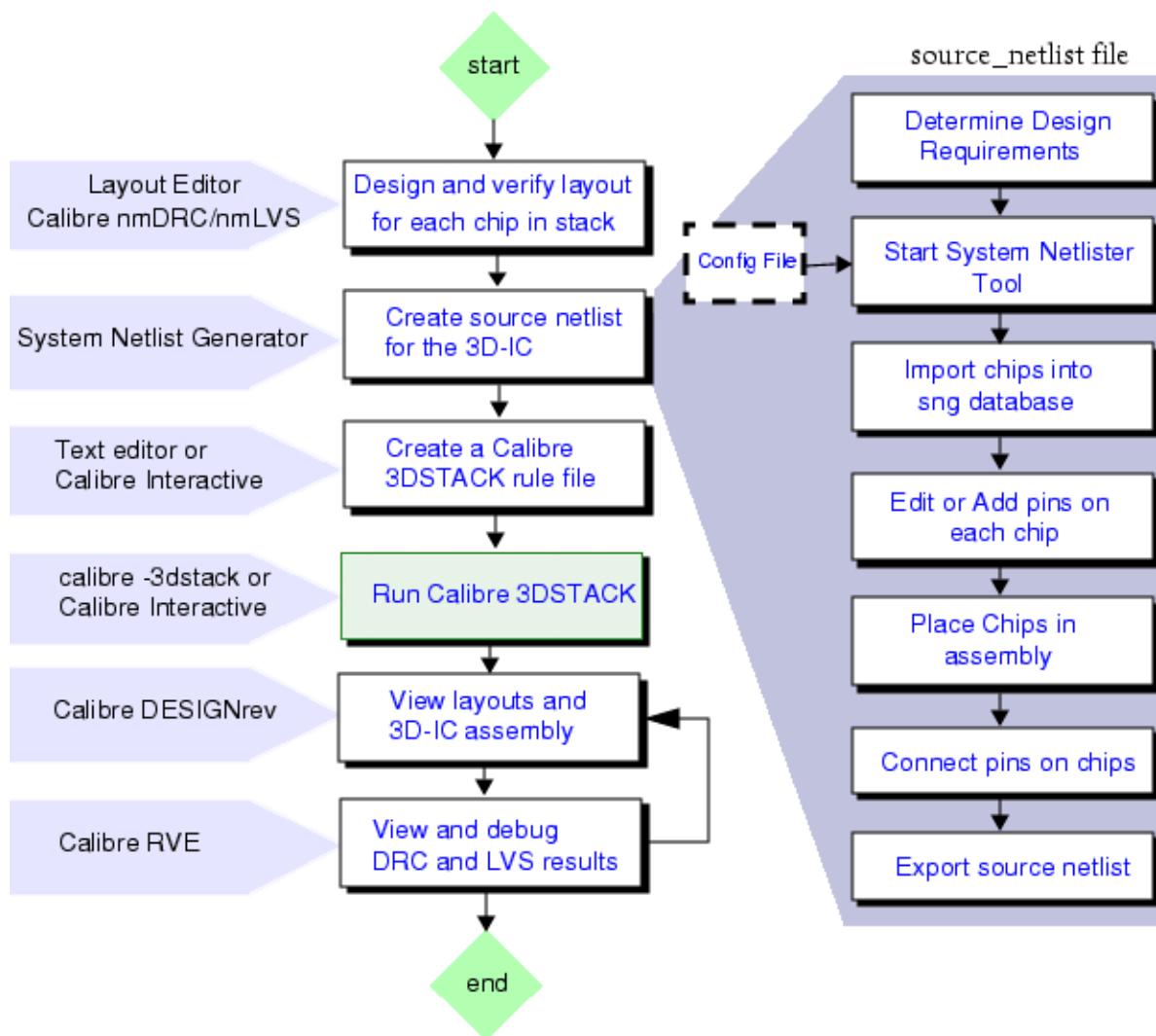
Workflow	220
sng	222

Workflow

The System Netlist Generator can be invoked from Calibre Interactive to generate a netlist from an existing rule file or it can be used in a stand-alone flow.

The System Netlist Generator workflow is illustrated in the following figure.

Figure 4-1. System Netlist Generator Workflow



sng

Invocation arguments for the System Netlist Generator control the startup mode and operation of the tool.

Usage

Interactive (GUI) Invocation

```
sng [-gui] [-project project_file] [-log_dir log_file_directory]
```

Batch Script Invocation

```
sng -batch -script script_file [-script_args script_args] [-log_dir log_file_directory]
```

Hierarchical Source Netlist Creation

```
sng -batch{-stitch -assembly assembly_netlist -top_cell top_cell -die_map die_map_file  
[-run_dir run_dir] [-output output_file] [-rename_cells]}
```

Arguments

- **-batch**

Required keyword for batch mode and hierarchical source netlist creation. This keyword cannot be specified with **-gui**.

- **-script *script_file***

Keyword and path to a script file that contains batch Tcl commands for the System Netlist Generator. This argument is required for batch mode invocation and cannot be specified with **-gui**.

- **-script_args *script_args***

Keyword and list of arguments that are passed to the Tcl **script_file**. The passed arguments can be accessed in the script using standard predefined variables, such as argv0, argv, and so on. This argument cannot be specified with the **-gui** argument.

- **-gui**

Optional keyword that invokes the System Netlist Generator graphical user interface. This argument cannot be specified with the **-batch** argument. If you invoke **sng** without any arguments, the GUI is automatically loaded.

- **-project *project_file***

Optional keyword and path to an existing project. The *project_file* database opens when you invoke the System Netlist Generator GUI. This argument cannot be specified with the **-batch** argument.

- **-log_dir *log_file_directory***

Optional keyword and path to a directory that contains generated log files. By default, a directory named *sng_log_dir* is created where you launch the tool.

- **-stitch**

Generates a hierarchical netlist from specified input netlists. This argument set must be used with **-batch**.

- **-assembly *assembly_netlist***

Required path to the assembly netlist file. The netlist file contains instantiations of all chips in the stack and the top cell for the assembly.

- **-top_cell *cell_name***

Required name of the top cell in the assembly netlist file for the stitching flow.

- **-die_map *die_map_file***

Required file that specifies die netlist files that belong to chip instantiations (subcircuits) in the assembly netlist file. Each line in the file contains a single mapping and the fields are space-separated. You must enter the fields in the order as follows:

chip_name *format* *file_path* *cell_name*

chip_name — Name of the chip subcircuit in the assembly netlist file.

format — Format of the input netlist file. This must be SPICE.

file_path — Path to the netlist file for the chip.

cell_name — Name of the cell in the specified netlist file that represents the chip in the assembly.

The following is an example of a die map file:

```
#chip_name      format      file_path      cell_name
controller     SPICE       ./design/proc.spi   control_64b
ram_1          SPICE       ./design/ram.spi    myram
ram_2          SPICE       ./design/ram.spi    myram
```

The first column corresponds to placed chips in the 3D-IC assembly netlist. The second column indicates the format of the individual netlist files. The third column specifies the path to each netlist file. Finally, the last column maps the cell in the specified netlist file to the chip in the assembly netlist.

- **-run_dir *run_dir***

Optional path to a directory to which all output files are saved. The default run directory is the current working directory.

- **-output *output_file***

Optional path for the generated netlist file. The default netlist file is named *sng_stitch.spi*.

- **-rename_cells**

Optional keyword that instructs the System Netlist Generator to rename all cells in the individual die netlists. The cell names in the die netlists are prefixed with the chip name in the generated netlist.

Description

The System Netlist Generator application is located in the `$CALIBRE_HOME/bin` directory of the Calibre installation tree. You can invoke the tool in batch or interactive modes. You can specify a script with a set of supported Tcl commands and [System Netlist Generator Commands](#) in batch mode.

You can create a hierarchical netlist using the `-batch -stitch` keywords and related arguments.

Examples

GUI Invocation Example

Invoke the System Netlist Generator graphical user interface:

```
sng -gui
```

or

```
sng
```

Batch Invocation Example

Invoke the System Netlist Generator in batch mode and execute a set of Tcl and sng commands:

```
sng -batch -script system_netlist.tcl
```

The following is an example of a System Netlist Generator script:

```
set db [sng::new_db]
sng::create_chip_db -chip_name "cont_chip" -cell_name "cont"
sng::add_pin db -pin_name "select" -chip_name "cont" \
    -type "INPUT"
set iter [sng::get_pins db -chip_name cont_chip]
test {$iter ne ""}
puts [sng::get_property_names $iter]
sng::add_pin db -pin_name "enable" -chip_name "ram_chip" \
    -type "INPUT" -bus_name readdata -bus_order 1

while {$iter ne ""} {
    puts "property names = [sng::get_property_names $iter]"
    puts [sng::get_properties $iter]
    sng::incr iter 1
}
```

Hierarchical Netlist Generation Example

This example demonstrates using the batch mode to create a hierarchical netlist. The following input files are used in this example:

- *bb_netlist.spi* — The black box assembly netlist file created by a 3DSTACK assembly run.
- *base.spi* and *stack.spi* — White box versions of cells in the assembly.

- *chips_to_netlists.map* — The die mapping file that specifies the netlists to include and how the cells in the netlist correspond to the subcircuits in the *bb_netlist.spi* assembly.

bb_netlist.spi (the black box assembly netlist file):

```
.SUBCKT top_assembly
Xpointer 2 3 5 8 1 4 6 7 inter
.ENDS top_assembly

.SUBCKT base bclk_1 bclk_2 bgnd bin_1 bin_2 bout_1 bout_2 bvdd
.ENDS base

.SUBCKT inter ipad0 ipad1 ipad2 ipad3 ipad4 ipad5 ipad6 ipad7
Xpbase ipad1 ipad5 ipad4 ipad2 ipad3 ipad6 ipad7 ipad0 base
Xpstack ipad1 ipad5 ipad4 ipad2 ipad3 ipad6 ipad7 nIsolated0 stack
.ENDS inter

.SUBCKT stack sclk_1 sclk_2 sgnd sin_1 sin_2 sout_1 sout_2 svdd
.ENDS stack
```

base.spi (white box version of cell base):

```
.SUBCKT base bclk_1 bclk_2 bgnd bin_1 bin_2 bout_1 bout_2 bvdd
RR1 A B rap w=3.0 l=3.0 r=21.0m
RR1 C D rap w=3.0 l=3.0 r=21.0m
.ENDS base
```

stack.spi (white box version of cell stack):

```
.SUBCKT stack sclk_1 sclk_2 sgnd sin_1 sin_2 sout_1 sout_2 svdd
RR1 A B rap w=3.0 l=3.0 r=21.0m
RR1 C D rap w=3.0 l=3.0 r=21.0m
.ENDS stack
```

chips_to_netlists.map (the configuration file):

#chip_name	format	file_path	cell_name
base	SPICE	base.spi	base
stack	SPICE	stack.spi	stack

The following command generates the hierarchical netlist *hier.spi*:

```
sng -batch -stitch -assembly bb_netlist.spi \
-top_cell top_assembly -die_map chips_to_netlists.map \
-output complete_whitebox_output.spi
```

The netlist *complete_whitebox_output.spi* has the following contents:

```
.SUBCKT top_assembly
Xpinter 2 3 5 8 1 4 6 7 inter
.ENDS top_assembly

.SUBCKT inter ipad0 ipad1 ipad2 ipad3 ipad4 ipad5 ipad6 ipad7
Xpbase  ipad1 ipad5 ipad4 ipad2 ipad3 ipad6 ipad7 ipad0 base
Xpstack  ipad1 ipad5 ipad4 ipad2 ipad3 ipad6 ipad7 nIsolated0 stack
.ENDS inter

.SUBCKT stack_0 sclk_1 sclk_2 sgnd sin_1 sin_2 sout_1 sout_2 svdd
RR1 A B rap w=3.0 l=3.0 r=21.0m
RR1 C D rap w=3.0 l=3.0 r=21.0m
.ENDS stack_0

.SUBCKT stack sclk_1 sclk_2 sgnd sin_1 sin_2 sout_1 sout_2 svdd
Xstack_0 sclk_1 sclk_2 sgnd sin_1 sin_2 sout_1 sout_2 svdd stack_0
.ENDS stack

.SUBCKT base_0 bclk_1 bclk_2 bgnd bin_1 bin_2 bout_1 bout_2 bvdd
RR1 A B rap w=3.0 l=3.0 r=21.0m
RR1 C D rap w=3.0 l=3.0 r=21.0m
.ENDS base_0

.SUBCKT base bclk_1 bclk_2 bgnd bin_1 bin_2 bout_1 bout_2 bvdd
Xbase_0 bclk_1 bclk_2 bgnd bin_1 bin_2 bout_1 bout_2 bvdd base_0
.ENDS base
```

The “inter” subckt has no white box data and the sng command gives following warning for this flow: “Warning: Assembly netlist cell “inter” is not matched to any individual cell.”

System Netlist Generator Graphical User Interface

The graphical interface for the System Netlist Generator includes most of the batch command functionality.

The panels available in the System Netlist Generator GUI are described in the following sections.

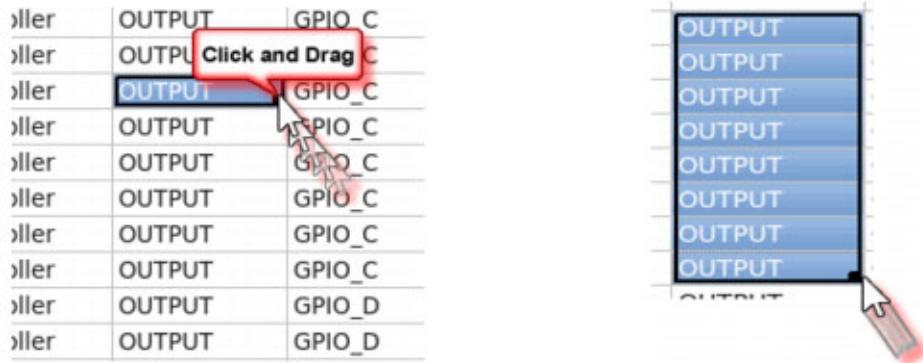
System Netlist Generator GUI Tips	227
Main Window	230
Properties Panel	233
Set Net Name Template	234

System Netlist Generator GUI Tips

The graphical interface includes a number of features that allow you to quickly edit your design.

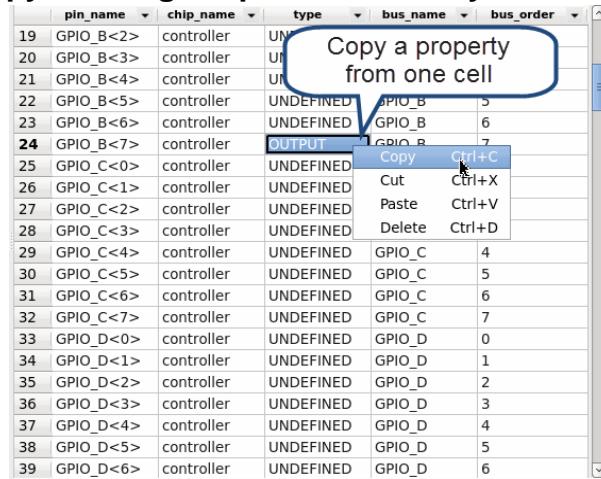
When you select a cell, a small rectangle appears in the cell's bottom-right corner. Click this rectangle to select multiple cells as shown in the following figure.

Figure 4-2. Select Multiple Cells by Clicking and Dragging the Cell Corner



Edit properties by entering values for each field or by copying and pasting values from existing fields. [Figure 4-3](#), [Figure 4-4](#), and [Figure 4-5](#) demonstrate how you can easily modify multiple cell properties at once.

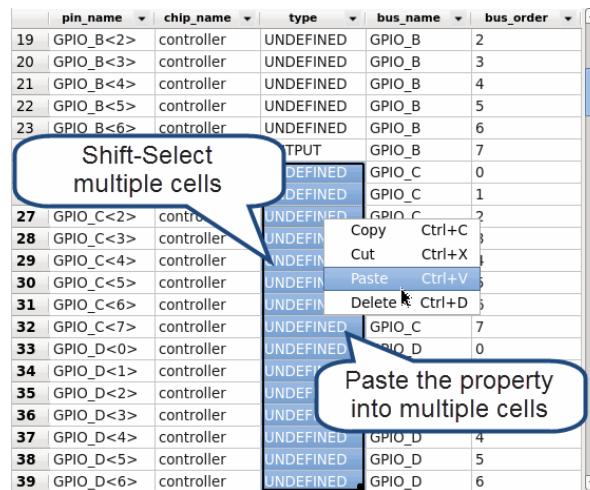
Figure 4-3. Copy Existing Properties in the System Netlist Generator



A screenshot of a spreadsheet-like interface for the System Netlist Generator. A context menu is open over a row labeled '24'. The menu items are: Copy (highlighted), Cut, Paste, and Delete. A callout bubble says 'Copy a property from one cell'.

pin_name	chip_name	type	bus_name	bus_order
19	GPIO_B<2>	controller	UNDEFINED	
20	GPIO_B<3>	controller	UNDEFINED	
21	GPIO_B<4>	controller	UNDEFINED	
22	GPIO_B<5>	controller	UNDEFINED	
23	GPIO_B<6>	controller	UNDEFINED	
24	GPIO_B<7>	controller	OUTPUT	
25	GPIO_C<0>	controller	UNDEFINED	
26	GPIO_C<1>	controller	UNDEFINED	
27	GPIO_C<2>	controller	UNDEFINED	
28	GPIO_C<3>	controller	UNDEFINED	
29	GPIO_C<4>	controller	UNDEFINED	
30	GPIO_C<5>	controller	UNDEFINED	
31	GPIO_C<6>	controller	UNDEFINED	
32	GPIO_C<7>	controller	UNDEFINED	
33	GPIO_D<0>	controller	UNDEFINED	
34	GPIO_D<1>	controller	UNDEFINED	
35	GPIO_D<2>	controller	UNDEFINED	
36	GPIO_D<3>	controller	UNDEFINED	
37	GPIO_D<4>	controller	UNDEFINED	
38	GPIO_D<5>	controller	UNDEFINED	
39	GPIO_D<6>	controller	UNDEFINED	

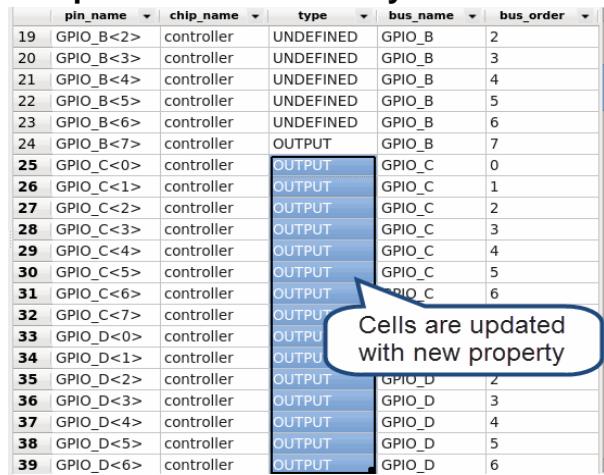
Figure 4-4. Shift-select and Paste Into Multiple Cells in the System Netlist Generator



A screenshot of the same spreadsheet interface. Multiple rows (from 27 to 31) are selected. A context menu is open over row 27, with the 'Paste' option highlighted. Other options in the menu are Copy, Cut, and Delete. Callouts indicate 'Shift-Select multiple cells' pointing to the selected range and 'Paste the property into multiple cells' pointing to the menu. The status bar at the bottom shows '15 rows selected'.

pin_name	chip_name	type	bus_name	bus_order
19	GPIO_B<2>	controller	UNDEFINED	GPIO_B 2
20	GPIO_B<3>	controller	UNDEFINED	GPIO_B 3
21	GPIO_B<4>	controller	UNDEFINED	GPIO_B 4
22	GPIO_B<5>	controller	UNDEFINED	GPIO_B 5
23	GPIO_B<6>	controller	UNDEFINED	GPIO_B 6
24	GPIO_B<7>	controller	OUTPUT	GPIO_B 7
25	GPIO_C<0>	controller	UNDEFINED	GPIO_C 0
26	GPIO_C<1>	controller	UNDEFINED	GPIO_C 1
27	GPIO_C<2>	controller	UNDEFINED	GPIO_C 2
28	GPIO_C<3>	controller	UNDEFINED	GPIO_C 3
29	GPIO_C<4>	controller	UNDEFINED	GPIO_C 4
30	GPIO_C<5>	controller	UNDEFINED	GPIO_C 5
31	GPIO_C<6>	controller	UNDEFINED	GPIO_C 6
32	GPIO_C<7>	controller	UNDEFINED	GPIO_C 7
33	GPIO_D<0>	controller	UNDEFINED	GPIO_D 0
34	GPIO_D<1>	controller	UNDEFINED	GPIO_D 1
35	GPIO_D<2>	controller	UNDEFINED	GPIO_D 2
36	GPIO_D<3>	controller	UNDEFINED	GPIO_D 3
37	GPIO_D<4>	controller	UNDEFINED	GPIO_D 4
38	GPIO_D<5>	controller	UNDEFINED	GPIO_D 5
39	GPIO_D<6>	controller	UNDEFINED	GPIO_D 6

Figure 4-5. Updated Cells In the System Netlist Generator



A screenshot of the spreadsheet after the paste operation. The cells from row 25 to 31 now have the 'OUTPUT' property assigned. A callout bubble says 'Cells are updated with new property'.

pin_name	chip_name	type	bus_name	bus_order
19	GPIO_B<2>	controller	UNDEFINED	GPIO_B 2
20	GPIO_B<3>	controller	UNDEFINED	GPIO_B 3
21	GPIO_B<4>	controller	UNDEFINED	GPIO_B 4
22	GPIO_B<5>	controller	UNDEFINED	GPIO_B 5
23	GPIO_B<6>	controller	UNDEFINED	GPIO_B 6
24	GPIO_B<7>	controller	OUTPUT	GPIO_B 7
25	GPIO_C<0>	controller	OUTPUT	GPIO_C 0
26	GPIO_C<1>	controller	OUTPUT	GPIO_C 1
27	GPIO_C<2>	controller	OUTPUT	GPIO_C 2
28	GPIO_C<3>	controller	OUTPUT	GPIO_C 3
29	GPIO_C<4>	controller	OUTPUT	GPIO_C 4
30	GPIO_C<5>	controller	OUTPUT	GPIO_C 5
31	GPIO_C<6>	controller	OUTPUT	GPIO_C 6
32	GPIO_C<7>	controller	OUTPUT	GPIO_C 7
33	GPIO_D<0>	controller	OUTPUT	GPIO_D 0
34	GPIO_D<1>	controller	OUTPUT	GPIO_D 1
35	GPIO_D<2>	controller	OUTPUT	GPIO_D 2
36	GPIO_D<3>	controller	OUTPUT	GPIO_D 3
37	GPIO_D<4>	controller	OUTPUT	GPIO_D 4
38	GPIO_D<5>	controller	OUTPUT	GPIO_D 5
39	GPIO_D<6>	controller	OUTPUT	GPIO_D 6

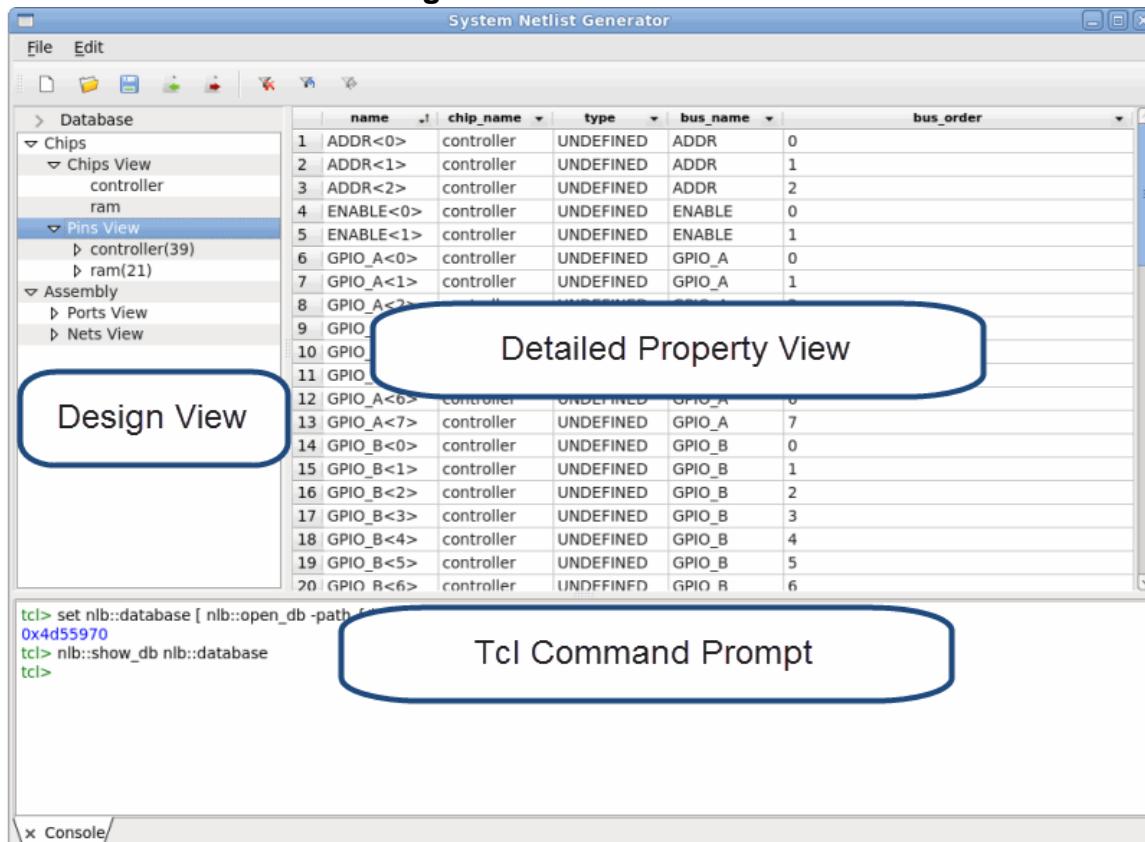
Enable the **Show generated netlist file in Calibre RVE SPICE File Viewer** checkbox to view a graphical representation of the complete netlist.

Main Window

Main Window

System Netlist Generator Main Window.

Figure 4-6. Main Window



Objects

Table 4-1. Main Window Contents

Item	Description
File > New	Creates a new System Netlist Generator project. Command equivalent: sng::new_db
File > Open	Opens an existing System Netlist Generator project. Command equivalent: sng::open_db
File > Save	Saves the current System Netlist Generator project. Command equivalent: sng::save_db
File > Close	Closes the active System Netlist Generator project. Command equivalent: none

Table 4-1. Main Window Contents (cont.)

Item	Description
File > Import Chips	Imports one or more chips from existing files. Command equivalent: sng::import_chip
File > Export Netlist	Exports the selected assembly or chips into a netlist file. Command equivalent: sng::export_netlist
File > Exit	Closes the System Netlist Generator GUI. Command equivalent: none
Edit > Create Chip	Invokes the Create Chip panel. Used to create a new chip name and cell in the project. Command equivalent: sng::create_chip
Edit > Remove Chips	Invokes the Remove Chip panel. Used to remove selected chips from the project. Command equivalent: sng::remove_chip
Edit > Add Pins	Invokes the Add Pins panel. Used to add pins to existing chip definitions without editing the source netlist for that chip. Command equivalent: sng::add_pin
Edit > Add Placement	Invokes the Add Placement panel. Used to create a new placement for a specified chip in the project. Command equivalent: sng::create_placement
Edit > Remove Placements	Invokes the Remove Placement panel. Used to remove selected placements from the project. Command equivalent: sng::remove_placement
Edit > Add Nets	Invokes the Add Nets panel. Used to connect ports in the assembly. Command equivalent: sng::connect
Edit > Set Net Name Template	Invokes the Set Net Name Template panel. Used to control how net names are generated when port connections are made. Command equivalent: sng::set_net_name_template
Edit > Find and Replace	Invokes the Find and Replace utility to find and change net names in the project.

Usage Notes

Enter the following command to invoke the System Netlist Generator main window:

```
sng -gui
```

The main window enables you to perform the following tasks:

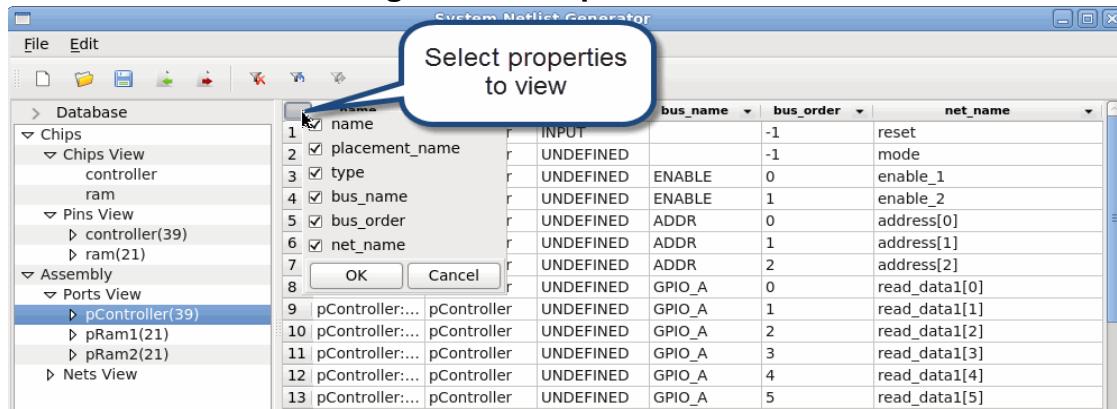
- Create, open, save, close, and edit netlist projects.
- Import and export netlists.
- Enter [System Netlist Generator Commands](#) or Tcl code in the command prompt area.

Properties Panel

Properties button

Shows detailed properties about objects in the design tree.

Figure 4-7. Properties Panel



Objects

Table 4-2. Properties Panel Contents

Item	Description
name	Name of the port or net.
chip_name	Chip associated with the port or net.
placement_name	Placement associated with the port or net.
type	Direction of the port or net.
bus_name	Bus name attached to the port or net.
bus order	Order of the bus pins for the port or net.
net_name	Net connectivity for the port or net.

Usage Notes

The properties panel opens when objects are selected in the Chip Definition tree in the main window. The panel allows you to view and sort all properties of the selected chips, assemblies, or placements.

Tip

i Click the button in the upper-left of the properties table to select items to show in the table. Right-click on any column header to apply a filter or to sort the column contents.

Related Topics

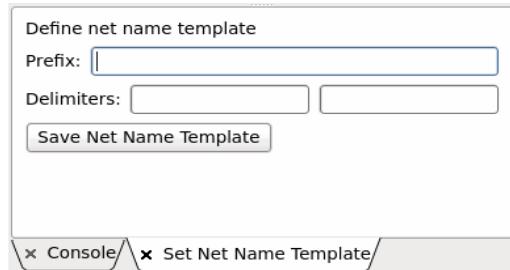
[Main Window](#)

Set Net Name Template

Edit > Set Net Name Template

Controls how net names are generated when port connections are assigned.

Figure 4-8. Set Net Name Template



Objects

Table 4-3. Set Net Name Template Contents

Item	Description
Prefix	String that is prefixed to all net names defined in a sng::connect command that do not have an assigned net name.
Delimiters	Start end delimiters for the net. For example, { and }.
Save Net Name Template	Saves the net name template so that it is applied to port connections when they are created.

Related Topics

[sng::connect](#)

System Netlist Generator Configuration File Format

The System Netlist Generator configuration file allows you to define placements and connectivity for your 3D-IC and then export the definitions to a new netlist file.

You can specify this file on the **Inputs > Netlist** tab of Calibre Interactive to control the creation of a new source netlist file before a Calibre 3DSTACK verification run. See “[Creating a Source Netlist](#)” on page 54 for complete details.

The commands used in the configuration file can also be entered directly into the System Netlist Generator Tcl command prompt or in a System Netlist Generator batch script.

The configuration file supports the following PLACEMENTS, CONNECTIVITY, and EXPORTS commands.

PLACEMENTS	236
CONNECTIVITY	237
EXPORTS.....	239

PLACEMENTS

Imports chips and creates placements.

Usage

```
PLACEMENTS {'  
    placement_name chip_name format file_path top_cell  
    ...  
}'
```

Arguments

- ***placement_name***
Required argument that specifies the name of the placement to create.
- ***chip_name***
Required argument that specifies the name of the chip of which the placement is an instance.
- ***format***
Required argument that specifies the format of the file containing the chip definition. A value of SPICE is currently supported.
- ***file_path***
Required argument that specifies the path to a file containing the chip definitions.
- ***top_cell***
Required argument that specifies the top cell name of the chip.

Description

Imports chip definitions from a specified file and creates a placements for them in the System Netlist Generator. The PLACEMENTS command must only appear once in the configuration file. Each line in the PLACEMENTS command within the required brackets '{' '}' must define the ***placement_name*, *chip_name*, *format*, *file_path*, and *top_cell*** for the placement. There is no limit to the number of placements that you create, but each placement definition must appear on a new line.

Examples

```
PLACEMENTS {  
    myram1 myram SPICE ram_netlist.spi myram  
    myram2 myram SPICE ram_netlist.spi myram  
    interposer1 interposer SPICE interp_rev0.spi interposer  
    controller1 controller SPICE controller.spi controller  
}
```

Related Topics

[CONNECTIVITY](#)

CONNECTIVITY

Connects specified pins with new nets.

Usage

```
CONNECTIVITY {  
    from_placement {'pin ...'} to_placement {'pin ...'} [to_placement {'pin ...'} ...]  
    ...  
}
```

Arguments

- ***from_placement* {'pin ...'}**

Required list of arguments that defines the primary placement name and a list of pins that connect to the secondary placements (to_placement). The from_placement placement must be defined in the **PLACEMENTS** command. Any number of pins can be defined; however, all pins must correspond to the pins in the subsequent to_placement pin lists.

- ***to_placement* {'pin ...'}**

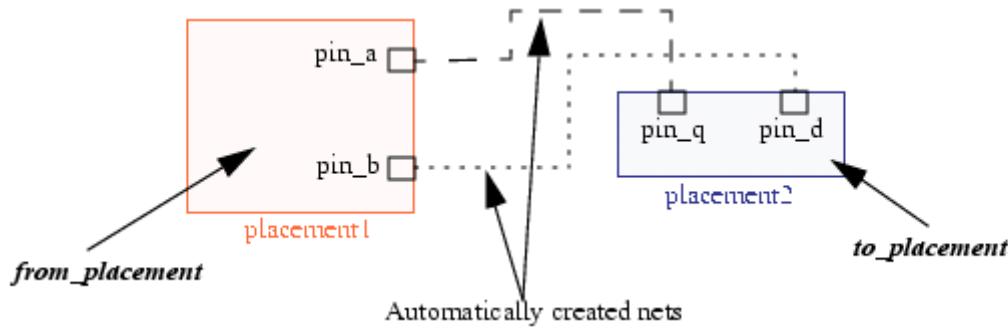
Required list of arguments that defines the secondary placement name and a list of pins that connect to the primary placement (from_placement). The to_placement placement must be defined in the **PLACEMENTS** command. Any number of secondary placements are allowed, but they must be written on the same line.

Description

Defines the connectivity for specified placements in the assembly. Each line in the CONNECTIVITY command defines pin connectivity for a primary placement. A primary placement is specified with the ***from_placement*** argument. Connectivity is established by specifying subsequent placement names and pin lists on the same line. The order of the pin lists establishes the connections. Nets are created automatically to connect the specified pins.

For example, if you want to connect pin_a and pin_b on placement1 to pin_q and pin_d on placement2, as shown in the following figure, your CONNECTIVITY command must be written as follows:

```
CONNECTIVITY {  
    placement1 {pin_a pin_b} placement2 {pin_q pin_d}  
}
```



Examples

This example demonstrates how to define connectivity for the *myram1* and *myram2* placements.

```
CONNECTIVITY {
    myram1{vdd} interposer1{vdd} controller1{vdd}
    myram1{gnd} interposer1{gnd} controller1{gnd}
    myram2{vdd gnd wri} myram1{wri} controller1{wri<1> wri<2> wri<3> wri<4>}
}
```

EXPORTS

Exports the connectivity and placements defined in the configuration file to a netlist file.

Usage

```
EXPORTS {'  
    top_cell format file_name  
    ...  
}'
```

Arguments

- *top_cell*
Required argument that specifies the top cell name for the exported netlist file.
- *format*
Required argument that specifies the netlist format of the exported netlist file. The only supported value is SPICE.
- *file_name*
Required argument that specifies the name of the exported netlist file.

Examples

```
EXPORTS {  
    myTopCell SPICE 3d_assembly.spi  
}
```

Related Topics

[CONNECTIVITY](#)

[PLACEMENTS](#)

System Netlist Generator Commands

A set of batch commands are supported by the System Netlist Generator.

The commands listed in the following table use the syntax conventions outlined in [Table 1-1](#).

Tip

-  All commands support the -help option. Use the -help option in the System Netlist Generator shell to report the usage for each command.
-

Table 4-4. System Netlist Generator Commands

Name	Description
sng::create_journal_file	Creates a journal file in your working directory while the System Netlist Generator tool is running. No journal is created if this command is not specified.
sng::export_db	Exports the current project to a file.
sng::export_netlist	Exports specified chip definitions or the complete assembly to a netlist file.
sng::new_db	Creates a new project.
sng::open_db	Opens the specified project.
sng::save_db	Saves the current state of the project.
sng::filter	Applies a filter to a specified iterator. This command is supported in batch mode only.
sng::get_begin	Moves the specified iterator to the first element of the container.
sng::get_buses	Returns an iterator for all buses defined in the specified chip.
sng::get_chips	Returns an iterator for all chips defined in the current project.
sng::get_nets	Returns an iterator used to access any created nets in the current project.
sng::get_pins	Returns an iterator for all pins on the specified chip.
sng::get_placements	Returns an iterator for all placements defined in the current project.
sng::get_ports	Returns an iterator for all ports defined for the specified placement.
sng::get_property	Returns a list of property values for the specified iterator.
sng::get_property_names	Returns a list of property names.
sng::incr	Increments the specified iterator.
sng::set_property	Changes the value of specified properties.
sng::sort	Sorts the elements of a list and returns a new sorted list to an iterator.

Table 4-4. System Netlist Generator Commands (cont.)

Name	Description
sng::add_pin	Adds a pin to the specified chip.
sng::connect	Creates a new connection.
sng::create_chip	Creates a new chip.
sng::create_placement	Creates a new placement.
sng::import_chip	Imports an existing chip definition.
sng::remove_chip	Removes the selected chip definition.
sng::remove_pin	Deletes a pin from the specified chip.
sng::remove_placement	Removes a placement from the project.
sng::set_net_name_template	Sets the net name template.
sng::unconnect	Deletes specified connections or ports from a net.

sng::create_journal_file

Creates a journal file in your working directory while the System Netlist Generator tool is running. No journal is created if this command is not specified.

Usage

sng::create_journal_file

Arguments

None.

Examples

`sng::create_journal_file`

Related Topics

[System Netlist Generator Commands](#)

sng::export_db

Exports the current project to a file.

Usage

```
sng::export_db db_instance -path file_path [-placements] [-connectivity]
```

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- **-path *file_path***
Required argument and value that specifies the path for the exported project file.
- **-placements**
Optional keyword that specifies to only save placement information to the database. No connectivity is exported unless the -connectivity argument is also specified.
- **-connectivity**
Optional keyword that specifies to only save connectivity information to the database. No placements are exported unless the -placements argument is also specified.

Examples

```
sng::export_db db -path /user/3d_netlist/my_db -placements -connectivity
```

Related Topics

[sng::save_db](#)

sng::export_netlist

Exports specified chip definitions or the complete assembly to a netlist file.

Usage

```
sng::export_netlist db_instance -path file_path
[-chip_name_list {chip_names ... | {chip_name cell_name} ...}]
[-top_cell_name top_cell_name] [-interposer chip_name]
```

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- **-path *file_path***
Required keyword and path for the new netlist file.
- **-chip_name_list {*chip_names* ... | {*chip_name cell_name*} ...}**
Optional keyword and list of chips that you want to write to the netlist. If you do not specify this option, all connectivity information is exported. You can specify a list of chips or a list of chip and cell name pairs. The *cell_name* is the subcircuit name used for the chip in the exported netlist. If you only specify a list of *chip_names*, the exported subcircuit uses the chip name as the name of the subcircuit.
- **-top_cell_name *top_cell_name***
Optional keyword and top cell name of the exported netlist file. The default top cell name is 3DSTACK_TOP_CELL. This option cannot be specified with the -chip_name_list option.
- **-interposer *chip_name***
Optional keyword set that indicates that the specified *chip_name* is an interposer.

Description

Exports an assembly or specified chips to a netlist file. The netlist is written in SPICE format. If the -chip_name_list argument is not used, the tool exports the assembly connectivity information for the entire project.

Examples

```
sng::export_netlist my_db -chip_name_list "myram1 myram2" -path ./3dic.net
```

Related Topics

[sng::save_db](#)

sng::new_db

Creates a new project.

Usage

sng::new_db [-name *db_name*]

Arguments

- -name *db_name*

Optional keyword and argument that specifies the name of the new database.

Examples

```
set db [sng::new_db -name my_3dic_source.sng]
```

Related Topics

[sng::save_db](#)

[sng::open_db](#)

sng::open_db

Opens the specified project.

Usage

sng::open_db -path *file_path* [-name *db_name*]

Arguments

- **-path *file_path***
Required keyword and path to an existing System Netlist Generator project database.
- **-name *db_name***
Optional keyword and argument that specifies the name of the new database.

Examples

Open an existing project:

```
set my_db [sng::open_db -path ./saved_db]
```

Related Topics

[sng::save_db](#)

[sng::remove_chip](#)

sng::save_db

Saves the current state of the project.

Usage

sng::save_db *db_instance* -path *file_path* [-compact]

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- **-path *file_path***
Required keyword and path to the location where you want to save the System Netlist Generator project database.
- **-compact**
Optional keyword that saves the database in compact mode using the **PLACEMENTS** and **CONNECTIVITY** commands. This command option can only be used if the **file_path** is a directory.

Description

Saves the current state of the project to the specified **file_path**. Compact mode saves the project as a System Netlist Generator configuration file. See “[System Netlist Generator Configuration File Format](#)” on page 235 for details.

Note

 Quotation marks (“ ”) cannot be used in the **file_path**.

Examples

`sng::save_db -path /home/user/existing_db`

Related Topics

[sng::open_db](#)

[sng::remove_chip](#)

sng::filter

Applies a filter to a specified iterator. This command is supported in batch mode only.

Usage

sng::filter iterator -expression filtering_expression

Arguments

- **iterator**

Required argument that specifies an iterator returned by one of the following commands:
[sng::get_buses](#), [sng::get_chips](#), [sng::get_nets](#), [sng::get_placements](#), [sng::get_ports](#),
[sng::get_property](#), [sng::get_property_names](#).

- **-expression filtering_expression**

Required keyword and filtering expression that is applied to the iterator object. The **filtering_expression** is any supported Tcl math expression. The expression can contain property names of the objects in the System Netlist Generator project and supports Tcl variables. The expression can contain any of the following symbols:

&& || == != <= < > >=

To distinguish between property names and values, values must be surrounded by literal quotation marks ("").

Examples

Example

This example gets the full list of pins from a chip and then retrieves the properties of one of the buses.

```
#Set the pins iterator to include all pins of the myram chip.  
set pins [sng::get_pins sng::database -chip_name "myram"]  
  
set propName bus_name  
  
#set filtered to equal only those pins that include the bus name adr<>  
set filtered [ sng::filter $pins -expression "$propName == \"adr<>\""]  
  
#get the properties of the filtered pin list  
while {$filtered != ""} {  
    puts [sng::get_properties $filtered -property name]  
    sng::incr filtered 1  
}
```

Example

This example demonstrates how to achieve the same result as the first example using a different method.

```
set pins [sng::get_pins sng::database -chip_name "myram"]
set filtered [ sng::filter $pins \
    -expression {bus_name == \"adr<>\") && (bus_order<=6)}]

while {$filtered != ""} {
    puts [sng::get_properties $filtered -property name]
    sng::incr filtered 1
}
```

Related Topics

[sng::unconnect](#)

[sng::incr](#)

sng::get_begin

Moves the specified iterator to the first element of the container.

Usage

sng::get_begin iterator

Arguments

- **iterator**

Required argument that specifies an iterator returned by one of the following commands:

[sng::get_buses](#), [sng::get_chips](#), [sng::get_nets](#), [sng::get_placements](#), [sng::get_ports](#),
[sng::get_property](#), [sng::get_property_names](#).

Description

Moves the specified iterator to the first element of the container. The iterator can be returned by any of the get_ commands. The iterator value changes each time you specify the **incr** command. The get_begin command allows you to move the iterator to the start of the container.

Examples

In this example, the net iterator changes when you add connectivity to the project. In these cases, it is recommended to use the **get_begin** command as shown.

The following excerpt from the complete example shows the correct usage of the \$ when referencing an iterator:

```
set inets [sng::get_nets db]
<code that affects the inets iterator>
#set inets iterator to first element
set inets [sng::get_begin $inets]
#get properties of nets
while { $inets != "" } {
    set sformat "%-25s"
    set l [list " ${prefix}:"]
    foreach i [lsort [sng::get_properties $inets]] {
        lappend l "$i "
        append sformat "%-24s"
    }
    sng::incr inets 1
    puts [eval format $sformat $l]
}
```

The complete example follows:

```

set db [sng::new_db]
#Start the iterator here to make sure it works properly with the nets
#create later
set inets [sng::get_nets db]
#create chips
foreach chipname {a b c} {
    sng::create_chip db -chip_name $chipname -cell_name top
#define pins for the chips
    foreach pinnmbr { 0 1 2 3 4 5 6 7 8 9 } {
        sng::add_pin db -pin_name pin_$pinnmbr -chip_name $chipname -type INOUT
    }
    #place chips in the design
    foreach placenmbr { 0 1 2 3 } {
        sng::create_placement db -placement_name p_${chipname}_${placenmbr} \
        -chip_name ${chipname}
    }
}
#create connections for each placement
foreach placenmbr {0 1 2 3} {
    foreach netnmbr {0 1 2 3 4 5 6 7 8 9} {
        sng::connect db -net_name net_${placenmbr}_${netnmbr} \
        -ports [list \
        p_a_${placenmbr}::pin_${netnmbr} \
        p_b_${placenmbr}::pin_${netnmbr} \
        p_c_${placenmbr}::pin_${netnmbr} \
        ]
    }
}
#unconnect certain nets
foreach netname {net_0_2 net_1_2 net_2_2 net_3_2} {
    sng::unconnect db -net_name $netname
}
set inets [sng::get_begin $inets]
#get properties of nets
while { $inets != "" } {
    set sformat "%-25s"
    set l [list " ${prefix}:"]
    foreach i [lsort [sng::get_properties $inets]] {
        lappend l "$i "
        append sformat "%-24s"
    }
    sng::incr inets 1
    puts [eval format $sformat $l]
}
}

```

Related Topics

[sng::incr](#)

sng::get_buses

Returns an iterator for all buses defined in the specified chip.

Usage

sng::get_buses *db_instance* -chip_name *name*

Arguments

- ***db_instance***

Required argument that specifies the database instance.

- **-chip_name *name***

Required keyword and name of a chip in the project.

Examples

This example sets an iterator to the buses on the mem_die chip and retrieves all of the bus names.

```
set busIterator [sng::get_buses my_db -chip_name mem_die]
while {$busIterator ne ""} {
    set bus_name [sng::get_properties $busIterator -property "bus_name"]
    sng::incr busIterator
}
```

Related Topics

[sng::incr](#)

[sng::get_chips](#)

[sng::get_nets](#)

[sng::get_placements](#)

[sng::get_ports](#)

[sng::get_property](#)

[sng::get_property_names](#)

[sng::get_pins](#)

sng::get_chips

Returns an iterator for all chips defined in the current project.

Usage

sng::get_chips *db_instance*

Arguments

- ***db_instance***

Required argument that specifies the database instance.

Examples

This example retrieves the names of all chips in the database.

```
set chipIterator [sng::get_chips my_db]
while {$chipIterator ne ""} {
    set chip_name [sng::get_properties $chipIterator -property "chip_name"]
    puts "Chip name is $name"
    sng::incr chipIterator
}
```

Related Topics

[sng::incr](#)
[sng::get_pins](#)
[sng::get_nets](#)
[sng::get_placements](#)
[sng::get_ports](#)
[sng::get_property](#)
[sng::get_property_names](#)
[sng::get_buses](#)

sng::get_nets

Returns an iterator used to access any created nets in the current project.

Usage

sng::get_nets *db_instance*

Arguments

- ***db_instance***

Required argument that specifies the database instance.

Examples

This example retrieves the names of all nets in the database.

```
set netIterator [sng::get_nets my_db ]  
while {$netIterator ne ""} {  
    set net_name [sng::get_properties $netIterator -property "net_name"]  
    sng::incr netIterator  
}
```

Related Topics

[sng::incr](#)

[sng::get_chips](#)

[sng::get_pins](#)

[sng::get_placements](#)

[sng::get_ports](#)

[sng::get_property](#)

[sng::get_property_names](#)

[sng::get_buses](#)

sng::get_pins

Returns an iterator for all pins on the specified chip.

Usage

```
sng::get_pins db_instance -chip_name name [-bus_name bus_name]
```

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- **-chip_name *name***
Required keyword and name of a chip in the project.
- **-bus_name *bus_name***
Optional keyword and name of a bus in the project. If this option is used, the iterator refers to all pins that share the *bus_name*.

Examples

This example sets the iterator to return the pins on the mem_die chip. The while loop retrieves and prints all of the pin names for that chip.

```
set pinIterator [sng::get_pins my_db -chip_name mem_die]
while {$pinIterator ne ""} {
    set pin_name [sng::get_properties $pinIterator -property "pin_name"]
    puts "Pin name is $name."
    sng::incr pinIterator
}
```

Related Topics

[sng::get_buses](#)
[sng::get_chips](#)
[sng::get_nets](#)
[sng::get_placements](#)
[sng::get_ports](#)
[sng::get_property](#)
[sng::get_property_names](#)

sng::get_placements

Returns an iterator for all placements defined in the current project.

Usage

sng::get_placements *db_instance* [-chip_name *name*]

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- **-chip_name *name***
Optional keyword and name of a chip in the project. This returns all placements of the specified chip. If this argument set is not specified, the iterator returns all placements in the current project.

Examples

This example demonstrates how to get all placements for the mem_die chip.

```
set placementIterator [sng::get_placements my_db -chip_name mem_die]
puts "Placements for mem_die chip are:"
while {$placementIterator ne ""} {
    set placement_name [sng::get_properties $placementIterator \
        -property "placement_name"]
    puts "\t$placement_name"
    sng::incr placementIterator
}
```

Related Topics

[sng::get_pins](#)
[sng::get_chips](#)
[sng::get_nets](#)
[sng::get_property_names](#)
[sng::get_ports](#)
[sng::get_property](#)
[sng::get_buses](#)
[sng::create_placement](#)

sng::get_ports

Returns an iterator for all ports defined for the specified placement.

Usage

```
sng::get_ports db_instance -placement_name placement_name [-bus_name bus_name]  
[-net_name net_name] [-unconnected]
```

Arguments

- **db_instance**
Required argument that specifies the database instance.
- **-placement_name placement_name**
Required keyword and name of a placement in the project.
- **-bus_name bus_name**
Optional keyword and name of a bus in the project. If this option is used, the iterator refers to all ports that share the *bus_name*.
- **-net_name net_name**
Optional keyword and name of a net in the project. If this option is used, the iterator refers to all ports that share the *net_name*.
- **-unconnected**
Optional keyword that returns all floating ports of the specified placement.

Description

Initializes an iterator that can be used to access the ports of specified placements. Use the **-unconnected** option to find all unconnected pins of the placement. The **-net_name** and **-bus_name** options can be used to narrow down the list of ports to a specified string.

Examples

Example 1

This example demonstrates how to get the port names of the mem_die_placement.

```
set portIterator [sng::get_ports my_db \  
-placement_name memory_die::mem_die_placement]

while {$portIterator ne ""} {  
    set port_name [sng::get_properties $portIterator -property "port_name"]  
    puts "\t$port_name"  
    sng::incr portIterator  
}
```

Example 2

This example demonstrates how to retrieve all unconnected ports of the memory_die::mem_die_placement placement that use the bus name “wda”.

```
set portIterator [sng::get_ports my_db
    -placement_name memory_die::mem_die_placement -unconnected \
    -bus_name "wda"]

while {$portIterator ne ""} {
    set port_name [sng::get_properties $portIterator -property "port_name"]
    puts "\t$port_name"
    sng::incr portIterator
}
```

Related Topics

[sng::incr](#)
[sng::get_chips](#)
[sng::get_nets](#)
[sng::get_placements](#)
[sng::get_property_names](#)
[sng::get_property](#)
[sng::get_pins](#)
[sng::get_buses](#)

sng::get_property

Returns a list of property values for the specified iterator.

Usage

```
sng::get_property iterator [-property property_name]
```

Arguments

- *iterator*

Required argument that specifies an iterator returned by one of the following commands:
[sng::get_buses](#), [sng::get_chips](#), [sng::get_nets](#), [sng::get_placements](#), [sng::get_ports](#),
[sng::sort](#).

- -property *property_name*

Optional keyword and name of a property. This can be any value returned by the
[sng::get_property_names](#) command. If this is not specified, the command returns all
property name and value pairs associated with the specified **iterator**.

Examples

```
sng::get_property $portIterator -property "port_name"
```

Related Topics

[sng::incr](#)

[sng::get_chips](#)

[sng::get_nets](#)

[sng::get_placements](#)

[sng::get_ports](#)

[sng::get_buses](#)

[sng::get_property_names](#)

[sng::get_pins](#)

sng::get_property_names

Returns a list of property names.

Usage

sng::get_property_names *iterator*

Arguments

- ***iterator***

Required argument that specifies an iterator returned by one of the following commands:

[sng::get_chips](#), [sng::get_pins](#), [sng::get_nets](#), [sng::get_placements](#), [sng::get_ports](#),
[sng::get_buses](#), [sng::sort](#).

Description

Returns a list of property names for the specified **iterator**. The properties returned for the iterator depend on the type of iterator. The following properties are returned for each issued command and iterator type:

Table 4-5. Property Values For Each Iterator Type

Command	Iterator Type	Returned Property Values
sng::get_chips	Chip iterator	“chip_name” “cell_name”
sng::get_pins	Pin Iterator	“pin_name” “chip_name” “type” “bus_name” “bus_order”
sng::get_buses	Bus Iterator	“bus_name” “order_range”
sng::get_placements	Placement Iterator	“placement_name” “chip_name”
sng::get_ports	Port Iterator	“port_name” “placement_name” “type” “bus_name” “bus_order”
sng::get_nets	Net Iterator	“net_name”

Examples

Get a list of ports from the mem_die_placement placement.

```
set portIterator [sng::get_ports my_db -placement_name mem_die_placement]
while {$portIterator ne ""} {
    set port_property_name_list [sng::get_property_names $portIterator]
    puts "\t$port_property_name_list"
    sng::incr portIterator
}
```

Related Topics

[sng::incr](#)

[sng::get_chips](#)

[sng::get_nets](#)

[sng::get_placements](#)

sng::incr

Increments the specified iterator.

Usage

sng::incr *iterator* [*step*]

Arguments

- ***iterator***

Required argument that specifies an iterator returned by one of the following commands:

[sng::get_buses](#), [sng::get_chips](#), [sng::get_nets](#), [sng::get_placements](#), [sng::get_ports](#),
[sng::get_property](#), [sng::get_property_names](#).

- ***step***

Optional argument that specifies an offset for the iterator. When the **incr** command is executed, the ***iterator*** is incremented by this *step* value. Negative values decrement the iterator. The default is 1.

Description

Changes an iterator by a specified step value. When the iterator reaches the end, it is reset to an empty string("")).

The \$ symbol is not used when referencing the iterator in the sng::incr command, however, the \$ symbol is used when passing a specific iterator element to other commands, as shown in the example.

Examples

Set the iterator to the ports on the mem_die placement and print them by retrieving the property names with the [sng::get_property](#) command.

```
set portIterator [sng::get_ports my_db -placement_name mem_die_placement]
while {$portIterator ne ""} {
    set port_name [sng::get_properties $portIterator -property "PortName"]
    puts "\t$port_name"
    sng::incr portIterator
}
```

sng::set_property

Changes the value of specified properties.

Usage

sng::set_property *iterator* -property *property_name* -value *property_value*

Arguments

- ***iterator***

Required argument that specifies an iterator returned by one of the following commands:
[sng::get_nets](#), [sng::get_buses](#), [sng::get_placements](#), and [sng::get_pins](#).

- **-property *property_name***

Required keyword and property name to change.

- **-value *property_value***

Required keyword and new value for the **property_name** value.

Description

Sets the value of a specified **property_name** to a different **property_value**. This command can only be used for pins, nets, and placements.

Examples

This example demonstrates how to change the placement names for the logic_die chip using the set_property command.

```
puts "Changing placement names for logic_die chip"
set placementIterator [ sng::get_placements -chip_name "logic_die"]
while {$placementIterator ne ""} {
    set old_value [ sng::get_properties $placementIterator \
        -property "placement_name"]
    puts "Old value is $old_value"
    sng::set_property $placementIterator -property placement_name \
        -value new_name_of_placement
    set new_value [ sng::get_properties $placementIterator \
        -property "placement_name"]
    puts "New value is $new_value"
    sng::incr placementIterator
}
```

Related Topics

[sng::get_property_names](#)

sng::sort

Sorts the elements of a list and returns a new sorted list to an iterator.

Usage

```
sng::sort iterator -property property_name [-increasing | -decreasing]  
[-type { Dictionary | Integer | Real } ]
```

Arguments

- ***iterator***

Required argument that specifies an iterator returned by one of the following commands:
[sng::get_buses](#), [sng::get_chips](#), [sng::get_nets](#), [sng::get_placements](#), [sng::get_ports](#),
[sng::get_property](#), [sng::get_property_names](#).

- **-property *property_name***

Required keyword and property name to sort.

- **-increasing**

Optional keyword that sorts the list from least to greatest. This command option cannot be specified with -decreasing. This is the default.

- **-decreasing**

Optional keyword that sorts the list from greatest to least. This command option cannot be specified with -increasing.

- **-type { Dictionary | Integer | Real }**

Optional keywords that specify the type of sorting to apply. The default is Dictionary. The sorting keywords function as follows:

- Dictionary

Sorts the ***iterator*** strings in lexicographic order.

- Integer

Converts the ***iterator*** value list to integers and sorts the list in -increasing or -decreasing order.

- Real

Converts the ***iterator*** value list to floating-point values and sorts the list using floating comparison.

Examples

This example demonstrates how to generate nets with the names net0, net1, net2, and so on, and connect them to the ports of two placements. The sort command is used to connect one of the placements with an increasing net order and the other with a decreasing net order.

```
set portIt1 [sng::get_ports -placement "logic_die::placement1"]
set increasedPortIt1 [sng::sort $portIt1 -property "PortName"]
set portIt2 [sng::get_ports -placement "logic_die::placement2"]
set decreasedPortIt2 [sng::sort $portIt2 -property "PortName" -decreasing]
sng::set_net_name_template -template {net*} -type WILDCARD
while {($decreasedPortIt2 ne "") && ($increadedPortIt1 ne "")} {
    set elem1 [sng::get_properties $increadedPortIt1 -property "PortName"]
    set elem2 [sng::get_properties $decreasedPortIt2 -property "PortName"]
    sng::connect $elem1 $elem2
    sng::incr decreasedPortIt2
    sng::incr increadedPortIt1
}
if {($decreasedPortIt ne "") || ($increadedPortIt1 ne "")} {
    error "Connectivity Error: Port counts for placements are not equal!"
}
```

Related Topics

[sng::get_property](#)

sng::add_pin

Adds a pin to the specified chip.

Usage

```
sng::add_pin db_instance -pin_name pin_name -chip_name chip_name
    -type { INPUT | OUTPUT | INOUT } [-bus_name bus_name [-bus_order bus_order]]
```

Arguments

- ***db_instance***
Required argument that specifies the database instance to modify.
- **-pin_name *pin_name***
Required keyword and name of a new pin to create.
- **-chip_name *chip_name***
Required keyword and name of an existing chip to which the new pin is added.
- **-type { INPUT | OUTPUT | INOUT }**
Required keyword set that specifies the direction of the new pin.
- **-bus_name *bus_name***
Optional keyword and bus name for the pin.
- **-bus_order *bus_order***
Optional keyword and order of the bus for the pin. This argument must only be specified with the -bus_name argument.

Examples

Add a ground pin to a new chip and connect a bus of pins to a memory chip.

```
sng::create_chip -name memory -cell_name mem_die
sng::add_pin -pin_name VSS -chip_name memory -type INOUT
for {set index 0} {$index < 10} {inc index} {
    sng::add_pin -pin_name test_[${index}] -chip_name memory \
        -type OUTPUT -bus_name test -bus_order ${index}
}
```

Related Topics

[sng::remove_pin](#)

sng::connect

Creates a new connection.

Usage

```
sng::connect db_instance [-net_name net_name]
  { -ports port_list {-add_ports_to_net | -create_new }
    [-delimiters '{' delimiter_pair '}']
    [-range range] [-starting_index index] | -buses bus_name_list }
```

Arguments

- **db_instance**

Required argument that specifies the database instance.

- **-net_name net_name**

Optional keyword and name of the net to create. If this command is not specified, the net name for the generated net is taken from the net name template. Net name templates are created using the [sng::set_net_name_template](#) command.

- **-ports port_list**

Required keyword and list of ports that you want to connect to the specified net.

- **-add_ports_to_net**

Keyword that adds all specified ports to an existing net. If the net does not exist, a new net is created. This keyword must only be used with the **-ports** argument.

- **-create_new**

Required keyword that connects all specified ports to a new net. This keyword must only be used with the **-ports** argument.

- **-delimiters '{' delimiter_pair '}'**

Optional keyword and pair of delimiters that indicate the start and end character of a bus. This keyword must only be used with the **-ports** argument. The default delimiter_pair value is {[\]}.

- **-range range**

Optional keyword and index *range* to which the bus is connected. This keyword must only be used with the **-ports** argument. All ports are connected by default.

- **-starting_index index**

Optional keyword and integer index from which the bus starts counting. This keyword must only be used with the **-ports** argument.

- **-buses bus_name_list**

Keyword and list of buses that you want to connect to a specified net.

Description

Specifies net connectivity for ports in the design. The -net_name option can be used if you have net name templates configured with the [sng::set_net_name_template](#) command.

Examples

Example 1

This example connects the ports of two placements. If the number of ports do no match, an error is issued.

```
set portIt1 [sng::get_ports my_db -placement_name "logic_die::placement1"]
set portIt2 [sng::get_ports my_db -placement_name "logic_die::placement2"]
set portList1 [list]
while {$portIt1 ne ""} {
    lappend portList1 [sng::get_properties $portIt1 -property "port_name"]
    sng::incr portIt1 1
}
set portList2 [list]
while {$portIt2 ne ""} {
    lappend portList2 [sng::get_properties $portIt2 -property "port_name"]
    sng::incr portIt2 1
}
If {[llength $portList2] != [llength $portList1]} {
    error "Ports of placements logic_die::placement1 and \
    logic_die::placement2 are not same"
} else {
    set_net_name_template my_db {net_} ; # net_[0], net_[1] ...
    foreach port1 $portList1 port2 $portList2 {
        sng::connect my_db -ports {$port1 $port2}
    }
}
```

Example 2

This example generates a net named DUMMY_CUPO[0], which connects the RELI3_placement::DUMMY_CUPI[0] pin to the RELI2_placement::DUMMY_CUPO[0] pin, the DUMMY_CUPO[1] pin to the RELI3_placement::DUMMY_CUPI[1] pin, and so on.

```
sng::connect my_db -net_name DUMMY_CUPO \
    -buses "RELI3_placement::DUMMY_CUPI RELI2_placement::DUMMY_CUPO"
```

Related Topics

[sng::unconnect](#)

[sng::incr](#)

[sng::set_net_name_template](#)

sng::create_chip

Creates a new chip.

Usage

```
sng::create_chip db_instance -chip_name name [-cell_name cell_name]
```

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- **-chip_name *name***
Required keyword and name of the chip to create. The name must be unique.
- **-cell_name *cell_name***
Optional keyword and cell name for the chip. If this argument is not specified, the name of the cell is given by the name -chip_name argument.

Examples

```
sng::create_chip -chip_name memory -cell_name mem_die
```

Related Topics

[sng::import_chip](#)

sng::create_placement

Creates a new placement.

Usage

```
sng::create_placement db_instance -chip_name chip_name  
-placement_name placement_name
```

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- **-chip_name *chip_name***
Required keyword and name of an existing chip in the project.
- **-placement_name *placement_name***
Required keyword and name of the placement to create. The name must be unique.

Examples

```
sng::create_placement my_db -placement_name "lg_placement1" \  
-chip_name "logic_die"
```

Related Topics

[sng::create_chip](#)

sng::import_chip

Imports an existing chip definition.

Note

-  Set the MGC_SNG_SPICE_IMPORT_TEMP_DIR environment variable to the path of a temporary directory if you are importing large netlists.
-

Usage

```
sng::import_chip db_instance -chip_name name -path file_path
    -type SPICE -cell_name cell_name
    { [-bus_detection {YES | NO}] [-bus_pattern reg_expression]
      [-bus_order_pattern reg_expression] }
```

Arguments

- ***db_instance***

Required argument that specifies the database instance.

- **-chip_name *name***

Required keyword and name of the chip in the **file_path**. The name of the chip must be identical to the name of the chip in the specified netlist file.

- **-path *file_path***

Required keyword and path to a netlist file that contains the chip you want to import.

- **-type SPICE**

Required keywords that specify the format of the netlist file. The System Netlist Generator currently supports SPICE netlists.

- **-cell_name *cell_name***

Required keyword and name of the cell in the input netlist file.

- **-bus_detection {YES | NO}**

Optional keywords that specify whether smart bus detection is enabled. Specify YES to enable smart bus detection. The default is YES.

- **-bus_pattern *reg_expression***

Optional keyword and expression used to generate the name of the buses. This command option must only be used with the -bus_detection YES option. Regular expression statements inside parentheses “()” are used in the bus name generation. If the corresponding pin is not matched with the corresponding regular expression, the pin is not associated with a bus. The default value for *reg_expression* is `\{(\D+)(\d*)(\D+)\d+(\D*)\}`.

For example, if *reg_expression* is set to the following:

```
-bus_pattern \{ (\D+)_.* \}
```

And the following pins are detected in the imported chip:

```
test_[1] test_[2] test_[3] pins
```

The bus for those pins is named “test”.

- **-bus_order_pattern *reg_expression***

Optional keyword and regular expression that defines a bus order pattern for the imported chip netlist. The bus order is a regular pattern that identifies each pin of a bus (for example, bus_1, bus_2, or bus[1], bus[2]). This command option must only be used with the -bus_detection YES option. The default value for *reg_expression* is {(\d+)\D*\\$}.

If a pin has a bus, but the bus order is not recognized, a warning is issued.

Examples

This example demonstrates different bus detection options when importing a new chip into the project database.

```
#use a bus_order_pattern
sng::import_chip my_db -chip_name memory -type SPICE -path ./mem.spi \
-cell_name mem_die -bus_detection YES \
-bus_pattern {(\D+)\d+(.*)} -bus_order_pattern {\D+(\d+)_.*}
#use a bus_pattern
sng::import_chip my_db -chip_name memory -type SPICE -path ./mem.spi \
-cell_name mem_die -bus_detection YES -bus_pattern {(\D+)\d+(. *)}
#do not use smart detection
sng::import_chip my_db -chip_name memory -type SPICE -path ./mem.spi \
-cell_name mem_die -bus_detection NO
```

Related Topics

[sng::create_chip](#)

sng::remove_chip

Removes the selected chip definition.

Usage

sng::remove_chip *db_instance* -chip_name *chip_name*

Arguments

- ***db_instance***

Required argument that specifies the database instance.

- **-chip_name *name***

Required keyword and name of a chip in the project that you want to remove.

Examples

```
sng::remove_chip db -chip_name mem_die
```

Related Topics

[sng::remove_pin](#)

[sng::remove_placement](#)

sng::remove_pin

Deletes a pin from the specified chip.

Usage

sng::remove_pin db_instance -pin_name pin_name -chip_name chip_name

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- ***-pin_name pin_name***
Required keyword and name of the pin that you want to delete.
- ***-chip_name name***
Required keyword and name of an existing chip in the project.

Examples

This example creates a new chip, adds a 10-bit bus to the chip with the prefix “test_”, and then removes one of the pins from the bus.

```
set my_db [new_db]
sng::create_chip my_db -chip_name memory -cell_name mem_die

for {set index 0} {$index < 10} {inc index} {
    sng::add_pin db -pin_name test_[$index] -chip_name memory -type OUTPUT \
        -bus_name test -bus_order $index
}
sng::remove_pin db -pin_name test_0 -chip_name memory
```

Related Topics

[sng::remove_chip](#)

[sng::remove_placement](#)

sng::remove_placement

Removes a placement from the project.

Usage

sng::remove_placement *db_instance* -placement_name *placement_name*

Arguments

- ***db_instance***

Required argument that specifies the database instance.

- **-placement_name *placement_name***

Required keyword and name of a placement in the project that you want to delete.

Examples

```
sng::remove_placement my_db -placement_name "lg_placement1"
```

Related Topics

[sng::remove_chip](#)

[sng::remove_pin](#)

[sng::unconnect](#)

[sng::get_placements](#)

sng::set_net_name_template

Sets the net name template.

Usage

```
sng::set_net_name_template db_instance prefix [-delimiters '{' delimiter_pair '}' ]
```

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- ***prefix***
Required string that is prefixed to all net names defined in a [sng::connect](#) command that do not have an assigned net name.
- -delimiters '{' *delimiter_pair* '}'
Optional keyword and pair of delimiters that indicate the start and end character of a bus.
This keyword must only be used with the **-ports** argument. The default delimiter_pair value is {[\]}.

Description

Applies a net name template to all nets. The ***prefix*** argument is a string that is prefixed to all net names that do not have a net name property. For example, if ***prefix*** is specified as {Net_}, all nets generated with the [sng::connect](#) command without the -net_name option are named as follows:

Net_0, Net_1, Net_3 ...

Examples

```
sng::set_net_name_template my_db {Net_} -delimiters {\<\>}
```

Related Topics

[sng::connect](#)

sng::unconnect

Deletes specified connections or ports from a net.

Usage

```
sng::unconnect db_instance {-net_name net_name | -ports port_list}
```

Arguments

- ***db_instance***
Required argument that specifies the database instance.
- **-net_name *net_name***
Required keyword and name of the net to disconnect from all ports.
- **-ports *port_list***
Required keyword and list of ports that you want to disconnect from any connected nets.
Ports that do not have any defined connectivity are flagged as warnings.

Examples

Remove Net1 connectivity.

```
sng::unconnect my_db -net_name "Net1"
```

Remove Port1 and Port2 from the Net2 net.

```
sng::unconnect my_db -net_name "Net2" -ports {Port1 Port2}
```

Related Topics

[sng::connect](#)

AIF Converter Reference

Calibre 3DSTACK includes three utilities that allow you to convert AIF designs to OASIS and GDS layouts, and SPICE netlists.

AIF Support	278
AIF Export File Format	280
3dstack::aif2gds	282
3dstack::aif2oasis	283
3dstack::aif2spice	284

AIF Support

The AIF converters are invoked on the Calibre command-line. Not all AIF sections are used by Calibre 3DSTACK.

AIF Format

The AIF format is documented at the following location:

http://www.artwork.com/package/aif/aif_netlist.htm

Supported AIF Sections

The following sections are supported by the AIF converters:

- [DATABASE]
- [DIE]
- [MCM_DIE]
- [BGA]
- [PADS]
- [NETLIST]
- [RINGS]

Unsupported AIF Sections

The following sections are not ignored by the AIF converters:

- [BONDABLE_RING_AREA]
- [WIRE]

- [FIDUCIALS]
- [DIE_LOGO]

AIF Export File Format

AIF file generated by the [export_connectivity](#) and [config](#) commands.

Calibre 3DSTACK can export the netlist of the layout assembly in AIF format, among others.

Format

The multi-die AIF file is generated with the following format:

- The AIF version is 2.0.
- All units are in microns.

Parameters

AIF File Section	Description
[DIE]	Specifies the top cell of assembly layout.
[BGA]	This section is left empty.
[MCM_DIE]	Specifies all placements of the assembly layout
[PADS]	Specifies the pads on each placement. SQUARE, RECTANGLE, and POLYGON pad types are supported. The pad types are generated for each cell (not placement) of the assembly layout. The pad types are enumerated for each cell
[NETLIST]	Specifies the pins of all placements in the assembly layout. Each pin can have more than one row, with each row corresponding to a pin instance. The NETNAME column indicates the net name attached to the pin instance. The PAD column of each row consists of the placement name, the pin name, and the ID number of pin instance (if the pin has several pin instances). The TYPE column corresponds to the pad type of the current pin instance. The PAD_X and PAD_Y columns correspond to the center coordinates of the pad.

Examples

The following is an example an of AIF output file.

```
[DATABASE]
TYPE=AIF
VERSION=2.0
UNITS=UM
MCM=TRUE

[DIE]
NAME=TOPCELL_3DI

[MCM_DIE]
BGA=assembly1_tier1_BGA1
C4_BUMP=assembly1_tier3_C4_BUMP1
...
[MCM_BGA_assembly1_tier1_BGA1]
[MCM_C4_BUMP_assembly1_tier3_C4_BUMP1]
...
[PADS]
BGA_pad_0=SQUARE 2
C4_BUMP_pad_0=SQUARE 2
...
[NETLIST]
;NETNAME PAD# TYPE PAD_X PAD_Y BALL# TYPE BALL_X BALL_Y FIN# TYPE FIN_X
FIN_Y ANGLE
685 assembly1_tier1_BGA1.BS_SW1_GPIO_40 BGA_pad_0 3200.001 13600.010 - - -
- - - - -
668 assembly1_tier1_BGA1.BS_SW1_GPIO_41 BGA_pad_0 5600.100 14400.000 - - -
- - - - -
...
```

3dstack::aif2gds

Reads an AIF file and exports a GDS layout file.

Usage

`3dstack::aif2gds -aif aif_file -gds gds_file`

Arguments

- **-aif *aif_file***

Required argument and path to an AIF file. See “[AIF Support](#)” on page 278 for details on supported AIF statements.

- **-gds *gds_file***

Required argument and path to the GDS output file.

Examples

Enter the following commands from the command-line:

```
calibre -ys -3dstack
3dstack::aif2gds -aif design.aif -gds converted_aif.gds
exit
```

3dstack::aif2oasis

Reads an AIF file and exports an OASIS layout file.

Usage

3dstack::aif2oasis -aif *aif_file* -oasis *oas_file*

Arguments

- **-aif *aif_file***

Required argument and path to an AIF file. See “[AIF Support](#)” on page 278 for details on supported AIF statements.

- **-oasis *oas_file***

Required argument and path to the OASIS output file.

Examples

Enter the following commands from the command-line:

```
calibre -ys -3dstack
3dstack::aif2oasis -aif in.aif -oasis converted_aif.oas
exit
```

3dstack::aif2spice

Reads an AIF file and exports a SPICE netlist file.

Usage

3dstack::aif2spice {-aif *aif_file* [-aif *aif_file*] ...} -spice *spice_file*

Arguments

- **-aif *aif_file* [-aif *aif_file*] ...**

Required argument and path to one or more AIF files. See “[AIF Support](#)” on page 278 for details on supported AIF statements.

- **-spice *spice_file***

Required argument and path to the SPICE output file.

Examples

Enter the following commands from the command-line:

```
calibre -ys -3dstack
3dstack::aif2spice -aif in.aif -spice converted_aif.sp
exit
```

Spreadsheet Converters

You can convert CSV-formatted spreadsheets into SPICE or GDS and OASIS for use in Calibre 3DSTACK assembly and verification flows.

For information on creating and managing existing netlists, see the “[System Netlist Generator](#)” on page 218.

The following converters are available:

3dstack::ss2gds	286
3dstack::ss2oasis	288
3dstack::ss2spice	290
3dstack::xsi2gds	292
3dstack::xsi2oasis	295
3dstack::xsi2spice	298

3dstack::ss2gds

Creates GDS layouts that include pin and die boundary locations from a spreadsheet (CSV) file.

Usage

```
3dstack::ss2gds -ss ss_file
  [-order column_list]
  [-die_info GDS '{'
    -layout_path layout_path
    [-die_name die_name]
    [-precision precision]
    [-vertices {x1 y1 ... xn yn}]
  '}']...
  [-ignore [reg_exp_list]]
  [-package package_name]
  [-package_mapping '{'
    -die_name die_name
    -layer_number layer_number
    [-vertices {x1 y1 ... xn yn}]
  '}']...
  [-text_only labels_file
    [-labels {NET_NAME | PIN_NUMBER | PIN_NAME}]
    [-attach_to {component layer ...}]]]
```

Arguments

- **-ss ss_file**
Required path to the spreadsheet (CSV) input file.
- **-order column_list**
Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF_DES, PIN_NUMBER, PIN_X, PIN_Y, PIN_NAME, and NET_NAME.
- **-die_info GDS '{'die_arguments '}'**
Optional set of statements that specify information about the output layout. The following *die_arguments* are available:
 - layout_path *layout_path* — Path to the output GDS layout. You can specify a directory or a full file path.
 - die_name *die_name* — Name of the output GDS layout file.
 - precision *precision* — Precision of the output GDS layout.
 - vertices {*x1 y1 ... xn yn*} — Polygon coordinates for generated markers in the GDS output. These coordinates are not multiplied by the *precision* during generation.

- **-ignore [reg_exp_list]**
Optional argument set that specifies a list of regular expressions to match pins to ignore. If you specify -ignore without the *reg_exp_list* value, the default expression is {^NC.*\$}. The default expression searches for all pins that begin with the string “NC”.
- **-package package_name**
Optional name for the generated package layout.
- **-package_mapping ‘{’ package_arguments‘}’**
Optional set of statements that specify mapping information about a package design from the REFDES column of the spreadsheet file. The following *package_arguments* are available:
 - die_name *die_name* — Name of the die for which the mapping is given.
 - layer_number *layer_number* — Layer number of the mapped die. The *layer_number* must be greater than 2.
 - vertices {*x1 y1 ... xn yn*} — Polygon coordinates for the pin cells. These coordinates are not multiplied by the *precision* during generation.

- **-text_only labels_file**
Optional argument set that generates a text label file at the specified *labels_file* path. When this argument set is applied, the only output from the 3dstack::ss2gds command is the text labels file. The text labels file uses the labels and locations defined in the *xsi_file*.

The following options are exclusive to the -text_only argument set:

-labels {NET_NAME | PIN_NUMBER | PIN_NAME}

Optionally specifies the column in the *xsi_file* from which to retrieve text labels. The default is the NET_NAME column.

-attach_to layer -component component_name

Specifies the layer of the layout and the component name to which the labels are attached.

Examples

```
calibre -ys -3dstack
```

```
3dstack::ss2gds -ss spreadsheet.txt -die_info GDS ./output \
    -package pkg -package_info BGA 3 3 3 -3 -3
```

3dstack::ss2oasis

Creates OASIS layouts that include pin and die boundary locations from a spreadsheet (CSV) file.

Usage

```
3dstack::ss2oasis -ss ss_file
  [-order column_list]
  [-die_info OASIS '{'
    -layout_path layout_path
    [-die_name die_name]
    [-precision precision]
    [-vertices {x1 y1 ... xn yn}]
  '}']...
  [-ignore [reg_exp_list]]
  [-package package_name]
  [-package_mapping '{'
    -die_name die_name
    -layer_number layer_number
    [-vertices {x1 y1 ... xn yn}]
  '}']...
```

Arguments

- **-ss ss_file**

Required path to the spreadsheet (CSV) input file.

- **-order column_list**

Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF_DES, PIN_NUMBER, PIN_X, PIN_Y, PIN_NAME, and NET_NAME.

- **-die_info OASIS '{'die_arguments '}'**

Optional set of statements that specify information about the output layout. The following *die_arguments* are available:

- layout_path *layout_path* — Path to the output OASIS layout. You can specify a directory or a full file path.
- die_name *die_name* — Name of the output OASIS layout file.
- precision *precision* — Precision of the output OASIS layout.
- vertices {*x1 y1 ... xn yn*} — Polygon coordinates for generated markers in the OASIS output. These coordinates are not multiplied by the *precision* during generation.

- **-ignore [reg_exp_list]**
Optional argument set that specifies a list of regular expressions to match pins to ignore. If you specify -ignore without the *reg_exp_list* value, the default expression is {^NC.*\$}. The default expression searches for all pins that begin with the string “NC”.
- **-package package_name**
Optional name for the generated package layout.
- **-package_mapping ‘{’ package_arguments‘}’**
Optional set of statements that specify mapping information about a package design from the REFDES column of the spreadsheet file. The following *package_arguments* are available:
 - die_name *die_name* — Name of the die for which the mapping is given.
 - layer_number *layer_number* — Layer number of the mapped die. The *layer_number* must be greater than 2.
 - vertices {*x1 y1 ... xn yn*} — Polygon coordinates for the pin cells. These coordinates are not multiplied by the *precision* during generation.

Examples

```
calibre -ys -3dstack
3dstack::ss2oasis -ss spreadsheet.txt -die_info OASIS ./output \
    -package pkg -package_info BGA 3 3 3 -3 -3
exit
```

3dstack::ss2spice

Converts a CSV input file to a SPICE netlist.

Usage

3dstack::ss2spice

```
-ss ss_file
-spice spice_file
[-order column_list]
[-ignore [reg_exp_list]]
[-top_cell top_cell_name]
[-hier]
[-subckt_pins {NET_NAME | PIN_NUMBER | PIN_NAME}]
```

Arguments

- **-ss ss_file**
Required path to the spreadsheet (CSV) input file.
- **-spice spice_path**
Required path to the SPICE output file.
- **-order column_list**
Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF_DES, PIN_NUMBER, PIN_X, PIN_Y, PIN_NAME, and NET_NAME.
- **-ignore [reg_exp_list]**
Optional argument set that specifies a list of regular expressions to match pins to ignore. If you specify -ignore without the *reg_exp_list* value, the default expression is {^NC.*\$}. The default expression searches for all pins that begin with the string “NC”.
- **-top_cell top_cell_name**
Optional name of the top cell. The default value is TOPCELL_3DI.
- **-hier**
Optional argument that instructs the converter to generate a hierarchical SPICE netlist.
- **-subckt_pins {NET_NAME | PIN_NUMBER | PIN_NAME}**
Optional argument set that specifies the type of information specified for the pin column of the CSV file. The default is NET_NAME.

Examples

Generate a SPICE netlist from a spreadsheet CSV file as follows:

```
calibre -ys -3dstack
3dstack::ss2spice -ss input.csv \
    -spice output.sp \
    -top_cell BGA \
    -hier

exit
```

3dstack::xsi2gds

Creates GDS layouts that include pin and die boundary locations from a CSV file generated by the Mentor Graphics Xpedition® Substrate Integrator tool.

Usage

```
3dstack::xsi2gds -xsi xsi_file
  [-order column_list]
  [-die_info GDS '{'
    -layout_path layout_path
    [-die_name die_name]
    [-precision precision]
    [-vertices {x1 y1 ... xn yn}]
  '}' | -no_die_layout]...
  [-ignore [reg_exp_list]]
  [-package package_name]
  [-package_mapping '{'
    -die_name die_name
    -layer_number layer_number
    [-vertices {x1 y1 ... xn yn}]
  '}'...]
  [-layer_number layer_number]
  [-vertices {x1 y1 ... xn yn}]
  [-text_only labels_file
    [-labels {NET_NAME | PIN_NUMBER | PIN_NAME}]
    [-attach_to {component layer ...}]]]
```

Arguments

- **-xsi *xsi_file***

Required path to the Xpedition Substrate Integrator CSV input file.

- **-order *column_list***

Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF_DES, PIN_NUMBER, PIN_X, PIN_Y, PIN_NAME, NET_NAME, COMPONENT_NAME, and IGNORE. The default order is as follows:

```
NET_NAME IGNORE IGNORE COMPONENT_NAME PIN_NUMBER REF_DES
```

- **-die_info GDS '{*die_arguments*}'**

Optional set of statements that specify information about the output layout. The following *die_arguments* are available:

-layout_path *layout_path* — Path to the output GDS layout. You can specify a directory or a full file path.

-precision *precision* — Precision of the output GDS layout.

- vertices $\{x_1\ y_1 \dots x_n\ y_n\}$ — Polygon coordinates for generated markers in the GDS output. These coordinates are not multiplied by the *precision* during generation.
 - die_name *die_name* — Name of the output GDS layout file. This option is mutually exclusive with the other *die_arguments*.
 - -no_die_layout
Optional argument that specifies not to generate a layout for the die.
 - -ignore [*reg_exp_list*]
Optional argument set that specifies a list of regular expressions to match pins to ignore. If you specify -ignore without the *reg_exp_list* value, the default expression is $\{\text{^NC.}*\$$. The default expression searches for all pins that begin with the string “NC”.
 - -package *package_name*
Optional name for the generated package layout.
 - -package_mapping ‘{’ *package_arguments*‘}’
Optional set of statements that specify mapping information about a package design from the REFDES column of the spreadsheet file. The following *package_arguments* are available:
 - die_name *die_name* — Name of the die for which the mapping is given.
 - layer_number *layer_number* — Layer number of the mapped die. The *layer_number* must be greater than 2.
 - vertices $\{x_1\ y_1 \dots x_n\ y_n\}$ — Polygon coordinates for the pin cells. These coordinates are not multiplied by the *precision* during generation.
 - -text_only *labels_file*
Optional argument set that generates a text label file at the specified *labels_file* path. When this argument set is applied, the only output from the 3dstack::xsi2gds command is the text labels file. The text labels file uses the labels and locations defined in the *xsi_file*.
- The following options are exclusive to the -text_only argument set:
- labels {NET_NAME | PIN_NUMBER | PIN_NAME}
Optionally specifies the column in the *xsi_file* from which to retrieve text labels. The default is the NET_NAME column.
 - attach_to *layer* -component *component_name*
Specifies the layer of the layout and the component name to which the labels are attached.

Global Command Options

The following options are applied to all dies in the input netlist. The -package_mapping options take precedence over these options.

- `-layer_number layer_number`
Optional layer number used for all dies in the netlist. The *layer_number* must be greater than 2. If you specified `-layer_number` in the `-package_mapping` argument set, then this option is ignored for that specific die.
- `-vertices {x1 y1 ... xn yn}`
Optional set of polygon coordinates for generated markers for all dies in the GDS output. These coordinates are not multiplied by the *precision* during generation.

Examples

```
calibre -ys -3dstack  
3dstack::xsi2gds -xsi net.csv -die_info GDS ./output \  
-package pkg -package_info BGA 3 3 3 -3 -3
```

3dstack::xsi2oasis

Creates OASIS layouts that include pin and die boundary locations from a CSV file generated by the Mentor Graphics Xpedition® Substrate Integrator tool.

Usage

```
3dstack::xsi2oasis -xsi xsi_file
  [-order column_list]
  [-die_info OASIS '{'
    -layout_path layout_path
    [-die_name die_name]
    [-precision precision]
    [-vertices {x1 y1 ... xn yn}]
  '}' | -no_die_layout]...
  [-ignore [reg_exp_list]]
  [-package package_name]
  [-package_mapping '{'
    -die_name die_name
    -layer_number layer_number
    [-vertices {x1 y1 ... xn yn}]
  '}'...]
  [-layer_number layer_number]
  [-vertices {x1 y1 ... xn yn}]
  [-text_only labels_file
    [-labels {NET_NAME | PIN_NUMBER | PIN_NAME}]
    [-attach_to {component layer ...}]]]
```

Arguments

- **-xsi *xsi_file***

Required path to the Xpedition Substrate Integrator CSV input file.

- **-order *column_list***

Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF_DES, PIN_NUMBER, PIN_X, PIN_Y, PIN_NAME, COMPONENT_NAME, and NET_NAME. The default order is the following:

```
NET_NAME IGNORE IGNORE COMPONENT_NAME PIN_NUMBER REF_DES
```

- **-die_info OASIS '{'*die_arguments*'}'**

Optional set of statements that specify information about the output layout. The following *die_arguments* are available:

-layout_path *layout_path* — Path to the output OASIS layout. You can specify a directory or a full file path.

-die_name *die_name* — Name of the output OASIS layout file.

- precision *precision* — Precision of the output OASIS layout.
- vertices {*x1 y1 ... xn yn*} — Polygon coordinates for generated markers in the OASIS output. These coordinates are not multiplied by the *precision* during generation.
- -no_die_layout
 - Optional argument that specifies not to generate a layout for the die.
- -ignore [*reg_exp_list*]
 - Optional argument set that specifies a list of regular expressions to match pins to ignore. If you specify -ignore without the *reg_exp_list* value, the default expression is {^NC.*\$}. The default expression searches for all pins that begin with the string “NC”.
- -package *package_name*
 - Optional name for the generated package layout.
- -package_mapping ‘{’ *package_arguments*‘}’
 - Optional set of statements that specify mapping information about a package design from the REFDES column of the spreadsheet file. The following *package_arguments* are available:
 - die_name *die_name* — Name of the die for which the mapping is given.
 - layer_number *layer_number* — Layer number of the mapped die. The *layer_number* must be greater than 2.
 - vertices {*x1 y1 ... xn yn*} — Polygon coordinates for the pin cells. These coordinates are not multiplied by the *precision* during generation.

Global Command Options

The following options are applied to all dies in the input netlist. The -package_mapping options take precedence over these options.

- -layer_number *layer_number*
 - Optional layer number used for all dies in the netlist. The *layer_number* must be greater than 2. If you specified -layer_number in the -package_mapping argument set, then this option is ignored for that specific die.
- -vertices {*x1 y1 ... xn yn*}
 - Optional set of polygon coordinates for generated markers for all dies in the OASIS output. These coordinates are not multiplied by the *precision* during generation.
- -text_only *labels_file*
 - Optional argument set that generates a text label file at the specified *labels_file* path. When this argument set is applied, the only output from the 3dstack::xsi2oasis command is the text labels file. The text labels file uses the labels and locations defined in the *xsi_file*.

The following options are exclusive to the -text_only argument set:

-labels {NET_NAME | PIN_NUMBER | PIN_NAME}

Optionally specifies the column in the *xsi file* from which to retrieve text labels. The default is the NET_NAME column.

-attach_to *layer* -component *component_name*

Specifies the layer of the layout and the component name to which the labels are attached.

Examples

```
calibre -ys -3dstack  
  
3dstack::xsi2oasis -xsi net.csv -die_info OASIS ./output \  
-package pkg -package_info BGA 3 3 3 -3 -3  
  
exit
```

3dstack::xsi2spice

Creates a SPICE netlist from a CSV file generated by the Mentor Graphics Xpedition® Substrate Integrator tool.

Usage

```
3dstack::xsi2spice -xsi xsi_path -spice spice_path [-hier] [-ignore [reg_exp_list]]  
-order {'NET_NAME IGNORE COMPONENT_NAME PIN_NUMBER  
REF_DES PIN_X PIN_Y}'  
[-subckt_pins {NET_NAME | PIN_NUMBER | PIN_NAME}] [-top_cell top_cell_name]
```

Arguments

- **-xsi *xsi_path***
Required path to the Xpedition Substrate Integrator CSV input file.
- **-spice *spice_path***
Required path to the SPICE output file.
- **-hier**
Optional argument that instructs the converter to generate a hierarchical SPICE netlist.
- **-ignore [*reg_exp_list*]**
Optional argument set that specifies a list of regular expressions to match pins to ignore. If you specify -ignore without the *reg_exp_list* value, the default expression is {^NC.*\$}. The default expression searches for all pins that begin with the string “NC”.
- **-order {'NET_NAME IGNORE COMPONENT_NAME PIN_NUMBER REF_DES
PIN_X PIN_Y}'**
Optional argument set that specifies the order of the columns in the input CSV file. If this argument set is not specified, the default order is the following:

NET_NAME IGNORE IGNORE COMPONENT_NAME PIN_NUMBER REF_DES

Note



You must specify -order with PIN_X and PIN_Y to read pin locations from the XSI netlist.

- **-subckt_pins {NET_NAME | PIN_NUMBER | PIN_NAME}**
Optional argument set that specifies the type of information specified for the pin column of the CSV file. The default is PIN_NUMBER.
If you specify PIN_NAME and there are multiple pins that are connected to the same net and have the same name, then the tool ignores the repeated pins.
- **-top_cell *top_cell_name***
Optional name of the top cell. The default value is TOPCELL_3DI.

Examples

Generate a SPICE netlist from an Xpedition CSV file as follows:

```
calibre -ys -3dstack
3dstack::xsi2spice -xsi input.csv \
    -spice output.sp \
    -top_cell BGA \
    -hier
exit
```


Appendix A Examples

The Calibre 3DSTACK rule file contains assembly operations that build a model of your 3D-IC and a set of verification rules. This section contains rule file examples and a complete walk-through of the Calibre 3DSTACK flow.

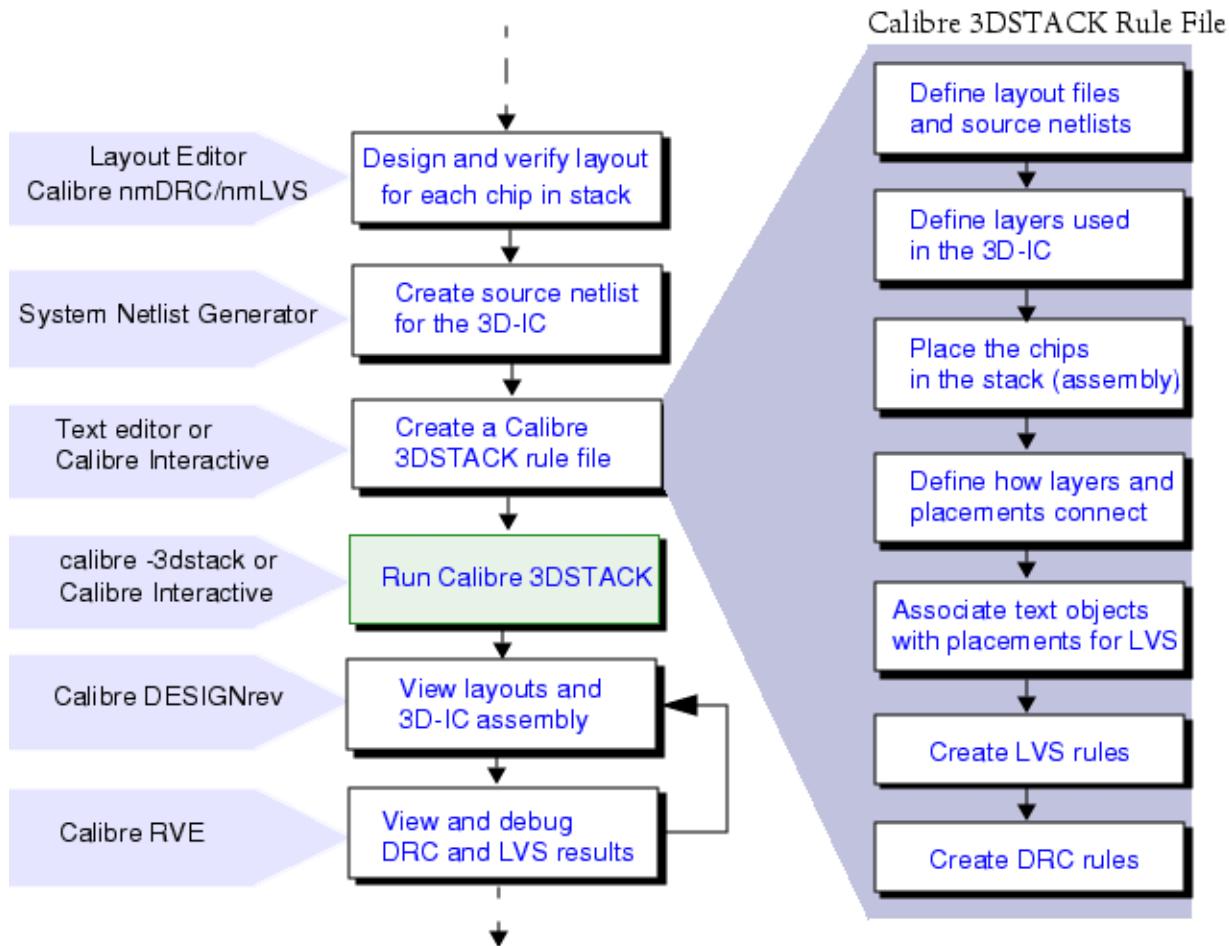
Table A-1. List of Examples

Topic	Description
Calibre 3DSTACK Flow Example	This example steps you through the complete flow for creating a new rule file for your 3D-IC.
System Netlist Generator Flow Example	Create a source netlist for an interposer design using the System Netlist Generator GUI. Design data is included.
Rule File Examples	The extended syntax is recommended for all new rule files.
Report File Format	The Calibre 3DSTACK report file is generated using the report command.

Calibre 3DSTACK Flow Example

This example steps you through the complete flow for creating a new rule file for your 3D-IC.

Figure A-1. Calibre 3DSTACK Rule Creation and Verification Flow



Example Design Information	302
Preparing the Dies for Assembly	303
Creating the Calibre 3DSTACK Rule File	304
Writing Assembly Operations in the Calibre 3DSTACK Rule File	306
Writing Verification Checks in the Calibre 3DSTACK Rule File	323

Example Design Information

The design used in this example is a 2.5D IC that consists of a memory controller and two identical ram dies stacked on a silicon interposer.

The controller and ram chips contain only the pad shapes and port text on layer 255. The full layout is not necessary for the example flow, but the dies are all assumed to be DRC and LVS

clean. The interposer contains landing pads and pin text for the three chips and metal redistribution layers to connect the controller to the two ram chips and the controller to the IO pads of the interposer. [Figure A-2](#) shows the input layouts before assembly and [Figure A-3](#) shows the final assembly of the stack.

Figure A-2. Design Inputs

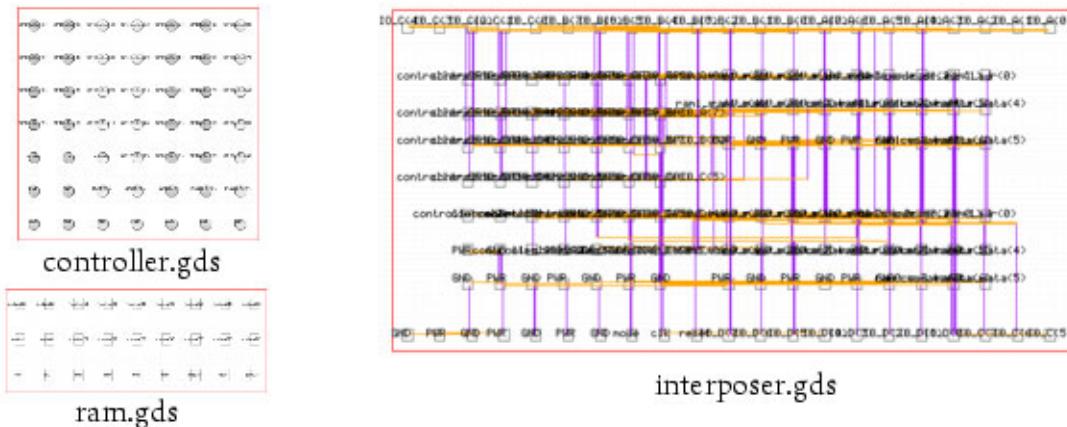
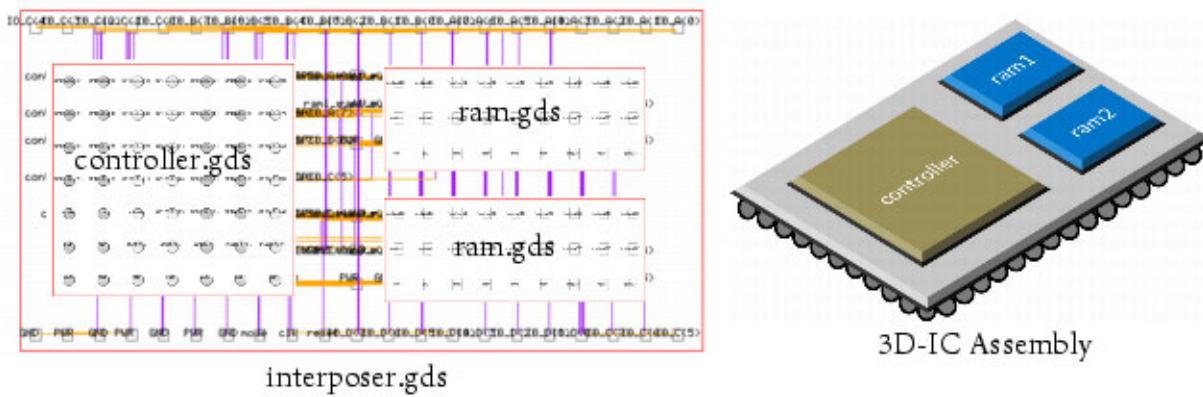


Figure A-3. Desired 3D-IC Assembly



Preparing the Dies for Assembly

Before you verify your 3D-IC with Calibre 3DSTACK, you must complete your layout and verify each die in the stack. The layout is usually performed with a combination of full custom layout tools and digital place and route tools.

Procedure

1. Design each die in the stack. Ensure that the TSV or interface pins are texted with the port names.

In this example, the controller, ram, and interposer layouts only include the pad shapes and pin text on layer 255. The full placement information is not necessary to assemble and verify your 3D-IC.

2. Run Calibre nmLVS and Calibre nmDRC on each die and perform corrections until they are completely LVS and DRC clean.
3. Extract the layout netlists for the dies.
4. Create a source netlist for your complete 3D-IC. The System Netlist Generator allows you to easily place chips, connect pins, and generate a full netlist for your 3D-IC. For a full example of the System Netlist Generator flow with executable data, see “[System Netlist Generator Flow Example](#)” on page 327.

If you do not have a source netlist, you can still verify the connectivity of your layout using internal text tracing.

5. Continue to “[Creating the Calibre 3DSTACK Rule File](#)” on page 304.

Creating the Calibre 3DSTACK Rule File

Create a new extended syntax rule file. The Calibre 3DSTACK rule file is a text file that defines run options, chips, assembly operations, and verification checks.

Procedure

1. Change to the directory where you want to assemble your 3D-IC.
2. Create a new text file. For example, *3dstack.rules*.
3. Add the following statement to the first line in your rule file:

```
#!3dstack+
```

This specifies that the rule file uses the extended Calibre 3DSTACK+ syntax.

4. Add the following command:

```
set_version -version 1.0
```

This sets the syntax version of the Calibre 3DSTACK description language. There is currently only one version of the 3D-IC description language syntax.

5. Save the file and continue to “[Writing Assembly Operations in the Calibre 3DSTACK Rule File](#)” on page 306.

Examples

```
#!3dstack+
#####
#
#           CALIBRE 3DSTACK+ Rule File for a 2.5D Interposer
#
#           v1.0 12/25/2017
#
#####
set_version -version 1.0
```

Writing Assembly Operations in the Calibre 3DSTACK Rule File

Before writing any rule checks for your 3D-IC, you must define the layouts, layers, location, and text for the dies, and the layer connectivity. These rule file components are called assembly operations.

Setting Up Global Options	306
Defining Dies Used in the Assembly.....	307
Defining Layers	309
Defining Layer Connectivity.....	317
Placing the Dies	319

Setting Up Global Options

The rule file supports a configuration command that sets up global properties and settings for the Calibre 3DSTACK assembly and verification process.

Prerequisites

- Completion of “[Creating the Calibre 3DSTACK Rule File](#)” on page 304.

Procedure

- Open your *3dstack.rules* file.
- After the `set_version` command statement, apply the `config` command as follows:

```
config \
```

Using a line continuation character improves the readability of the rule file.

- Apply the following options to the config command:

- Specify the top cell name for the assembly:

```
-layout_primary TOP_CELL \
```

- Specify the netlist that defines the connectivity of the assembly:

```
-source_netlist -file top_cell.spi -format SPICE -case YES \
```

Although optional, a source netlist is highly recommended for debugging issues in your design. Without a source netlist, Calibre 3DSTACK uses the attached text in the layout to trace connectivity by matching the text labels on each placement.

- Generate a detailed report of the run:

```
-report {-file output/3dstack.report} \
```

- d. Write out the extracted connectivity in Verilog format:

```
-export_connectivity {-file output/3dstack.v -format verilog } \
```

4. Save the file and continue to “[Defining Layers](#)” on page 309.

Results

The layout files are defined and ready to be placed in the assembly. The ideal electrical connectivity between the chips is read from the source netlist.

Examples

```
#!3dstack+
#####
#
#           CALIBRE 3DSTACK+ Rule File for a 2.5D Interposer
#
#           v1.0 12/25/2017
#
#####
set_version -version 1.0

#####
# Configure run
#####
config \
    -layout_primary TOP_CELL \
    -source_netlist {-file ./top_cell.spi -format SPICE } \
    -report {-file output/3dstack.report} \
    -export_connectivity {-file output/3dstack.v -format verilog }
```

Defining Dies Used in the Assembly

The first assembly operation in your rule file flow is to define the layouts that compose your 3D-IC.

In the example design shown in [Figure A-3](#), a controller and two memory chips are stacked on an interposer. The two ram chips use the same layout file (*ram.gds*), while the controller and interposer have their own layouts (*controller.gds* and *interposer.gds*, respectively).

Prerequisites

- DRC and LVS clean layout files.
- Completion of “[Creating the Calibre 3DSTACK Rule File](#)” on page 304.

Procedure

1. Open your *3dstack.rules* file.

2. After the config command entry, apply the `die` command to define the interposer die and specify its layout file and properties:

```
die -die_name interposer \
    -interposer \
    -layout { \
        -path ./design/interposer.gds \
        -type gdsii \
        -primary interposer \
        -depth all \
    } \
```

The name of the die must be unique. The `-interposer` option specifies that this die serves as an interposer for connectivity purposes. The `-layout` option specifies the path, layout type, and top cell name of the layout. The “`-depth all`” option instructs Calibre 3DSTACK to read in all layers of the layout.

Note

 If your die uses LEF/DEF data, then specify the `-lefdef` option instead of the `-layout` option.

3. Create die definitions for all dies used in your assembly. In this example, there is one controller and two identical ram dies stacked on the interposer:

```
die -die_name controller \
    -layout { \
        -path ./design/controller.gds \
        -type gdsii \
        -primary controller \
    } \

die -die_name ram \
    -layout { \
        -path ./design/ram.gds \
        -type gdsii \
        -primary ram \
    } \
```

Although the ram die is instantiated twice in the assembly, you only need to create a single die definition.

Note

 The `component` command enables you to define passive components, such as resistors and capacitors.

Results

The dies and their respective layouts have been defined, *but they are not yet finalized*. The next step is to define the physical layers and text used within the dies that interact with the assembly.

Examples

```

#!3dstack+
#####
# CALIBRE 3DSTACK+ Rule File for a 2.5D Interposer
#
# v1.0 12/25/2017
#
#####
set_version -version 1.0

#####
# Configure run
#####
config \
    -layout_primary TOP_CELL \
    -source_netlist {-file ./top_cell.spi -format SPICE } \
    -report {-file output/3dstack.report} \
    -export_connectivity {-file output/3dstack.v -format verilog }

#####
# Define dies in stack
#####

die -die_name interposer \
    -layout { \
        -path ./design/interposer.gds \
        -type gdsii \
        -primary interposer \
        -depth all \
    } \

die -die_name controller \
    -layout { \
        -path ./design/controller.gds \
        -type gdsii \
        -primary controller \
    } \

die -die_name ram \
    -layout { \
        -path ./design/ram.gds \
        -type gdsii \
        -primary ram \
    } \

```

Defining Layers

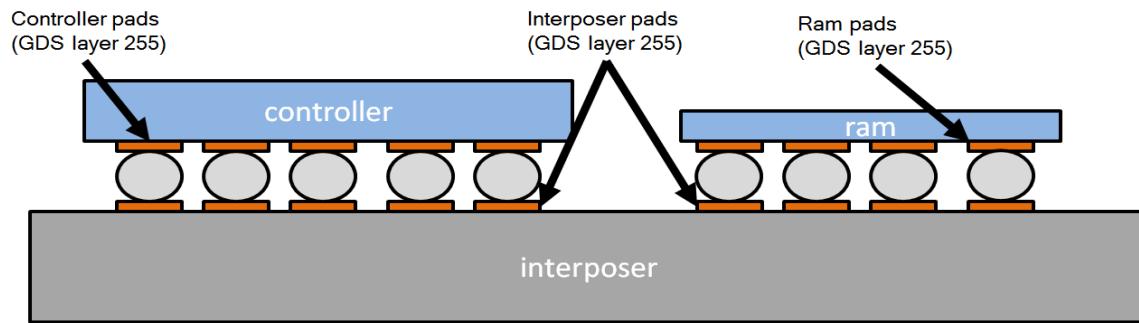
You must define all layers used in the assembly and verification of your 3D-IC. If your layout files include detailed placement and routing, at a minimum, you only need to define the interface layers used to connect the dies in the stack. This is referred to as a black-box

verification flow. You can perform full white-box verification flow if you define all layers in your layout files.

Note

Instead of applying the `-layer_info` argument as described in this topic, you can apply the die `-process` option and specify a predefined set of layer definitions.

In the example shown in [Figure A-3](#), the controller and ram die have bump pad shapes and text on layer 255. These bump pads connect to the pads on the interposer that are also on layer 255.



Prerequisites

- Completion of “[Setting Up Global Options](#)” on page 306

Procedure

- Open your `3dstack.rules` rule file.
- Find your first die definition for the interposer.

This statement defines the die and its layout, but it currently contains no layers:

```
interposer
```

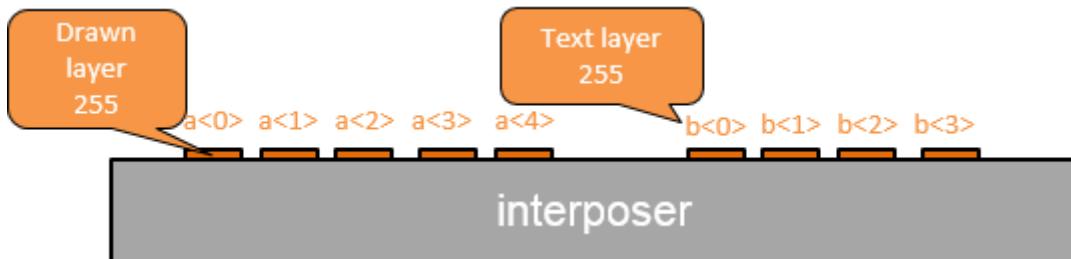
3. In your first die command for the interposer, using line continuation characters where necessary, apply the [-layer_info](#) option to add your first layer:

```
die -die_name interposer \
    -layout { \
        -path ./design/interposer.gds \
        -type gdsii \
        -primary interposer \
        -depth all \
    } \
    -layer_info { \
        -type PAD \
        -name INT_PAD \
        -ext_connect \
        -layer { \
            255 \
            -depth all \
        } \
        -text { 255 } \
        -bottom \
    } \
}
```

The highlighted addition to the die statement defines a layer named interposer_pads of type pad. The type is any user-defined string, but it is important because verification checks are controlled by the specified layer types. If you apply the same type to different layers, then you can check interactions between those layer types. For example, layers of type pad on the interposer die and layers of type pad on the controller die.

The pad layer and text in the *interposer.gds* belong to layer 255, so the -layer definition and -text definition both specify 255.

The -ext_connect option indicates that the layer is used for connectivity outside of the die to other interacting external layers.

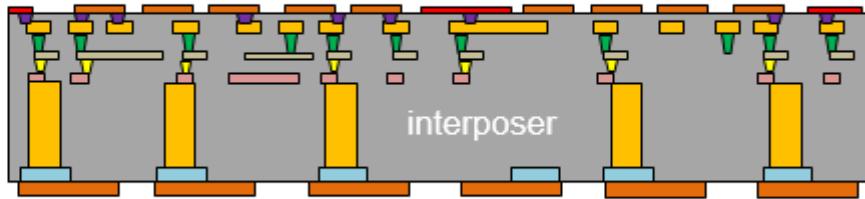


4. Specify the remaining layers for the interposer die.

The interposer has landing pad shapes and text on layer 255, but also has additional metal interconnect and vias that distribute connections between the dies. These metal and via shapes are on layers 11 through 17.

Layer Name Type GDS Layer

■	RDL_M1	RDL_M	11
■	RDL_V1	RDL_V	12
■	RDL_M2	RDL_M	13
■	RDL_V2	RDL_V	14
■	RDL_M3	RDL_M	15
■	RDL_V3	RDL_V	16
■	RDL_M4	RDL_M	17
■	INT_PAD	PAD	255
■	uPAD	uPAD	256
■	BS_RDL	BRDL_1	257
■	TSV	TSV	258
■	BUMP	BUMP	259



```

-layer_info { \
    -type RDL_M \
    -name RDL_M1 \
    -ext_connect \
    -layer { \
        11 \
        -depth all \
    } \
    -bottom \
} \
-layer_info { \
    -type RDL_V \
    -name RDL_V1 \
    -layer { \
        12 \
        -depth all \
    } \
    -bottom \
}
...

```

Tip

You can also specify derived layers using the `-svrf` argument to the `-layer_info` command.

5. Specify the layers for the other dies in the stack:

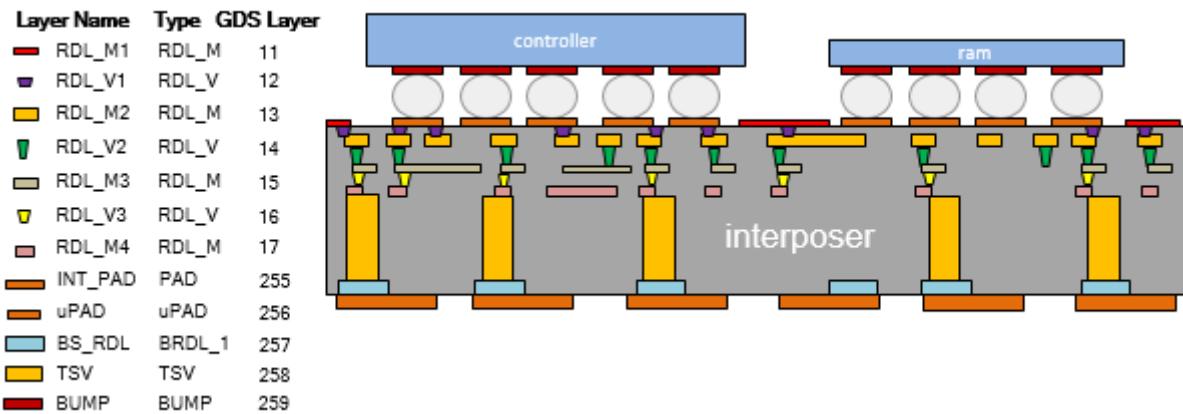
```

die -die_name controller \
    -layout { \
        -path ./design/controller.gds \
        -type gdsii \
        -primary controller \
    } \
    -layer_info { \
        -type BUMP \
        -name cont_c4 \
        -ext_connect \
        -layer { \
            255 \
        } \
        -text { 255 } \
        -bottom \
    }

die -die_name ram \
    -layout { \
        -path ./design/ram.gds \
        -type gdsii \
        -primary ram \
    } \
    -layer_info { \
        -type BUMP \
        -name ram_c4 \
        -ext_connect \
        -layer { \
            255 \
        } \
        -text { 255 } \
        -bottom \
    }

```

For the purpose of this example, the stacked dies are treated as black boxes, so only the interface layers (pads) that connect to the interposer are defined.



6. Save the file and continue to “[Defining Layer Connectivity](#)” on page 317.

Results

You have defined the pad layers from each of the chips in the stack. You have also defined the RDL layers of the interposer that are used to provide connectivity between the controller and ram dies.

Examples

```

#!3dstack+
#####
#
#           CALIBRE 3DSTACK+ Rule File for a 2.5D Interposer
#
#                   v1.0 12/25/2017
#
#####
set_version -version 1.0
#####
# Configure run
#####
config \
    -layout_primary TOP_CELL \
    -source_netlist {-file ./top_cell.spi -format SPICE } \
    -report {-file output/3dstack.report} \
    -export_connectivity {-file output/3dstack.v -format verilog }
#####
# Define dies in stack
#####
die -die_name interposer \
    -layout { \
        -path ./design/interposer.gds \
        -type gdsii \
        -primary interposer \
        -depth all \
    } \
    -layer_info { \
        -type PAD \
        -name INT_PADS \
        -ext_connect \
        -layer { \
            255 \
            -depth all \
        } \
        -text { 255 } \
        -bottom \
    } \
    -layer_info { \
        -type RDL_M \
        -name RDL_M1 \
        -ext_connect \
        -layer { \
            11 \
            -depth all \
        } \
        -bottom \
    } \
    -layer_info { \
        -type RDL_V \
        -name RDL_V1 \
        -layer { \
            12 \
            -depth all \
        } \
        -bottom \
    }

```

```
 } \
-layer_info { \
-type RDL_M \
-name RDL_M2 \
-layer { \
13 \
-depth all \
} \
-bottom \
} \
-layer_info { \
-type RDL_V \
-name RDL_V2 \
-layer { \
14 \
-depth all \
} \
-bottom \
} \
-layer_info { \
-type RDL_M \
-name RDL_M3 \
-layer { \
15 \
-depth all \
} \
-bottom \
} \
-layer_info { \
-type RDL_V \
-name RDL_V3 \
-layer { \
16 \
-depth all \
} \
-bottom \
} \
-layer_info { \
-type RDL_M \
-name RDL_M4 \
-layer { \
17 \
-depth all \
} \
-bottom \
}
```

```

die -die_name controller \
    -layout { \
        -path ./design/controller.gds \
        -type gdsii \
        -primary controller \
    } \
    -layer_info { \
        -type PAD \
        -name cont_c4 \
        -ext_connect \
        -layer { \
            255 \
        } \
        -text { 255 } \
        -bottom \
    }
}

die -die_name ram \
    -layout { \
        -path ./design/ram.gds \
        -type gdsii \
        -primary ram \
    } \
    -layer_info { \
        -type PAD \
        -name ram_c4 \
        -ext_connect \
        -layer { \
            255 \
        } \
        -text { 255 } \
        -bottom \
    } \
}

```

Defining Layer Connectivity

You must define layer interactions and connectivity within each die before assembling the 3D-IC.

Before connectivity verification, the Calibre 3DSTACK rule file must already define the die layouts and layers.

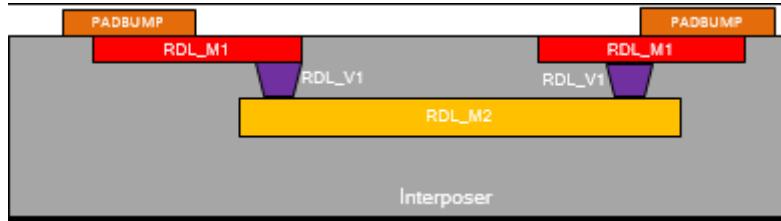
Prerequisites

- Completion of “[Defining Layers](#)” on page 309.
- Layer information (how layers and die placements are connected).

Procedure

1. Open your *3dstack.rules* rule file.
2. Find your first die definition for the interposer.

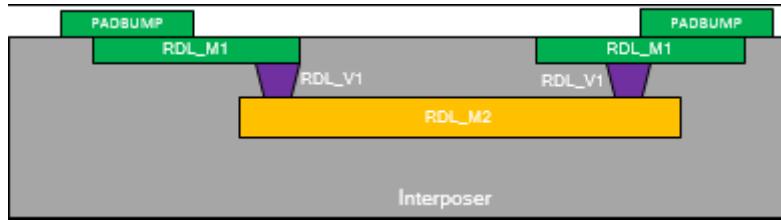
This statement defines the die, its layout, and all layers relevant to the composition of the stack, but it currently contains no information on how those layers electrically interact with each other.



3. In your first die command for the interposer, using line continuation characters where necessary, apply the `-wb_connect` option to define the connectivity between the pad and redistribution layer (RDL):

```
die -die_name interposer \
    -layout { ... } \
    -layer_info { ... } \
    ...
    -wb_connect PADBUMP RDL_M1
```

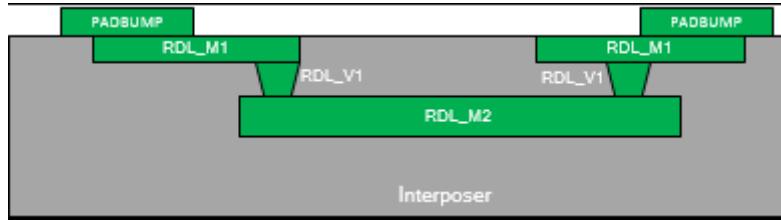
This statement specifies that the pad and first metal layer of the RDL are electrically connected.



4. Add statements that connect the rest of the RDL layers of the interposer:

```
die -die_name interposer \
    -layout { ... } \
    -layer_info { ... } \
    ...
    -wb_connect PADBUMP RDL_M1 \
    -wb_connect RDL_M1 RDL_M2 BY RDL_V1 \
    -wb_connect RDL_M2 RDL_M3 BY RDL_V2 \
    -wb_connect RDL_M3 RDL_M4 BY RDL_V3 \
```

These statements connect the M1 to M4 metal layers through the appropriate via layers.



5. Save the rule file.
6. Continue to “[Placing the Dies](#)” on page 319.

Results

The layers and placements defined earlier in the rule file are now electrically connected to match the layer design intent. The other chips in this example stack do not have any internal layers since they are treated as black boxes. If your dies contain internal layers and you want to verify them, then you must apply the -wb_connect operation to each die.

Placing the Dies

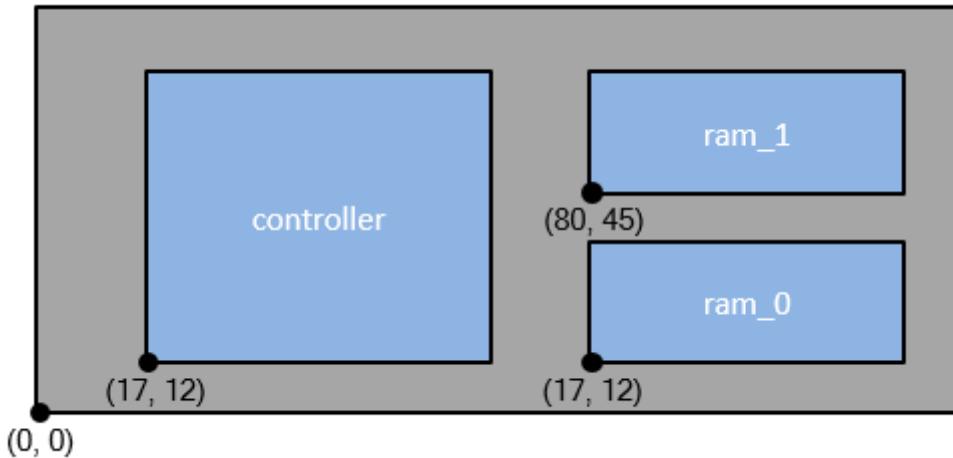
Assemble the 3D-IC by placing, rotating, and arranging the chips. This is the first physical action in the rule file. The coordinates of each die should be made available by the layout designer.

In order for the pads on the chips to align with the landing pads on the interposer (shown in [Figure A-3](#)) the placements must have the following lower-left coordinates:

Table A-2. Die Placement Coordinates

Placement Name	X (um)	Y (um)
controller	17	12
ram_0	80	12
ram_1	80	45
interposer	0	0

Figure A-4. Example Placement Coordinates



Prerequisites

- Completion of “[Defining Layer Connectivity](#)” on page 317.

Procedure

1. Open your *3dstack.rules* rule file.
2. Add the stack command after the die definitions as follows:

```
stack -stack_name assembly \
-die { \
    -name interposer
    -source interposer
    -placement 0 0
    -invert
} \
-tier {
    -die {-name controller -placement 17 12 -source controller }
    -die {-name ram -placement 80 12 -source ram_0 }
    -die {-name ram -placement 80 45 -source ram_1 }
}
```

A tier defines a set of dies that occupy the same horizontal plane. Since the interposer is at the bottom of the stack, it occupies a different vertical plane than the three chips stacked on top of it.

The -source argument specifies the corresponding sub-circuit cell name in the source netlist if it is different than the die name.

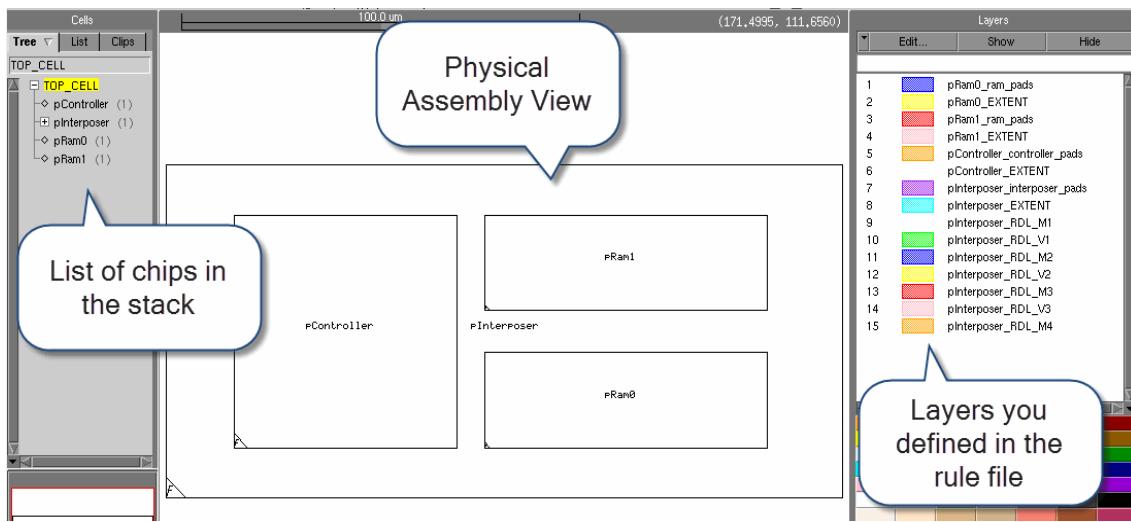
3. Save the rule file and close it.
4. Invoke Calibre 3DSTACK and generate the overlay view of the 3D-IC by entering the following command:

```
calibre -3dstack -create_assembly 3dstack.assembly 3dstack.rules
```

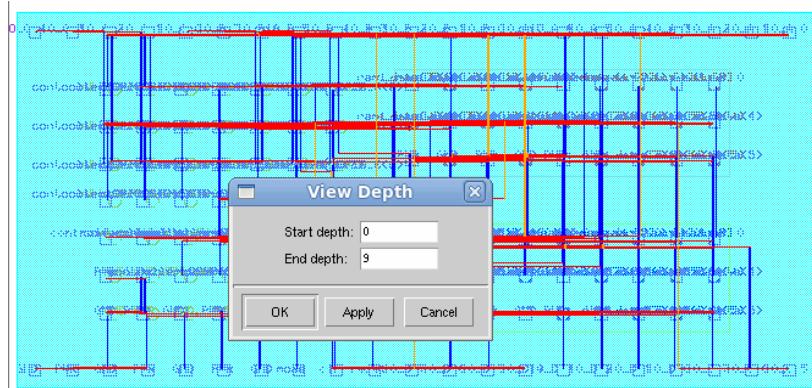
The -create_assembly invocation argument only generates the physical view of the stacked ICs. It does not perform any verification checks. The physical overlay view is useful for checking the layout, layer, and placement definitions in your rule file without performing a full run.

5. Open the assembly in Calibre DESIGNrev using the following command:

```
calibredrv 3dstack.assembly.oas
```



6. Press 9 to change the view depth.



This allows you to see the detailed layout within the placements.

7. Close Calibre DESIGNrev and continue to “[Defining Layer Connectivity](#)” on page 317.

Results

You physically placed the chips in the 3D-IC assembly. You can check for placement errors as described under “[Writing Verification Checks in the Calibre 3DSTACK Rule File](#)” on page 323

Examples

```
...
#####
# Arrange chips in 3DSTACK
#####
stack -stack_name assembly \
-die { \
    -name interposer
    -source pInterposer
    -placement 0 0
    -invert
} \
-tier {
    -die {-name controller -placement 17 12 -source pController }
    -die {-name ram -placement 80 12 -source pRam0 }
    -die {-name ram -placement 80 45 -source pRam1 }
}
...
...
```

Writing Verification Checks in the Calibre 3DSTACK Rule File

LVS and DRC rules are written using Calibre 3DSTACK rule file commands.

Checking Connectivity (LVS).....	323
Checking Layer Spacing (DRC).....	324
Checking For Sufficient Pad Overlap (DRC)	325

Checking Connectivity (LVS)

Connectivity checks are performed on your 3D-IC using the placement information, source netlist connectivity, assembly connect statements, and attached text.

To verify the connectivity between chip placements, you use the [connected](#) rule file command. During a Calibre 3DSTACK verification run, the net connectivity between layout instances in your assembly is generated based on the layer stackup design you specified with the [connect](#) statement. Any text is also attached to the layout pins and traced to the pin names on each placement. If you specified a source netlist, the original connectivity is compared to the extracted netlist.

This is a continuation of the assembled design shown in [Figure A-3](#).

Note

 In the case where multiple text labels overlap a pad in the layout, the tool generates a multi-text error and chooses to use one of the text labels attached to the pad for the connectivity analysis. Because the chosen label may not match your design intent, it is important to review and resolve all multi-text errors to ensure that the layout is correct. If you do not resolve the multi-text errors, your connectivity analysis may not be correct.

Prerequisites

- Completion of “[Defining Layer Connectivity](#)” on page 317.

Procedure

- Open your *3dstack.rules* rule file.
- Add the [connected](#) statements after the assembly statements as follows:

```
connected -check_name CONNECT_PADS_TO_BUMPS \
    -layer_type1 pad \
    -layer_type2 bump \
    -black_box \
    -net_mismatch ALL \
    -comment "CONNECT::INTERPOSER TO DIES CONNECTIVITY"
```

Each rule check must have a unique name.

The `-layer_type1` and `-layer_type2` commands specify the types of layers that should be connected. The types are specified in the individual die definitions.

The `-detailed` option writes out additional information to the report file.

Note

 You can also check connectivity between two dies by specifying the die names with the connected ... `-die1 name` and `-die2 name` options.

3. Save the rule file.
4. Continue to “[Checking Layer Spacing \(DRC\)](#)” on page 324.

Results

Connectivity rules are established. Calibre 3DSTACK uses the layer connectivity and text on the port shapes (pins) to determine the net connectivity between dies. If a source netlist is specified, the tool compares the connectivity extracted from the layout to the source connectivity. If no source netlist is specified (not recommended), you must specify the text layers for the dies in your assembly. In this case, the comparison is performed by text matching only (port text on the placements should match).

Checking Layer Spacing (DRC)

Calibre 3DSTACK includes a number of physical checks that operate similar to SVRF spacing checks. It is recommended that you use the internal Calibre 3DSTACK commands to check for spacing violations.

The `enclosure`, `external`, and `internal` commands provide basic spacing checks.

Prerequisites

- Completion of “[Writing Assembly Operations in the Calibre 3DSTACK Rule File](#)” on page 306.

Procedure

1. Open your `3dstack.rules` rule file.
2. To check spacing between dies, add `external` statements after the connected statements as follows:

```
external -check_name PAD_SPACE -layer_type1 pad \
-constraint "< 65 REGION" -comment "ERR:Placement within 65um!"

external -check_name BUMP_SPACE -layer_type1 bump \
-constraint "< 65 REGION" -comment "ERR:Placement within 65um!"
```

Each rule check must have a unique name.

3. To check spacing between other layers, such as the metal interconnect on the interposer, specify the following commands:

```
external -check_name RDL_M1.1 -layer_type1 rdl \
    -constraint "< 1 REGION" -comment "ERR: Spacing < 0.1um on rdl!" \
    ...
```

4. Add **enclosure**, **external**, and **internal** commands for each layer as appropriate.
5. Save the rule file.
6. Continue to “[Checking For Sufficient Pad Overlap \(DRC\)](#)” on page 325.

Checking For Sufficient Pad Overlap (DRC)

If a chip is placed slightly off of its target, or the geometry of the pads do not match exactly between chips, the pads may still be electrically connected, but not acceptable for manufacturing. The overlap tolerance must be checked to avoid false connected check reports.

Prerequisites

- Completion of “[Writing Assembly Operations in the Calibre 3DSTACK Rule File](#)” on page 306.

Procedure

1. Open your *3dstack.rules* rule file.
2. To check for sufficient pad overlap by the percentage of the overlap, add **overlap** commands after the assembly operations as follows:

```
overlap -check_name "OVERLAP_PAD_TO_BUMP" \
    -layer_type1 pad \
    -layer_type2 bump \
    -constraint "< 99" -intersection \
    -comment "Pad overlap must be greater than 90%!"
```

In this case, if the overlap between the controller and interposer pads is not greater than 90%, then the rule check fails.

3. To specify the pad overlap by the area of the overlap, use the **-by_area** command as follows:

```
overlap -check_name "OVERLAP_PAD_TO_BUMP_AREA" \
    -layer_type1 pad \
    -layer_type2 bump \
    -constraint "< 13" -intersection \
    -comment "Pad overlap must be greater than 13 um^2!"
```

In this case, if the overlap between the controller and interposer pads is not greater than 13 um², then the rule check fails.

4. Save the rule file and close it.

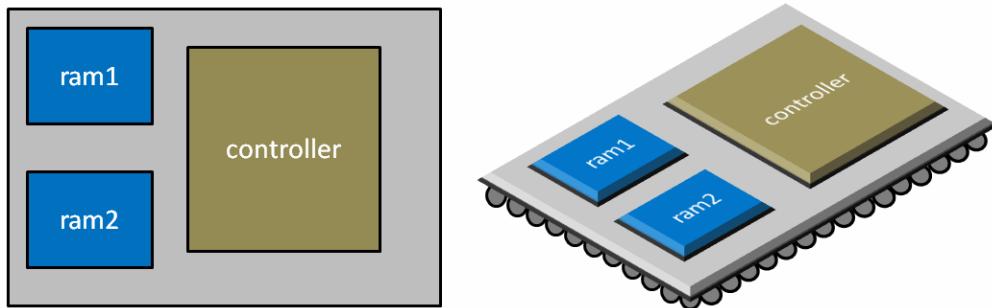
5. Continue to “[Running a Calibre 3DSTACK Verification from the Command Line](#)” on page 29.

System Netlist Generator Flow Example

Create a source netlist for an interposer design using the System Netlist Generator GUI. Design data is included.

Two identical RAM memory chips and a controller are stacked on an interposer as shown in [Figure A-5](#). Note, the interposer netlist is left out for simplicity.

Figure A-5. System Netlist Generator Interposer Design



The SPICE netlists for the memory and controller are defined in [Table A-3](#).

Table A-3. Netlists for Chips Used In This Example

SPICE Filename	Contents of SPICE Netlists
<i>myram_netlist.spi</i>	<pre>.SUBCKT myram mode enable adr[0] adr[1] adr[2] w_data[0] w_data[1] w_data[2] w_data[3] w_data[4] w_data[5] w_data[6] w_data[7] r_data[0] r_data[1] r_data[2] r_data[3] r_data[4] r_data[5] r_data[6] r_data[7] .ENDS myram</pre>
<i>controller_netlist.spi</i>	<pre>.SUBCKT controller reset mode \$--Chip enable bus-- ENABLE<0> ENABLE<1> \$--Address bus-- ADDR<0> ADDR<1> ADDR<2> \$--Port A+B-- GPIO_A<0> GPIO_A<1> GPIO_A<2> GPIO_A<3> GPIO_A<4> GPIO_A<5> GPIO_A<6> GPIO_A<7> GPIO_B<0> GPIO_B<1> GPIO_B<2> GPIO_B<3> GPIO_B<4> GPIO_B<5> GPIO_B<6> GPIO_B<7> \$--Port C+D-- GPIO_C<0> GPIO_C<1> GPIO_C<2> GPIO_C<3> GPIO_C<4> GPIO_C<5> GPIO_C<6> GPIO_C<7> GPIO_D<0> GPIO_D<1> GPIO_D<2> GPIO_D<3> GPIO_D<4> GPIO_D<5> GPIO_D<6> GPIO_D<7> .ENDS controller</pre>

The desired net connectivity of the chips on the interposer design is defined in [Table A-4](#).

Table A-4. Net Connectivity of the Interposer Design

Chip Placement	Placement Port	Net
ram1	mode	mode
	enable	enable_1
	adr (3-bit)	address
	w_data (8-bit)	write_data1
	r_data (8-bit)	read_data1
ram2	mode	mode
	enable	enable_2
	adr (2-bit)	address
	w_data (8-bit)	write_data2
	r_data (8-bit)	read_data2
controller	reset	reset
	mode	mode
	ENABLE<0>	enable_1
	ENABLE<1>	enable_2
	ADDR (2-bit)	address
	GPIO_A (8-bit)	write_data1
	GPIO_B (8-bit)	read_data1
	GPIO_C (8-bit)	write_data2
	GPIO_D (8-bit)	read_data2

This data is used throughout the following procedures:

Creating a New Project	328
Importing Chips into the Database	329
Modifying and Adding Pins on Chips	331
Creating Placements for the Chips.....	332
Defining Net Connectivity.....	334
Exporting the Complete Design	336

Creating a New Project

Launch the System Netlist Generator GUI and create a new project.

Prerequisites

- You have the necessary licenses and software as described under “[Requirements](#)” on page 16.

Procedure

1. Enter the following command at the command line to invoke the System Netlist Generator:

```
$CALIBRE_HOME/bin/sng -gui
```

2. Choose **File > New**.

The previously disabled menu icons () are now enabled. This indicates that an active database is loaded and is ready to be modified.

3. Choose **File > Save** and enter a new name for the project (for example, TOP_3D_sng).

The console at the bottom of the main window can be used to enter supported Tcl commands and any command listed under “[System Netlist Generator Commands](#)” on page 240. It also reports information, warnings, and errors generated by the tool.

Related Topics

[System Netlist Generator Flow Example](#)

[Workflow](#)

[sng](#)

[sng::new_db](#)

[sng::open_db](#)

[sng::save_db](#)

Importing Chips into the Database

Import SPICE netlists and create chips for the design.

When you import a chip netlist into an active System Netlist Generator project, a SPICE subcircuit is defined for that chip. You can view the pins and properties of the chips in the Chips tree.

Note

Set the MGC_SNG_SPICE_IMPORT_TEMP_DIR environment variable to the path of a temporary directory if you are importing large netlists.

Prerequisites

- You have netlists to import or you have saved the *myram_netlist.spi* and *controller_netlist.spi* netlists in [Table A-3](#) to two separate files.

Procedure

1. Choose **File > Import Chips** to load the SPICE netlists into the active database.
The Import Chips pane opens in a tab in the Console area at the bottom of the main window.
2. Click the **Add Row** button.
3. Enter the Chip Name, Cell Name, Type, File Location, and Bus Characters of the netlist as shown:

Chips to Import							
	Chip Name	Cell Name	Type	File Location	Bus Detection	Bus Characters	Bus Pattern
1	controller	controller	SPICE	./controller_netlist.spi		Yes	\<\>
2	ram	myram	SPICE	./myram_netlist.spi		Yes	\\

Note, the controller and ram chips use different characters to denote a bus. The Bus columns are described under “[sng::import_chip](#)” on page 271.

4. Click the **Import chips** button.

Any syntax errors in the source netlists are immediately reported in the console.

If you make a mistake and want to import the source files again, choose **Edit > Remove Chips** and select the items that you want to delete from the project. After removing the chips, you can then follow steps 1 to 4 again.

5. Save the database when you are done.

Results

The imported chips are now displayed in the Chips tree as shown here:

The screenshot shows the Calibre interface with the 'Database' tree on the left. Under 'Chips', 'Chips View' is selected, showing two entries: 'controller' and 'ram'. To the right is a table with two rows:

	name	cell_name
1	controller	controller
2	ram	myram

Examples

The following statements achieve the same steps using the Tcl console prompt:

```
sng::sng::import_chip {sng::database} -chip_name {controller} \
-cell_name {controller} -type {SPICE} -path {./controller_netlist.spi} \
-bus_detection {Yes} -bus_chars {\<\>}

sng::import_chip {sng::database} -chip_name {ram} -cell_name {myram} \
-type {SPICE} -path {./myram_netlist.spi} -bus_detection {Yes} \
-bus_chars {\[\]}
```

Related Topics

[System Netlist Generator Flow Example](#)

[Workflow](#)

[sng::import_chip](#)

[sng::remove_chip](#)

Modifying and Adding Pins on Chips

Modify pins on chip definitions.

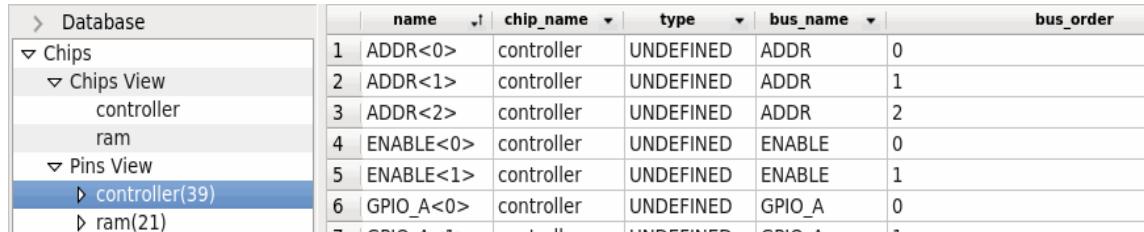
Prerequisites

- You have imported at least one chip definition as described under “[Importing Chips into the Database](#)” on page 329.

Procedure

1. Expand the Chips tree and the Pins View tree.
2. Click on the chip you want to modify under the Pins View. For this example, click on the controller chip.

Properties of all pins are listed in the table to the right of the Chips tree. You can directly edit the columns in the table to modify the pin properties.



The screenshot shows the Calibre 3DSTACK interface. On the left, there is a tree view labeled "Database". Under "Chips", there are "Chips View" and "Pins View". "Chips View" has nodes for "controller" and "ram". "Pins View" has a node for "controller(39)". On the right, there is a table with the following data:

	name	chip_name	type	bus_name	bus_order
1	ADDR<0>	controller	UNDEFINED	ADDR	0
2	ADDR<1>	controller	UNDEFINED	ADDR	1
3	ADDR<2>	controller	UNDEFINED	ADDR	2
4	ENABLE<0>	controller	UNDEFINED	ENABLE	0
5	ENABLE<1>	controller	UNDEFINED	ENABLE	1
6	GPIO_A<0>	controller	UNDEFINED	GPIO_A	0
7	GPIO_A<1>	controller	UNDEFINED	GPIO_A	1

3. For the reset pin in this example, change the type to INPUT.
4. Choose **Edit > Add Pins** to add a pin to the chip.

Note

Alternatively, you can right-click in the Pins View and choose **Add Pins**. The Add Pins pane opens as a tab at the bottom of the main window.

5. Click **Add Row**, enter the pin name and chip name, and then click **Add Pins** to insert the new pin on the chip.
6. Save the database when you are done.

Results

You have added or modified a pin on a chip. This allows you to modify the netlist of the source chips without manually modifying and importing the SPICE netlists.

Examples

Console equivalent commands to list all of the pins on a chip.

```
set my_db [sng::open_db -path ./TOP_3D_sng]
set controller_pins [sng::get_pins my_db -chip_name controller]
sng::get_property $controller_pins -property "pin_name"
```

Related Topics

[System Netlist Generator Flow Example](#)

[Workflow](#)

[sng::add_pin](#)

[sng::remove_pin](#)

[sng::set_property](#)

[sng::get_property](#)

Creating Placements for the Chips

Create placements, or instantiations, of your chips in the System Netlist Generator project.

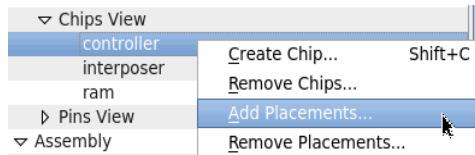
This is the netlist equivalent of the [place_chip](#) assembly command.

Prerequisites

- You have imported at least one chip definition as described under “[Importing Chips into the Database](#)” on page 329.

Procedure

1. Right-click on the controller in the Chips tree and choose **Add Placements** to instantiate the controller chip.



The Add Placement dialog opens as a tab at the bottom of the main window.

2. Click **Add Row** to start a new placement.
3. Enter a placement name for the controller (each instantiation must have a unique name) and click **Add Placements**.



The *pController* placement now appears under the Assembly tree as an instantiated chip. The pins on the chip are converted to ports on the placement.

Database Tree:

- > Database
- > Chips
- > Assembly
 - > Ports View
 - > pController(39)

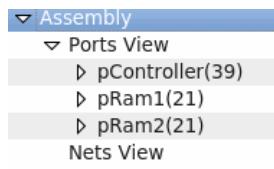
Table Data:

	name	placement...	type	bus_name	bus_order	net_name
1	pController::...	pController	INPUT		-1	reset
2	pController::...	pController	UNDEFINED		-1	mode
3	pController::...	pController	UNDEFINED	ENABLE	0	enable_1
4	pController::...	pController	UNDEFINED	ENABLE	1	enable_2
5	pController::...	pController	UNDEFINED	ADDR	0	address[0]
6	pController::...	pController	UNDEFINED	ADDR	1	address[1]
7	pController::...	pController	UNDEFINED	ADDR	2	address[2]
8	pController::...	pController	UNDEFINED	GPIO_A	0	read_data1[0]
9	pController::...	pController	UNDEFINED	GPIO_A	1	read_data1[1]
10	pController::...	pController	UNDEFINED	GPIO_A	2	read_data1[2]
11	pController::...	pController	UNDEFINED	GPIO_A	3	read_data1[3]

4. Create the placements for the two memory chips using the same method.
5. If you make any errors, you can remove unwanted placements by selecting **Edit > Remove Placements**.
6. Save the database.

Results

The three placements on the interposer are now defined in the project. You can view the placements and port properties by selecting the placements in the Assembly tree.



Examples

```
sng::create_placement my_db -placement_name {pController} -chip_name {controller}  
sng::create_placement my_db -placement_name {pRam2} -chip_name {ram}  
sng::create_placement my_db -placement_name {pRam1} -chip_name {ram}
```

Related Topics

[System Netlist Generator Flow Example](#)

[Workflow](#)

[sng::create_placement](#)

[sng::remove_placement](#)

Defining Net Connectivity

Electrically connect the placements in your design by defining and assigning nets in the database.

Prerequisites

- You have imported at least one chip definition as described under “[Importing Chips into the Database](#)” on page 329.
- You have created at least one placement in your design as described under “[Creating Placements for the Chips](#)” on page 332.

Procedure

1. Choose **Edit > Add Nets**.
2. Click **Add Row**.
3. Type “reset” in the net_name field and pController::reset in the ports_list field.
4. Click **Add Nets**.

5. Select the pController placement and note that the port in the properties table is now connected to the reset net in the net_name column:

	name	placement...	type	bus_name	bus_order	net_name
1	pController...	pController	INPUT		-1	reset
2	pController...	pController	UNDEFINED		-1	mode
3	pController...	pController	UNDEFINED	ENABLE	0	enable_1
4	pController...	pController	UNDEFINED	ENABLE	1	enable_2

For placements with large pin counts, it is recommended to use the Tcl console interface to define the connectivity.

6. Save the database and exit the tool.
7. Enter the following Tcl commands in a shell script to connect the four general purpose 8-bit data ports and address port of the controller to the 8-bit read, write, and address ports of the two ram placements:

```
#set db_instance to current project
set my_db [sng::open_db -path ./TOP_3D_sng]

#Connect read data from controller to pRam1
sng::connect my_db -net_name read_data1 \
-buses {pController::GPIO_A pRam1::r_data}

#Connect write data from controller to pRam1
sng::connect my_db -net_name write_data1 \
-buses {pController::GPIO_B pRam1::w_data}

#Connect read data from controller to pRam2
sng::connect my_db -net_name read_data2 \
-buses {pController::GPIO_C pRam2::r_data}

#Connect write data from controller to pRam2
sng::connect my_db -net_name write_data2 \
-buses {pController::GPIO_D pRam2::w_data}

#Connect address bus to all three chips
sng::connect my_db -net_name address \
-buses {pController::ADDR pRam1::adr pRam2::adr}

sng::save_db my_db -path ./TOP_3D_sng
```

8. Save the script as *connect.tcl* and close it.
 9. Source the Tcl script using the following command:
- ```
sng -batch -script connect.tcl
```
10. Invoke the System Netlist Generator and load the TOP\_3D\_sng database. The Assembly tree now displays all of the new connected nets and buses on the three chips.

11. Complete the remaining connections manually using [Table A-4](#) on page 328.
12. Save the database.

## Results

The Assembly tree and placements are populated with all of the bus net connections. The mode and enable connections are unconnected and can be connected using the GUI or a batch script.

## Related Topics

[System Netlist Generator Flow Example](#)

[Workflow](#)

[sng::get\\_ports](#)

[sng::incr](#)

[sng::connect](#)

[sng::unconnect](#)

# Exporting the Complete Design

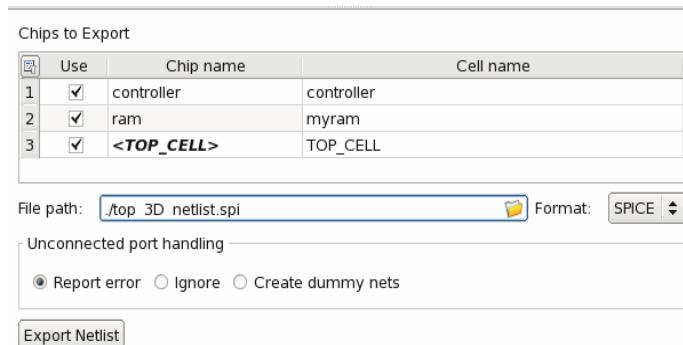
Export the completed source netlist from the System Netlist Generator.

## Prerequisites

- A completed System Netlist Generator database that defines all of the chips, placements, and net connectivity of your 3D-IC.

## Procedure

1. Save the current database or open an existing database.
2. Select **File > Export Netlist** or click on the (  ) icon at the top of the main window.  
The Export Netlist dialog opens at the bottom of the main window.
3. Enable all of the chips that you want to export and specify a path for the SPICE file:



4. Click **Export Netlist**.

## Results

The source netlist for the 3D-IC, shown in [Figure A-5](#) on page 327, was created using the System Netlist Generator. You can use this ideal electrical representation of your 3D-IC in a Calibre 3DSTACK run using the [source\\_netlist](#) command. This allows you to verify your design intent with the extracted layout.

## Examples

The batch command equivalent of this procedure is as follows:

```
sng:::export_netlist my_db -chip_name_list "myram1 controller" \
 -path ./top_3d_netlist.spi
```

The following SPICE netlist is the completed assembly netlist for this design:

```
.SUBCKT controller reset mode ENABLE<0> ENABLE<1> ADDR<0> ADDR<1> ADDR<2>
+ GPIO_A<0> GPIO_A<1> GPIO_A<2> GPIO_A<3> GPIO_A<4> GPIO_A<5> GPIO_A<6>
+ GPIO_A<7> GPIO_B<0> GPIO_B<1> GPIO_B<2> GPIO_B<3> GPIO_B<4> GPIO_B<5>
+ GPIO_B<6> GPIO_B<7> GPIO_C<0> GPIO_C<1> GPIO_C<2> GPIO_C<3> GPIO_C<4>
+ GPIO_C<5> GPIO_C<6> GPIO_C<7> GPIO_D<0> GPIO_D<1> GPIO_D<2> GPIO_D<3>
+ GPIO_D<4> GPIO_D<5> GPIO_D<6> GPIO_D<7>
.ENDS controller

.SUBCKT myram mode enable adr[0] adr[1] adr[2] w_data[0] w_data[1]
w_data[2]
+ w_data[3] w_data[4] w_data[5] w_data[6] w_data[7] r_data[0] r_data[1]
+ r_data[2] r_data[3] r_data[4] r_data[5] r_data[6] r_data[7]
.ENDS myram

.SUBCKT TOP_CELL
XpController reset mode enable_1 enable_2 address[0] address[1] address[2]
+ read_data1[0] read_data1[1] read_data1[2] read_data1[3] read_data1[4]
+ read_data1[5] read_data1[6] read_data1[7] write_data1[0] write_data1[1]
+ write_data1[2] write_data1[3] write_data1[4] write_data1[5]
write_data1[6]
+ write_data1[7] read_data2[0] read_data2[1] read_data2[2] read_data2[3]
+ read_data2[4] read_data2[5] read_data2[6] read_data2[7] write_data2[0]
+ write_data2[1] write_data2[2] write_data2[3] write_data2[4]
write_data2[5]
+ write_data2[6] write_data2[7] controller
XpRam1 mode enable_1 address[0] address[1] address[2] write_data1[0]
+ write_data1[1] write_data1[2] write_data1[3] write_data1[4]
write_data1[5]
+ write_data1[6] write_data1[7] read_data1[0] read_data1[1] read_data1[2]
+ read_data1[3] read_data1[4] read_data1[5] read_data1[6] read_data1[7]
myram
XpRam2 mode enable_2 address[0] address[1] address[2] write_data2[0]
+ write_data2[1] write_data2[2] write_data2[3] write_data2[4]
write_data2[5]
+ write_data2[6] write_data2[7] read_data2[0] read_data2[1] read_data2[2]
+ read_data2[3] read_data2[4] read_data2[5] read_data2[6] read_data2[7]
myram
.ENDS TOP_CELL
```

## Related Topics

[System Netlist Generator Flow Example](#)

[Workflow](#)

[sng::export\\_netlist](#)

[source\\_netlist](#)

[Specifying a Source Netlist](#)

# Rule File Examples

The extended syntax is recommended for all new rule files.

|                                                |     |
|------------------------------------------------|-----|
| Calibre 3DSTACK+ Extended Syntax Example ..... | 339 |
| Standard (Legacy) Syntax Example 1 .....       | 345 |
| Standard (Legacy) Syntax Example 2 .....       | 347 |

## Calibre 3DSTACK+ Extended Syntax Example

Extended syntax rule files are more flexible than conventional Calibre 3DSTACK rule files because the assembly operations can be separate from the rules.

The following example creates a simple 2.5D IC with a controller die and two memory dies stacked on a passive interposer.

### Opening Statements

Extended syntax rules must begin with the following line:

```
#!3dstack+
```

The next statement must be the `set_version` command to set the syntax version of the rule file. The only supported version is “1.0”

```
#!3dstack+
#####
#
CALIBRE 3DSTACK+ Rule File for 2.5D Interposer
#
#####
set_version -version 1.0
```

### Calibre 3DSTACK Settings

Apply the `config` command to set up layout, connectivity, and output options. In this example, the rule file provides a name for the generated stack assembly, specifies a source netlist for the complete assembly (highly recommended), specifies that a detailed report be output, and exports the generated assembly netlist to a Verilog file.

```
#####
Configure run
#####
config \
 -layout_primary TOP_CELL \
 -netlist {-file ./design/top_cell.spi -format SPICE -case YES } \
 -report {-file output/3dstack.report} \
 -export_connectivity {-file output/3dstack.v -format verilog }
```

## Die and Layer Definitions

The next section defines each die used in the stack. The die definitions must include paths to layout files and any layers used for connectivity in the stack. The first die definition is the interposer, which includes a number of metal layers used to connect the dies. In order for Calibre 3DSTACK to understand how these layers interact, you must define the layer-to-layer interactions with the `die -wb_connect` argument. This is referred to as white-box connectivity because you are defining layer connectivity within a die and not just the external interfaces. To indicate that a layer is part of an connectivity stack external to the die (for connectivity tracing between dies), apply the `-layer_info ... -ext_connect` option.

Note, that this section only defines the dies used in the stack. How these dies are physically placed in the assembly is performed with the `stack` command.

```
#####
Define dies in stack
#####

die -die_name interposer \
 -layout { \
 -path ./design/interposer.gds \
 -type gdsii \
 -primary interposer \
 -depth all \
 } \
 -layer_info { \
 -type pad \
 -name interposer_pads \
 -ext_connect \
 -layer { \
 255 \
 -depth all \
 } \
 -text { 255 } \
 -bottom \
 } \
}
```

```
-layer_info { \
 -type RDL_M1 \
 -name RDL_M1 \
 -ext_connect \
 -layer { \
 11 \
 -depth all \
 } \
 -bottom \
} \
-layer_info { \
 -type RDL_V1 \
 -name RDL_V1 \
 -layer { \
 12 \
 -depth all \
 } \
 -bottom \
} \
-layer_info { \
 -type RDL_M2 \
 -name RDL_M2 \
 -layer { \
 13 \
 -depth all \
 } \
 -bottom \
} \
```

```
-layer_info { \
 -type RDL_V2 \
 -name RDL_V2 \
 -layer { \
 14 \
 -depth all \
 } \
 -bottom \
} \
-layer_info { \
 -type RDL_M3 \
 -name RDL_M3 \
 -layer { \
 15 \
 -depth all \
 } \
 -bottom \
} \
-layer_info { \
 -type RDL_V3 \
 -name RDL_V3 \
 -layer { \
 16 \
 -depth all \
 } \
 -bottom \
} \
-layer_info { \
 -type RDL_M4 \
 -name RDL_M4 \
 -layer { \
 17 \
 -depth all \
 } \
 -bottom \
} \
-interposer \
-wb_connect RDL_M1 RDL_M2 BY RDL_V1 \
-wb_connect RDL_M2 RDL_M3 BY RDL_V2 \
-wb_connect RDL_M3 RDL_M4 BY RDL_V3 \
-wb_connect interposer_pads RDL_M1
```

```
die -die_name controller \
 -layout { \
 -path ./design/controller.gds \
 -type gdsii \
 -primary controller \
 -depth top-only \
 } \
 -layer_info { \
 -type bump \
 -name cont_c4 \
 -ext_connect \
 -layer { \
 255 \
 -depth top-only \
 } \
 -text { 255 } \
 -bottom \
 }

die -die_name ram \
 -layout { \
 -path ./design/ram.gds \
 -type gdsii \
 -primary ram \
 -depth top-only \
 } \
 -layer_info { \
 -type bump \
 -name ram_c4 \
 -ext_connect \
 -layer { \
 255 \
 -depth top-only \
 } \
 -text { 255 } \
 -bottom \
 }
```

## Assembly Operations

Up to this point in the rule file, all dies and layers are defined. The next step is to define how these dies are placed (assembled) using the [stack](#) command. In this example, the bottom die is placed first using the [-die](#) argument. The [-placement](#) argument determines the horizontal x and y placement of the interposer. The controller and two ram dies are on the same vertical plane (their initial z-coordinates are the same). This is referred to as a tier. The tier is stacked on top of the interposer, where the initial z-coordinate of the tier is determined implicitly from the thickness and specified [-z\\_origin](#) of the interposer.

[Figure A-6](#) and [Figure A-7](#) show the assembly generated by the statements in this example.

## Examples

### Calibre 3DSTACK+ Extended Syntax Example

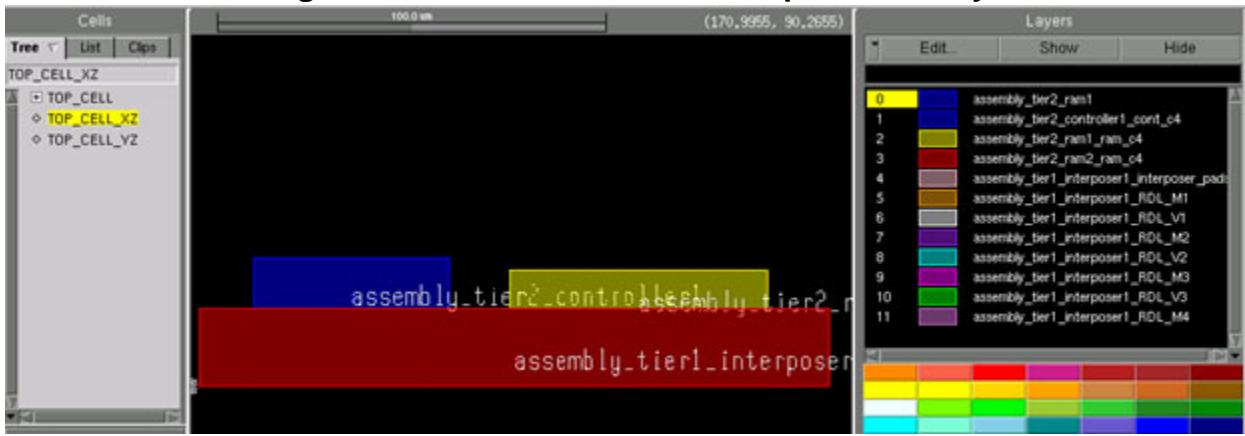
```
#####
Arrange chips in 3DSTACK
#####

stack -stack_name assembly \
-die { \
 -name interposer \
 -source pInterp \
 -placement 0 0 \
 -invert \
} \
-z_origin 0 \
-tier { \
 -die {-name controller -placement 14 12 -source pCont}\}
 -die {-name ram -placement 80 12 -source pRam1}\}
 -die {-name ram -placement 80 45 -source pRam2}\}
}
```

**Figure A-6. Overhead View of the Example Assembly**



**Figure A-7. Side View of the Example Assembly**



## Verification Rules

The verification rules in the 3DSTACK+ extended syntax are layer-based and not specific to a particular design. This enables you to write rules that can be used for any assemblies that use the same design kit layer definitions.

The examples in this section demonstrate just a few of the available verification checks. See “Rule Check Commands” on page 139 for a complete description of the verification commands for the extended syntax file.

```
#####
LVS Checks
#####

connected -check_name CONNECT_IO \
 -layer_type1 pad \
 -layer_type2 bump \
 -black_box \
 -comment "CONNECT::Controller to Interposer check"

#####
DRC Checks
#####

external -check_name PAD_SPACE -layer_type1 pad \
 -constraint "< 65 REGION" -comment "ERR:Placement within 65um!"

external -check_name RDL_M1.1 -layer_type1 RDL_M1 \
 -constraint "< 1 REGION" -comment "ERR: Spacing < 0.1um on RDL_M1!"

external -check_name RDL_M2.1 -layer_type RDL_M2 \
 -constraint "< 0.07 REGION" -comment "ERR: Spacing < 0.5um on RDL_M2!"

overlap -check_name "OVERLAP_PAD_BUMP" \
 -layer_type1 pad \
 -layer_type2 bump \
 -constraint "< 99" -intersection \
 -comment "Pad overlap must be greater than 90%!"

select_checks -check_names {OVERLAP_PAD_BUMP PAD_SPACE CONNECT_IO}
```

## Standard (Legacy) Syntax Example 1

A typical example using the conventional Calibre 3DSTACK rule file syntax.

```
This file includes a 3D-IC Description Language ("3DIC_DL")
used within/by Mentor Graphics' products supporting 2.5D/3D-IC IC
applications. You shall not use this 3DIC_DL # unless you are a Mentor
Graphics customer. The exact terms of your obligations and rights are
governed by your respective license.
You shall not use this 3DIC_DL except:
(a) for your internal business purposes and
(b) for use with Mentor Graphics' Calibre(r) tools.
The 3DIC_DL may constitute or contain trade secrets and confidential
information of Mentor Graphics or its licensors. You shall not make the
3DIC_DL available in any form to any person other than your employees
and on-site contractors, excluding Mentor Graphics competitors, whose
job performance requires access and who are under obligations of
confidentiality.
```

```

set_version -version 1.0

Ensure layouts are LVS/DRC clean by running Calibre

declare layouts

layout -chip_name c1 -primary TOPCELL -path ./chip1.gds -system GDS \
 -original_extent
layout -chip_name c2 -primary TOPCELL -path ./chip2.gds -system GDS \
 -original_extent
layout -chip_name ip -primary TOPCELL -path ./interposer.gds -system GDS

declare layers

layer -layer m11 -chip c1 -layer_number 0
layer -layer m21 -chip c1 -layer_number 1

layer -layer m12 -chip c2 -layer_number 0
layer -layer m22 -chip c2 -layer_number 1

layer -layer m13 -chip ip -layer_number 0
layer -layer m23 -chip ip -layer_number 1

declare additional layers

Note that TVF code inside the tvf_block check is stand-alone and is
outside the scope of the chip stack rule file.

tvf_block additional_layers {
 tvf::SETLAYER m11_m21 = clp_m11 AND clp_m21;
 tvf::SETLAYER m12_m22 = c2p_m12 AND c2p_m22;
} -export_layers [list m11_m21 m12_m22]

export connectivity
export_connectivity -file spice_output.spi -format SPICE

import connectivity
source_netlist -file spice_input.spi -format SPICE

place chip2 and the interposer

place_chip -placement c2p -chip c2 -x_origin 1.85 -y_origin 0
place_chip -placement ipp -chip ip -x_origin 0 -y_origin 0

create a 2x3 array of chip1 placements

for { set x 1 } { $x < 3 } { incr x } {
 for { set y 1 } { $y < 4 } { incr y } {
 place_chip -placement clp_${x}_${y} -chip c1 -x_origin [expr $x+1] \
 -y_origin [expr $y+1] -flip y
 }
}

```

```
dimensional checks

#results highlighted in Calibre RVE only show layers used in checks:
set_auto_rve_show_layers YES

enclosure -check_name enc_check -placement1 ipp_m13 \
 -placement2 c2p_m12 -constraint "<= 0.03 REGION" \
 -comment "enclosure check" -set_rve_highlight_color blue
external -check_name ext_check -placement1 ipp_m13 \
 -constraint "<= 0.05 REGION" \
 -comment "external check" -set_rve_highlight_color yellow
internal -check_name int_check -placement1 ipp_m13 \
 -constraint "<= 0.17 REGION" \
 -comment "internal check"

connectivity check

attach_text -placement c2p_m12 -text_placement c2p_m12
attach_text -placement ipp_m13 -text_placement ipp_m13

connected -check_name con_check -placement1 c2p_m12 \
 -placement2 ipp_m13 -comment "connectivity check" \
 -set_rve_priority 1

output report
report -file report.txt
```

## Standard (Legacy) Syntax Example 2

Simplified method for creating regular assemblies and connectivity checks using the conventional Calibre 3DSTACK rule file syntax.

```
This file includes a 3D-IC Description Language ("3DIC_DL")
used within/by Mentor Graphics' products supporting 2.5D/3D-IC IC
applications. You shall not use this 3DIC_DL # unless you are a Mentor
Graphics customer. The exact terms of your obligations and rights are
governed by your respective license.
You shall not use this 3DIC_DL except:
(a) for your internal business purposes and
(b) for use with Mentor Graphics' Calibre(r) tools.
The 3DIC_DL may constitute or contain trade secrets and confidential
information of Mentor Graphics or its licensors. You shall not make the
3DIC_DL available in any form to any person other than your employees
and on-site contractors, excluding Mentor Graphics competitors, whose
job performance requires access and who are under obligations of
confidentiality.
```

```
set_version -version 1.0

layout_primary TOPCELL_3DIC

stack three chips together (chip1.gds through chip3.gds)

for { set i 1 } { $i <= 3 } { incr i } {
 layout -chip_name c$i -primary TOPCELL -path ./chip$i\.gds \
 -system GDS
 layer -layer m$i -chip c$i -layer_number 0
 place_chip -placement c$i\p -chip c$i -x_origin [expr $i-1] \
 -y_origin 0
 attach_text -placement c$i\p_m$i -text_placement c$i\p_m$i
}

perform verification checks

for { set i 1 } { $i <= 2 } { incr i } {

connect c$i\p_m$i c[expr $i+1]p_m[expr $i+1]
connected -check_name con_check$i -placement1 c$i\p_m$i \
 -placement2 c[expr $i+1]p_m[expr $i+1]
internal -check_name int_check$i -placement1 c$i\p_m$i \
 -placement2 c[expr $i+1]p_m[expr $i+1] -constraint "<= 0.5" \
 -comment "internal check$i\", c$i\p_m$i to c[expr $i+1]p_m[expr $i+1]" \
 -set_rve_show_layers AUTO
}

output report

report -file report.txt
```

# Report File Format

The Calibre 3DSTACK report file is generated using the report command.

The sections of the report appear in the order that they are generated.

|                                                                   |            |
|-------------------------------------------------------------------|------------|
| <b>Report Header</b> .....                                        | <b>349</b> |
| <b>Summary of Drawn Layers, Placements, and Text Layers</b> ..... | <b>349</b> |
| <b>Verification Results</b> .....                                 | <b>352</b> |

## Report Header

The report begins with a summary of the input, output, user, and design information.

```
#####
C A L I B R E S Y S T E M
3 D S T A C K R E P O R T
#####
~ ~ ##
x x ## @
@ ##
/ \ ##
#####
REPORT FILE NAME: /home/userDir/3dstack_report.rpt
MAXIMUM RESULTS: 50
LAYOUT CHIP NAME: controller - ./designs/controller.gds ('controller')
LAYOUT CHIP NAME: myram - ./designs/fullchip_clean_myram.gds ('myram')
LAYOUT CHIP NAME: interposer - ./designs/interposer.gds ('interposer')
SOURCE NETLIST: /home/userDir/designs/system_netlist.spi ('TOP_3D')
3DSTACK ASSEMBLY: /home/userDir/3dstack_assembly.oas ('TOP_3D')
RULE FILE: /home/userDir/3dstack.rules
CREATION TIME: 07/17/13 16:28:59
CURRENT DIRECTORY: /home/userDir
USERNAME: user
CALIBRE VERSION: Calibre v2020.2_x.xxxx Tue Jun 16 13:25:32 PDT 2020
RENAMING: NO
```

## Summary of Drawn Layers, Placements, and Text Layers

The report summarizes the layers, placements, and text that were specified in your rule file.

RENAMING RULES:

LAYERS:

```
Layer Name: INTERP_FRONT_if
 Chip: interposer
 Layer Number: 100
Layer Name: bmet1
 Chip: interposer
 Layer Number: 51
Layer Name: bmet2
 Chip: interposer
 Layer Number: 53
Layer Name: tsv
 Chip: interposer
 Layer Number: 50
Layer Name: INTERP_BACK_if
 Chip: interposer
 Layer Number: 200
Layer Name: CONTROLLER_if
 Chip: controller
 Layer Number: 100
Layer Name: via_rdl1
 Chip: interposer
 Layer Number: 20
Layer Name: bvial
 Chip: interposer
 Layer Number: 52
Layer Name: metal_rdl1
 Chip: interposer
 Layer Number: 19
Layer Name: RAM_if
 Chip: myram
 Layer Number: 100
Layer Name: metal_rdl2
 Chip: interposer
 Layer Number: 21
```

## Report of Chip and Layer Placements in the Stack

All layers and placements are written to the report. This is useful for verifying the assembly.

```
ANCHOR PLACEMENT(S) :
PLACEMENT(S) :
 Chip Placement: pCont
 Layout: controller
 x-origin: 0.000
 y-origin: 0.000
 Magnification: 1.0
 Rotation: 0
 Flip Axis: y
 Chip Placement: pRAM_stack3
 Layout: myram
 x-origin: -780.000
 y-origin: 520.000
 Magnification: 1.0
 Rotation: 180
 Flip Axis: y
 Chip Placement: pRAM_stack4
 Layout: myram
 x-origin: -740.000
 y-origin: 80.000
 Magnification: 1.0
 Rotation: 270
 Flip Axis: y
 Chip Placement: pRAM_stack5
 Layout: myram
 x-origin: -740.000
 y-origin: 80.000
 Magnification: 1.0
 Rotation: 270
 Flip Axis: y
 Chip Placement: pRAM_stack1
 Layout: myram
 x-origin: 280.000
 y-origin: 65.000
 Magnification: 1.0
 Rotation: 0
 Flip Axis: y
 Chip Placement: pInterp
 Layout: interposer
 x-origin: 0.000
 y-origin: 0.000
 Magnification: 1.0
 Rotation: 0.0
 Flip Axis: NONE
 Chip Placement: pRAM_stack2
 Layout: myram
 x-origin: 230.000
 y-origin: 505.000
 Magnification: 1.0
 Rotation: 90
 Flip Axis: y
```

## Details on Text Layers Defined in the Stack

Each defined layer is listed separately.

```
TEXT LAYER(S) :
Layer: pRAM_stack2_RAM_if (pRAM_stack2_RAM_if)
Layer: pCont_CONTROLLER_if (pCont_CONTROLLER_if)
Layer: pRAM_stack3_RAM_if (pRAM_stack3_RAM_if)
Layer: pRAM_stack4_RAM_if (pRAM_stack4_RAM_if)
Layer: pRAM_stack1_RAM_if (pRAM_stack1_RAM_if)
```

## Net Mapping

Net mapping operations from the net\_map or config -net\_map commands are summarized in this section:

```
NET MAPPING:
 FROM: TO:

 PWR VDD
```

# Verification Results

Verification check results are reported under the RULECHECK SUMMARY section.

```
OVERALL VERIFICATION RESULTS

 RULECHECK SUMMARY

Status Result Count Rule

COMPLETED 0 Floating_Text (Selecting text labels from
pCont_CONTROLLER_if not having overlap with any of the pads. Selecting
text labels from pRAM_stack1_RAM_if not having overlap with any of the
pads (the same as for pRAM_stack2_RAM_if, pRAM_stack3_RAM_if,
pRAM_stack4_RAM_if).)
COMPLETED 5 No_Text (Selecting pads from pCont_CONTROLLER_if
not having text-labels attached. Selecting pads from pRAM_stack1_RAM_if
not having text-labels attached (the same as for pRAM_stack2_RAM_if,
pRAM_stack3_RAM_if, pRAM_stack4_RAM_if).)
COMPLETED 0 Multi_Text (Shapes from placement
pCont_CONTROLLER_if overlap multiple text labels from text placement
pCont_CONTROLLER_if. Shapes from placement pRAM_stack1_RAM_if overlap
multiple text labels from text placement pRAM_stack1_RAM_if (the same as
for pRAM_stack2_RAM_if, pRAM_stack3_RAM_if, pRAM_stack4_RAM_if).)
COMPLETED 0 ExtraPorts
```

## Physical Verification Results

```
COMPLETED 1 PLACEMENT_check
(RAM should be more than 70 um from controller.)
COMPLETED 50 (of 56) offgrid1
COMPLETED 1 centers_interposer
(Pad centers must be within exactly 80 um)
```

## Connected Check Verification Results

```
COMPLETED 2 CONNECT_RAM2_to_CONTROLLER
(CONNECTION CHECK between LAYER:pRAM_stack2_RAM_if and
LAYER:pCont_CONTROLLER_if)
COMPLETED 0 CONNECT_RAM3_to_CONTROLLER
(CONNECTION CHECK between LAYER:pRAM_stack3_RAM_if and
LAYER:pCont_CONTROLLER_if)
COMPLETED 0 CONNECT_RAM4_to_CONTROLLER
(CONNECTION CHECK between LAYER:pRAM_stack4_RAM_if and
LAYER:pCont_CONTROLLER_if)
COMPLETED 0 CONNECT_RAM1_to_CONTROLLER
(CONNECTION CHECK between LAYER:pRAM_stack1_RAM_if and
LAYER:pCont_CONTROLLER_if)
```

## Missing Placements in Source and Layout

```

MISSING PLACEMENTS

LAYOUT NAME SOURCE NAME
NONE
```

## Missing Ports in the Layout

```

SOURCE PORTS MISSING LAYOUT PORTS

RuleCheck: CONNECT_RAM1_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack1_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
RuleCheck: CONNECT_RAM2_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack2_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
RuleCheck: CONNECT_RAM3_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack3_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
RuleCheck: CONNECT_RAM4_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack4_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
```

## Missing Ports in the Source

```

 LAYOUT PADS MISSING SOURCE PORTS

 RuleCheck: CONNECT_RAM1_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack1_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
 RuleCheck: CONNECT_RAM2_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack2_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
 RuleCheck: CONNECT_RAM3_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack3_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
 RuleCheck: CONNECT_RAM4_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack4_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
```

## Net Differences Between Source and Layout

```

 INCORRECT NETS

 LAYOUT NAME SOURCE NAME
 RuleCheck: CONNECT_RAM1_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack1_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
 RuleCheck: CONNECT_RAM2_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack2_RAM_if and LAYER:pCont_CONTROLLER_if)

Net 105
 ** missing connection **
Net 61
 ** missing connection **
pRAM_stack2:ADR<0>
 RuleCheck: CONNECT_RAM3_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack3_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
 RuleCheck: CONNECT_RAM4_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack4_RAM_if and LAYER:pCont_CONTROLLER_if)

NONE
```

## Connectivity Results

```

 CONNECTIVITY RULECHECK RESULTS

 RuleCheck: CONNECT_RAM2_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack2_RAM_if and LAYER:pCont_CONTROLLER_if)

1
 Port: pCont:adr<0>
 Net: 61
 SourceNet: ADR<0>
 LNC: pCont:adr<0>
 SNC: pCont:ADR<0> pRAM_stack2:ADR<0>
2
 Port: pRAM_stack2:adr<0>
 Net: 105
 SourceNet: ADR<0>
 LNC: pRAM_stack2:adr<0>
 SNC: pCont:ADR<0> pRAM_stack2:ADR<0>
 RuleCheck: CONNECT_RAM3_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack3_RAM_if and LAYER:pCont_CONTROLLER_if)

 RuleCheck: CONNECT_RAM4_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack4_RAM_if and LAYER:pCont_CONTROLLER_if)

 RuleCheck: CONNECT_RAM1_to_CONTROLLER (CONNECTION CHECK between
LAYER:pRAM_stack1_RAM_if and LAYER:pCont_CONTROLLER_if)

```



# Appendix B

## Standard Calibre 3DSTACK Syntax Commands

---

The standard (or traditional) Calibre 3DSTACK rule file syntax supports a different set of assembly and system configuration commands than the extended 3DSTACK+ syntax.

The extended 3DSTACK+ syntax (see “[Command Reference](#)” on page 65) is recommended. Rule check commands are shared between the two syntaxes, but the standard syntax is less flexible because the rules must map to placement names used in the assembly and configuration commands.

The Calibre 3DSTACK tool standard syntax rule compiles a standard syntax rule file from your extended syntax rule input and writes it to your working directory during a run. This file can be used to debug your assembly and verification operations.

This section documents commands exclusive to the standard Calibre 3DSTACK syntax.

|                                                |            |
|------------------------------------------------|------------|
| <b>System and Miscellaneous Commands .....</b> | <b>358</b> |
| <b>Assembly Commands .....</b>                 | <b>388</b> |
| <b>Standard Rule Syntax .....</b>              | <b>425</b> |
| <b>set_auto_rve_show_layers.....</b>           | <b>432</b> |
| <b>set_rve_cto_file.....</b>                   | <b>433</b> |
| <b>tvf_block .....</b>                         | <b>435</b> |

# System and Miscellaneous Commands

System and miscellaneous commands specify the verification inputs, outputs, and run control options.

**Table B-1. System Control and Miscellaneous Commands**

| Name                                            | Description                                                                                                                                                      |
|-------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">assembly_path_extension</a>         | Sets the severity of circular GDS path extension errors.                                                                                                         |
| <a href="#">export_connectivity</a>             | Exports the connectivity information into a netlist or annotated GDS file.                                                                                       |
| <a href="#">export_layout (Standard Syntax)</a> | Generates an annotated GDS of a specified portion of the assembly, including the layers of the specified placements.                                             |
| <a href="#">export_template</a>                 | Exports a specified chip or placement from the stack into a GDS or OASIS file.                                                                                   |
| <a href="#">layer_props_file</a>                | Specifies a layer properties file for Calibre DESIGNrev that sets custom layer visibility options.                                                               |
| <a href="#">report</a>                          | Generates a report file that contains information about the verification run. This is useful for debugging the design.                                           |
| <a href="#">run</a>                             | Runs Calibre using the specified rule file and command-line options.                                                                                             |
| <a href="#">set_version</a>                     | Sets the syntax version of the Calibre 3DSTACK rule file.                                                                                                        |
| <a href="#">source_filter</a>                   | Applies filtering options to the imported source netlist file.                                                                                                   |
| <a href="#">source_netlist</a>                  | Imports a source netlist file used for connectivity comparison. It is highly recommended that you use a source netlist during the verification of your assembly. |
| <a href="#">svrf_specs</a>                      | Includes a file containing SVRF statements in the run.                                                                                                           |
| <a href="#">warning_severity</a>                | Specifies the severity level for warning messages.                                                                                                               |

## **assembly\_path\_extension**

Sets the severity of circular GDS path extension errors.

### **Usage**

**assembly\_path\_extension *value***

### **Arguments**

- ***value***

Required integer between 0 and 8. See PATH\_CIRCULAR in the “Exceptions for Layout Input Exception Severity” table of the *SVRF Manual*.

### **Examples**

```
assembly_path_extension 8
```

## export\_connectivity

Exports the connectivity information into a netlist or annotated GDS file.

### Usage

```
export_connectivity -file file_name
 [-format {VERILOG | AIF | MGC | SPICE | XSI }]
 [{-property number} | -text] [-flat]
 [-hier [-no_top]]
 [-pkg package_name]
 [-pex_map {calibrated_layers_list assembly_layers_list}]
```

### Arguments

- **-file *file\_name***

Required argument and value set that specifies the name of the output file.

- **-format {VERILOG | AIF | MGC | SPICE | XSI | GDS}**

Optional argument set that specifies the format for the output connectivity file (the default is Verilog).

- **-hier**

Optional argument that specifies to generate a hierarchical netlist for 2.5D ICs. This option only works with -format SPICE. You must also specify **layout** -interposer for one of the imported chips.

When the -interposer argument is applied to a layout and the -hier option is used, the generated netlist instantiates all dies within the layout instance specified by -interposer. The top cell of the assembly only contains the instantiated interposer die. The net names are generated from the interposer pin names.

- **-no\_top**

Optional argument that specifies not to create a new top cell for the entire design. This option requires -hier to be specified.

- **-pkg *package\_name***

Optional argument that specifies a package layout in the design. The layout must not be an interposer. The specified *package\_name* must have at least one layer defined in the connectivity stack. This option requires -hier to be specified.

### Description

Exports connectivity to a file in the specified format (Verilog, AIF, SPICE, MGC, or XSI).

The MGC format contains a list of add\_connection directives that describe single connections in the 3D assembly:

```
add_connection -connection1 {placement1 cell1 net1 pin_instance_name1} \
 -connection2 {placement2 cell2 net2 pin_instance_name2}
```

Warnings are issued for single-port connections (nets connected to one port only) in the extracted netlist.

### **Caution**

- ❑ There should not be spaces in cell or pin names as this causes the exported netlist to be incorrect. For pins, the tool creates multiple pins in such a case. The `export_connectivity` command issues a warning for these cases.

### AIF Format

See “[AIF Export File Format](#)” on page 280 for details on the AIF netlist exported by Calibre 3DSTACK.

### XSI Format

Calibre 3DSTACK generates the XSI CSV file using the pins from all placements in the following columns:

Signal Name, Instance Level Name, Design Name, Component Name, Pin Number, Ref Des

The following is an example.

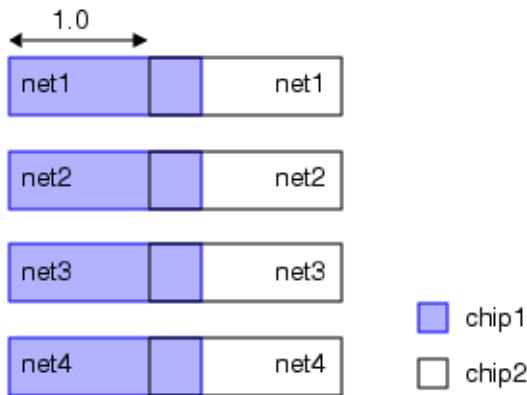
```
685,685,TOPCELL_3DI,assembly1_tier1_BGA1,BS_SW1_GPIO_40,BGA
```

### Examples

#### Example 1

Consider the chip stack shown in [Figure B-1](#). In this chip stack, two layouts (chip1 and chip2) are arranged such that they overlap. Polygons on each layout contain text objects labeled net1 through net4.

**Figure B-1. Connectivity Export Example**



The chip stack rule file for this example is shown as follows:

```
set_version -version 1.0

#define layouts
layout -chip_name c1 -primary TOPCELL -path ./chip1.gds -system GDS
layout -chip_name c2 -primary TOPCELL -path ./chip2.gds -system GDS

#write SPICE, Verilog, and MGC netlists for the design
export_connectivity -file spice.out -format SPICE
export_connectivity -file verilog.out -format VERILOG
export_connectivity -file mgc.out -format MGC

#define layers and place chips
layer -layer m1 -chip c1 -layer_number 0
layer -layer m2 -chip c2 -layer_number 0

place_chip -placement c1p -chip c1 -x_origin 0 -y_origin 0
place_chip -placement c2p -chip c2 -x_origin 1 -y_origin 0

#define connectivity
connect c1p_m1 c2p_m2

connected -check_name con_check -placement1 c1p_m1 \
-text_placement1 c1p_m1 -placement2 c2p_m2 -text_placement2 c2p_m2
```

The `export_connectivity` commands in this file generate three output files. These files, which are written in the SPICE, Verilog, and MGC formats, contain the connectivity information for the chip stack. The contents of these files are shown as follows:

- SPICE

```
.SUBCKT TOPCELL_3DIC
Xc1p 4 3 2 1 TOPCELL
Xc2p 4 3 2 1 TOPCELL
.ENDS TOPCELL_3DIC

.SUBCKT TOPCELL net1 net2 net3 net4
.ENDS TOPCELL
```

- VERILOG

```

module TOPCELL_3DIC (
) ;
wire 4 ;
wire 1 ;
wire 2 ;
wire 3 ;
TOPCELL c1p (
 .net1 (4) ,
 .net2 (3) ,
 .net3 (2) ,
 .net4 (1)) ;
TOPCELL c2p (
 .net1 (4) ,
 .net2 (3) ,
 .net3 (2) ,
 .net4 (1)) ;
endmodule

```

```

module TOPCELL (
 net1 ,
 net2 ,
 net3 ,
 net4) ;
inout net1 ;
inout net2 ;
inout net3 ;
inout net4 ;
endmodule

```

- MGC

```

add_connection -connection1 {c1p TOPCELL net1 net1} -connection2\
{c2p TOPCELL net1 net1}
add_connection -connection1 {c1p TOPCELL net2 net2} -connection2\
{c2p TOPCELL net2 net2}
add_connection -connection1 {c1p TOPCELL net3 net3} -connection2\
{c2p TOPCELL net3 net3}
add_connection -connection1 {c1p TOPCELL net4 net4} -connection2\
{c2p TOPCELL net4 net4}

```

### Example 2

This example demonstrates the `export_connectivity -hier` option. Assume you have the following chips in your assembly:

```

layout -chip_name interposer -primary interposer -path interposer.gds \
-system GDS -original_extent -interposer

layout -chip_name controller -primary controller -path controller.gds \
-system GDS -original_extent

layout -chip_name ram -primary ram -path ram.gds -system GDS \
-original_extent

```

Note that the -interposer option was applied to the interposer die. The top cell of the assembly is called TOP\_CELL:

```
layout_primary TOP_CELL
```

After assembling your 2.5D IC, you export the two SPICE netlists; one with -hier option:

```
export_connectivity -file output/3dstack_hier.sp -format SPICE -hier
export_connectivity -file output/3dstack_no_hier.sp -format SPICE
```

The two netlists are shown side-by-side for comparison:

**Table B-2. export\_connectivity -hier**

| 3dstack_hier.sp                                                                                                                                                                                                                                                           | 3dstack_no_hier.sp                                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>.SUBCKT TOP_CELL <b>XpInterposer</b> ... interposer .ENDS TOP_CELL  .SUBCKT controller ... .ENDS controller  .SUBCKT ram ... .ENDS ram  .SUBCKT interposer ... <b>XpController</b> ... controller <b>XpRam0</b> ... ram <b>XpRam1</b> ... ram .ENDS interposer</pre> | <pre>.SUBCKT TOP_CELL <b>XpInterposer</b> ... interposer <b>XpController</b> ... controller <b>XpRam0</b> ... ram <b>XpRam1</b> ... ram .ENDS TOP_CELL  .SUBCKT controller ... .ENDS controller  .SUBCKT ram ... .ENDS ram  .SUBCKT interposer ... .ENDS interposer</pre> |

Without -hier, all chips in the assembly are instantiated in the TOP\_CELL. With -hier, the chips that are not interposers are instantiated within the interposer.

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

# export\_layout (Standard Syntax)

Generates an annotated GDS of a specified portion of the assembly, including the layers of the specified placements.

## Usage

```
export_layout -file output_file [-output_dir directory]
 [-placement '{' -name placement_name [-pex_map calibrated_layer_name]
 [-rc_model] [-bottom_rc_interface] [-top_rc_interface] [-invert] [-ircx file]
 [-mipt file] '}' ... [-rename_text "expression ..."]
 [-second_ground '{'
 -name layer_name
 {-vertices {x1 y1 x2 y2} | -extent }
 -text "ground_name"
 [-pex_map layer_name] '}']]
 [-cci [-ground_name] [-power_name]] [-enable_rc_deck | -enable_rc_map]
 [-output_dir directory]
 [-netlist_format {[SPICE] [XSI] [VERILOG] [MGC] [AIF]}}
 [-property property_number] [-flat] [-text] [-rename_text "expression"]
 [-use_lvs_names]
```

## Arguments

- **-file *output\_file***

Required argument set that specifies the path to the generated layout file. If -output\_dir is not specified, the output is saved to a DFM database in the following location:  
*3dstack.dfmdb/pex/<output\_file>*.

- **-output\_dir *directory***

Optional argument set that specifies a directory in which to write all output files from the export\_layout command. If you do not specify this option, the tool generates all output from the export\_layout command to *3dstack.dfmdb/pex* in your working directory.

- **-placement {*arguments*}**

Optional argument that specifies to export one or more interacting placements.

---

**Note**

 The arguments inside the -placement option must be enclosed in brackets {}.

---

- **-name *placement\_name***

Optional argument that specifies the name of the placement to export.

- **-pex\_map *calibrated\_layer\_name***

Optional argument set that maps calibrated layer names to the physical layers in the assembly. This argument set only applies for -format GDS.

Calibre 3DSTACK uses this information to write an SVRF file when exporting an annotated GDS file. The SVRF file is named `<gds_name>.map.svrf`, where `gds_name` is the name you specified for the annotated GDS. The calibrated layer names and assembly layer pairings are written to the generated SVRF file with [PEX Map](#) statements.

- **-rc\_model**  
Optional argument that writes TSV PEX statements to the extraction rule file. See “[export\\_layout](#)” on page 131 for details.
- **-bottom\_rc\_interface**  
Optional argument that indicates that the specified layer represents the interface layer on the bottom of the die. You can only apply this argument to a single layer in the argument set. If you apply this argument, you must also specify the **-top\_rc\_interface** argument for one of the other layers in the die. This option is used to identify the interface layers for a virtual die between dies or from dies to interposers for extraction rule generation.
- **-top\_rc\_interface**  
Optional argument that indicates that the specified layer represents the interface layer on the top of the die. You can only apply this argument to a single layer in the argument set. If you apply this argument, you must also specify the **-bottom\_rc\_interface** argument for one of the other layers in the die. This option is used to identify the interface layers for a virtual die between dies or from dies to interposers for extraction rule generation.
- **-ircx *file***  
Optional argument set that specifies the path to a file that contains parasitic information in iRCX format. Use this option with the **-enable\_rc\_deck** option to automate RC extraction rule generation. This option is mutually exclusive with the **-mipt** option.
- **-mipt *file***  
Optional argument set that specifies the path to a file that contains parasitic information in MIPT format. Use this option with the **-enable\_rc\_deck** option to automate RC extraction rule generation. This option is mutually exclusive with the **-ircx** option.
- **-invert**  
Optional argument that specifies that the layer is inverted.
- **-rename\_text “*expression* ...”**  
Optional argument set that renames text in the placement. The expression is a GNU regular expression. Multiple expressions are supported, but the entire set of *expressions* must be enclosed in quotes or braces. For example:  

```
-rename_text "/</\|/\|>/\|/"
```
- **-second\_ground ‘{’*arguments* ‘}’**  
Optional argument set that enables you to create an additional ground layer in the generated annotated GDS, create a rectangular shape on the layer, and attach a text label. The tool also

writes the information to the *.map.svrf* file. To use this option, you must specify the following *arguments*:

- -name *layer\_name* — Specifies the name of the layer on which the second ground shape is generated.
- -vertices *x1 y1 x2 y2* — Specifies a set of coordinates for the ground pin shape. This option is mutually exclusive with -extent.
- -extent — Specifies that the extent of the die should be used as the shape for the second ground. This option is mutually exclusive with -vertices.
- -text “*ground\_name*” — Specifies the text label for the second ground.
- -pex\_map *layer\_name* — Specifies the calibrated layer name for the second ground.
- -cci [-ground\_name] [-power\_name]

Optional argument that generates a set of standard files that can be used in third-party extraction flows. This option cannot be specified with the -text, -flat, -enable\_rc\_deck, or -netlist\_format options.

**-ground\_name *name***

Specifies a ground name used for ground nets in the generated Layer Net Specs (LNS) file.

**-power\_name *name***

Specifies a power name used for power nets in the generated LNS file.

When you specify the -cci argument set, the tool creates a Layer Net Specs file in your working directory. The LNS file is an SVRF file that contains the following sections:

- Layer definitions.
- Connectivity statements.
- Power and ground name statements, if specified using the -power\_name and -ground\_name options.

The tool generates a flat spice netlist that contains a top subcircuit without any instances and with sorted net numbers as pins.

If you set the CALIBRE\_3DSTACK\_ENABLE\_HIER\_CCI environment variable to a non-null value, the following items include placement information:

- lnn (Layout Netlist Names)
- ixf (Instance Xref File)
- nxf (Net Xref File)
- lnx (Layout Net Xref File)

- **-enable\_rc\_deck**

Optional argument that specifies to generate an extraction rule file for interface layers between two specified dies. You can specify the export\_layout command with this option for each of the interacting dies. This option cannot be specified with -enable\_rc\_map.

- **-enable\_rc\_map**

Optional argument that specifies to only generate the xcalibrate\_map file. This argument only applies to -enable\_rc\_deck and cannot be specified with -layout\_only. If the MIPT files were not specified, the tool creates placeholders in the map file with the following format:

```
<DIE_NAME>_mipt
```

This option cannot be specified with -enable\_rc\_deck.

- **-netlist\_format {[SPICE] [XSI] [VERILOG] [MGC] [AIF]}**

Optional argument set that specifies the netlist format to output. You can specify a single format or a space-separated list containing multiple formats.

- **-property *property\_number***

Optional argument that specifies the property number to which to write connectivity information in the annotated GDS file. The default property number is 1. The *number* argument must an integer between 0 and 65535. This option only applies to -format GDS and is mutually exclusive with -text.

- **-flat**

Optional argument that specifies to export a flat layout. If this is not specified, then the output is hierarchical.

- **-text**

Optional argument that specifies to include text in the generated layout file. If this argument is specified, the -flat argument is also applied automatically.

- **-rename\_text "expression ..."**

Optional argument set that renames text in the exported layout. The expression is a GNU regular expression. Multiple expressions are supported, but the entire set of *expressions* must be enclosed in quotes or braces. For example:

```
-rename_text "/</\[/ />/\]/"
```

- **-use\_lvs\_names**

Optional argument that specifies to use original layer names from the layer placement options in the generated LVS rules file. If this argument is not specified, the tool uses the layer names from the generated Calibre 3DSTACK assembly. This argument cannot be specified with -cci.

## **export\_template**

Exports a specified chip or placement from the stack into a GDS or OASIS file.

### **Usage**

```
export_template {-chip chip_name} | {-placement placement_name}
 {[-neighbor layer [-bump value]] ... } [-clip distance] } -file output_file
 [-system {GDS | OASIS}]
```

### **Arguments**

- **-chip** *chip\_name*  
Argument and value set that specifies the chip name to export (chip names are defined using the [layout](#) command). You must specify either this argument or the **-placement** argument. The layer properties and layer map files are generated when **export\_template -chip** is specified in your rule file.
- **-placement** *placement\_name*  
Argument and value set that specifies an existing placement in the assembly to export. You must specify either this argument or the **-chip** argument. The layer properties and layer map files are generated when **export\_template -placement** is specified in your rule file.
- **-neighbor** *layer* ...  
Optional argument set that specifies a layer in the assembly to export with the specified placement. Multiple layers require a separate **-neighbor** *layer* argument pair for each layer. This option can only be used with the **-placement** argument.
- **-bump** *value*  
Optional argument set that increments the layer number of the *layer* by an integer *value*. This argument can be specified for each **-neighbor** *layer* pair.
- **-clip** *distance*  
Optional argument set that specifies a clipping distance in microns around the chip extent to exclude from the exported layout. The **-clip** argument accepts positive floating-point numbers and can only be specified once for each **export\_template** command.
- **-file** *output\_file*  
Required argument and value set that specifies the name of the output file. Each application of the **export\_template** command requires a unique **output\_file** name.
- **-system** {GDS | OASIS}  
Optional argument that specifies the layout system. The default is GDS.

### **Description**

Exports the specified chip or placement from the 3D assembly to a GDS or OASIS file. Layers are written (with their original layer numbers) only if they are declared in [layer](#) commands for the specified **chip\_name**. The output file is written using the precision of the input files. Empty

layers are supported. Layermap and layer properties files for Calibre DESIGNrev are exported automatically using the same naming convention.

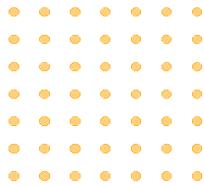
## Examples

### Example

Export a GDS for the controller chip in the assembly.

```
layout -chip_name controller -primary controller \
 -path ${dsn_dir}/controller.gds -system GDS -original_extent
...
export_template -chip controller -file controller_export.gds -system GDS
```

The exported layout contains the layout of the controller chip as shown in the following figure. In this example, the original controller only contains pad shapes.



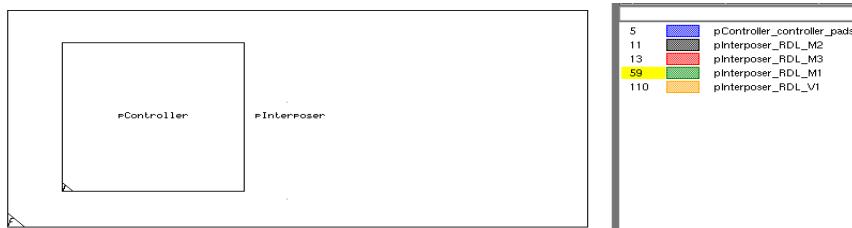
### Example

Export a GDS for the placement of the controller chip in the assembly. This example also specifies to include some of the surrounding interconnect layers on the interposer. The -bump argument for each included layer increments the layer number by the specified integer value.

```
layout -chip_name controller -primary controller \
 -path ${dsn_dir}/controller.gds -system GDS -original_extent

place_chip -placement pController -chip controller \
 -x_origin $controller_x -y_origin $controller_y

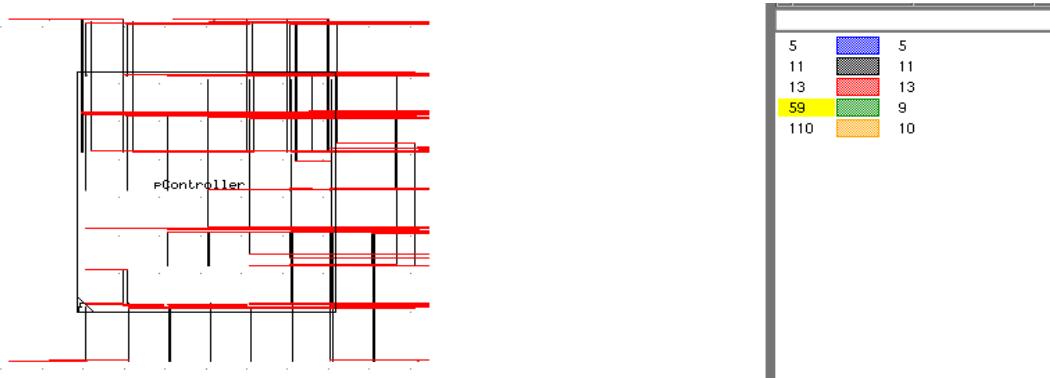
export_template -file pController.gds -system GDS -placement pController \
 -neighbor pInterposer_RDL_M1 -bump 50 -neighbor pInterposer_RDL_V1 \
 -bump 100 -neighbor pInterposer_RDL_M2 -neighbor pInterposer_RDL_M3
```



### Example

Export an OASIS layout for the placement of the controller chip in the assembly. This is the same as the previous example, except that the included layers are clipped at 15 um from the edge of the controller placement.

```
export_template -file pController_clipped.oas -system OASIS \
 -placement pController -neighbor pInterposer_RDL_M1 -bump 50 \
 -neighbor pInterposer_RDL_V1 -bump 100 -neighbor pInterposer_RDL_M2 \
 -neighbor pInterposer_RDL_M3 -clip 15
```



### Related Topics

- [System and Miscellaneous Commands](#)
- [Assembly Commands](#)
- [Rule Check Commands](#)

## layer\_props\_file

Specifies a layer properties file for Calibre DESIGNrev that sets custom layer visibility options.

### Usage

**layer\_props\_file** *props\_file*

### Arguments

- *props\_file*

Required path to a layer properties file. See the “Description” section for details.

### Description

The specified *props\_file* defines the properties of the layers used in Calibre DESIGNrev and follows this format:

```
layer_name layer_color layer_fill layer_visibility layer_width
```

where:

- *layer\_name* — is an alphanumeric string specifying the name of the layer.
- *layer\_color* — is any valid Tcl color name.
- *layer\_fill* — is one of the following fill types:

|         |               |            |
|---------|---------------|------------|
| clear   | diagonal_1    | diagonal_2 |
| wave    | brick         | circles    |
| speckle | light_speckle | solid      |

- *layer\_visibility* — Specifies whether the layer is visible. The *layer\_visibility* must be one of the following:
  - 0 — the layer is not visible.
  - 1 — the layer is visible.
- *layer\_width* — an integer greater than 1 that specifies the line width of the polygon outline on that layer.

### Examples

```
layer_props_file assembly.layerprops
```

where *assembly.layerprops* is written as follows:

```
controller1 blue speckle 1 1
ram_1 yellow diagonal_1 1 1
ram_2 red wave 1 1
...
```

## report

Generates a report file that contains information about the verification run. This is useful for debugging the design.

### Usage

```
report -file report_file [-max_results value] [-child_rdb {NO | YES}]
[-report_ignored_pins {NO | YES}]
```

### Arguments

- **-file *report\_file***

Required argument set that specifies the name of the output file.

- **-max\_results *value***

Optional argument set that limits the number of reported results (per check) to the number specified with the *value* keyword. The max\_results argument applies to each check in the rule file. The *value* keyword must be a positive integer, unless “all” is specified. If *value* is set to “all”, no limitation is applied to the report command. The default is 50.

---

#### Note

 If you do not specify the report command in your rule file, then there is no limit to the number of results written to the results database.

---

- **-child\_rdb {NO | YES}]**

Optional argument set that specifies whether to generate results databases for each chip in the stack. Specify yes to generate child RDBs for all chips in the design. The default is NO.

- **-report\_ignored\_pins {NO | YES}**

Optional argument set that specifies to list all ignored pins in the report. Ignored pins are specified in the [ignore\\_pin](#) command. The default is not to report any ignored pins. The following is an example of an ignored pin in the report:

```
IGNORE PIN STATEMENTS:
 Chip Placement: assembly_tier1_interposer1
 Expression: {*clk*}
 Matched Pins:
 controller_clk
 clk
```

### Description

Generates a file containing a report of the run. The format of the report is similar to the reports for Calibre nmLVS and Calibre PERC runs. Refer to [Report File Format](#) for an example.

The following sections are only generated when a [source\\_netlist](#) command is present in the chip stack rule file:

- Layout pads missing source ports.

- Source ports missing layout pads.
- Incorrect nets.
- Connectivity rulecheck results.

Errors are reported only for [connected](#) checks. Errors for other verification commands ([internal](#), [external](#), and [copy](#)) are not included in the report.

## Examples

Use the following command to generate a report file with no reported result limitations:

```
report -file report.txt -max_results all
```

Use the following command to limit the results for each rule check to a maximum of 30:

```
report -file report.txt -max_results 30 -child_rdbs YES
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## run

Used in standard and 3DSTACK+ rule files.

Runs Calibre using the specified rule file and command-line options.

### Usage

```
run {-drc | -lvs | -perc | -xrc} -directory run_dir -rule_deck rules [-run_options options]
```

### Arguments

- **-drc | -lvs | -perc | -xrc**

Required argument set that specifies the type of Calibre run to perform.

- **-directory run\_dir**

Required argument and value set that specifies the path to the run directory. Output files from the run are written to this directory.

- **-rule\_deck rules**

Required argument and value set that specifies the path to a Calibre rule file, such as Calibre nmDRC, Calibre nmLVS or Calibre\_PERC.

- **-run\_options options**

Optional argument and value set that specifies command-line options for the run. Multiple command-line *options* must be enclosed in quotes.

### Description

Runs Calibre nmDRC, Calibre nmLVS, Calibre PERC, or Calibre xRC using the specified rule file and command-line options. The specified Calibre run is done before the Calibre 3DSTACK run.

### Examples

```
run -drc -directory ./DRC \
 -rule_deck ./n45_m.rules -run_options "-turbo -hier"

run -lvs -directory ./LVS \
 -rule_deck ./LVS/n45_l.rules -run_options "-turbo -hier"

run -perc -directory ./PERC \
 -rule_deck ./n22.rules -run_options "-turbo -hier"

run -xrc -directory ./XRC \
 -rule_deck ./rules_xcalh -run_options "-fmt -all"
```

### Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

## Rule Check Commands

## **set\_version**

Sets the syntax version of the Calibre 3DSTACK rule file.

### **Usage**

**set\_version -version *version***

### **Arguments**

- **-version *version***

Required argument and value set that specifies the version of the rule file syntax. Currently, the only valid value for **version** is “1.0”.

### **Description**

Specifies the version of the rule file syntax. This command must appear once in your rule file.

### **Examples**

```
set_version -version 1.0
```

### **Related Topics**

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## source\_filter

Applies filtering options to the imported source netlist file.

### Usage

```
source_filter -chip chip_name
 {-subckt subckt_name -short_pins '{' pins '}' } |
 {-short_device '{' device_type '}' [-short_pins '{' pins '}'] [-constraint "expression"] }
```

### Arguments

- **-chip\_name *chip\_name***

Required argument and value set that specifies a chip name that can be referenced in other commands. If this command is used multiple times, each **chip\_name** must be unique.

- **-subckt *subckt\_name***

Required argument and value set if **-short\_device** is not specified. When **-subckt** is specified, all the instantiations of the specified subcircuit are filtered by shorting the nets connected to specified pins.

- **-short\_pins '{' *pins* '}'**

Required when **-subckt** is specified, optional when **-short\_device** is specified. When **-short\_device** is specified, all the instantiations of the specified device matching with the constraint (if specified) are filtered by shorting the nets connected to the device pins.

If a device or subckt does not have specified pins given by the **-short\_pins** argument, then filtering is not applied.

- **-short\_device '{' *device\_type* '}'**

Required argument and value set if **-subckt** is not specified. Use this to specify that all instantiations in the *device\_type* list that match the constraint (if specified with the **-constraint** expression) are filtered by shorting the nets connected to the device pins.

Optionally, if **-short\_pins** is specified with **-short\_device**, then the nets connected to the specified pins of the device are also shorted.

Model-based filtering for devices can be achieved by specifying the device name followed by the model name enclosed with '(' and ')'.

- **-constraint "*expression*"**

Optional argument and value set that specifies a constraint expression. A single constraint expression can only be applied to one **chip** and **subckt** pair, otherwise an error is issued.

The filtering expression should adhere to the following format:

```
-constraint "property_name condition value"
```

When **-constraint** is specified, the properties used in the filtering expression should be defined in the source SPICE file, otherwise filtering is not applied.

## Description

Use this command to define filtering options for the source netlist data imported for a Calibre 3DSTACK verification run. When the **source\_filter** command is applied, a new SPICE file containing the “filtered” data is generated and stored in the Calibre 3DSTACK DFM database directory. Calibre 3DSTACK run information used by Calibre RVE is automatically updated with the path of the new SPICE file.

The [report](#) file contains a separate section describing the source filtering options.

---

### Note

---

 Filtering is applied for all specified subcircuits or devices regardless of the context (chip specified by **-chip** argument). The **source\_filter** command only accepts netlist files formatted in SPICE. If a non-SPICE netlist is detected, a warning is issued.

The **source\_filter** command is not supported if the **source\_netlist** command includes the **-hier** argument specifying a hierarchical source netlist and white box analysis.

---

## Examples

### Example 1

Apply filtering by shorting pins, *bottom* and *top*, in the *tsv* subcircuit on the *interposer* chip.

```
source_filter -chip interposer -subckt tsv -short_pins {bottom top}
```

### Example 2

Apply filtering by shorting device, *R(SH)*, and pins, *p* and *n*, on the *interposer* chip that meet a constraint of resistance, *R*, equal to zero.

```
source_filter -chip interposer -short_devices {R(SH)} -short_pins {p n} \
-constraint "R == 0"
```

### Example 3

Different constraint expression examples. See the **-constraint** argument for a description of the required format.

```
source_filter -chip i -short_devices {R(SH)} -constraint "R >= 0"
source_filter -chip i -short_devices {R(SH)} -constraint "R == 10"
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## source\_netlist

Imports a source netlist file used for connectivity comparison. It is highly recommended that you use a source netlist during the verification of your assembly.

### Usage

```
source_netlist -file file_name
 -format {SPICE | MGC | CSV | VERILOG [-version {2001 | 1995}] }
 [-hier [-wrap interposer_name]]
 [-case {NO | YES}]
 [-order {column_list}]
 [-subckt_pins {NET_NAME | PIN_NUMBER | PIN_NAME}]
 [-apply_bboxing cell_list]
```

### Arguments

- **-file *file\_name***  
Required argument and value set that specifies the name of the source netlist file.
- **-format {SPICE | MGC | CSV | VERILOG [-version {2001 | 1995}] }**  
Required argument and value set that specifies the format of the input file.

The MGC format contains a list of add\_connection directives that describe single connections in the 3D assembly:

```
add_connection -connection1 {placement1 cell1 net1 pin_instance_name1} \
 -connection2 {placement2 cell2 net2 pin_instance_name2}
```

If you specify to read a Verilog netlist, apply the -version argument to specify the version of the input netlist. The default version is Verilog 2001.

The CSV keyword supports comma-separated value input files in AIF, XSI, or spreadsheet format. Calibre 3DSTACK also writes a SPICE file with the extension .spi to your working directory for debugging purposes. See “[AIF Converter Reference](#)” on page 278 and “[Spreadsheet Converters](#)” on page 285 for details on these files.

- **-order *column\_list***  
Optional argument set that specifies the order of the columns in the spreadsheet file. The allowed values are the following: REF\_DES, PIN\_NUMBER, PIN\_X, PIN\_Y, PIN\_NAME, and NET\_NAME. This option can only be specified with the CSV keyword.
- **-subckt\_pins {NET\_NAME | PIN\_NUMBER | PIN\_NAME}**  
Optional argument set that specifies the type of information specified for the pin column of the CSV file. The default is NET\_NAME. This option can only be specified with the CSV keyword.
- **-hier**  
Optional argument that specifies the source netlist is hierarchical. The source netlist format must be SPICE if -hier is specified.

- **-wrap *interposer\_name***

Optional argument that specifies to automatically wrap the specified interposer subcircuit with a top-level circuit for use in Calibre 3DSTACK. This option only applies to source netlists of format SPICE that use the -hier option. When you specify -wrap, Calibre 3DSTACK does the following at runtime:

- a. Creates a new source netlist file in your run directory named <*source\_netlist\_file*>.wrapped. The new file contains the following:
    - An include statement for the specified source netlist file.
    - A top subcircuit definition with the specified -wrap *interposer\_name*.
    - A placement inside the top subcircuit for the specified interposer and its nets.
  - b. Automatically changes placement mapping as follows:
    - Ignores the interposer placement mapping and issues a warning message.
    - Adds the interposer placement name to mapped placements.
- **-case {NO | YES}**
- Optional argument and value set that controls whether the source netlist file is handled as case-sensitive or not. Note that this option only applies to the source netlist file. The default value is NO (the file is handled regardless of case).
- **-apply\_bboxing *cell\_list***
- Optional argument set that automatically enables black boxes for specified cells in the source netlist. The tool internally generates LVS Box SVRF statements for specified cells for use in LVS checks and the SPICE netlist import.

## Description

Imports connectivity information from the specified *file\_name*.

If there are [connected](#) commands specified in the chip stack rule file, the imported connectivity information is checked against the netlist information extracted from the chip stack. If your source netlist file is case-sensitive, specify the -case YES option.

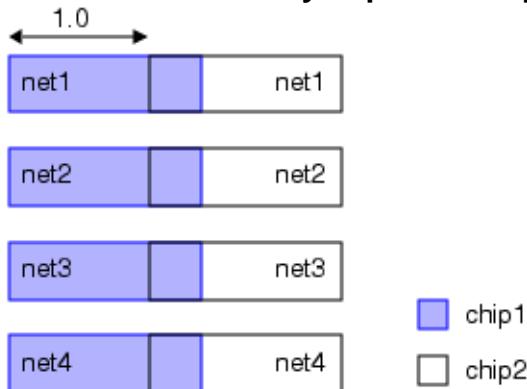
Specify the -hier option if the source netlist is hierarchical. The source\_filter command is not supported when the -hier argument is used. See the [sng](#) command for information on generating a hierarchical source netlist.

You can create a source netlist using the System Netlist Generator. See “[System Netlist Generator Flow Example](#)” on page 327.

## Examples

Consider the chip stack shown in [Figure B-2](#). In this chip stack, two layouts (chip1 and chip2) are arranged such that they overlap. Polygons on each layout contain text objects labeled net1 through net4.

**Figure B-2. Connectivity Import Example**



The chip stack rule file for this example is shown as follows:

```

set_version -version 1.0

layout -chip_name c1 -primary TOPCELL -path ./chip1.gds -system GDS
layout -chip_name c2 -primary TOPCELL -path ./chip2.gds -system GDS

source_netlist -file spice.spi -format SPICE -case NO

layer -layer m1 -chip c1 -layer_number 0
layer -layer m2 -chip c2 -layer_number 0

place_chip -placement c1p -chip c1 -x_origin 0 -y_origin 0
place_chip -placement c2p -chip c2 -x_origin 1 -y_origin 0

connect c1p_m1 c2p_m2

attach_text -placement c1p_m1 -text_placement c1p_m1
attach_text -placement c2p_m2 -text_placement c2p_m2

connected -check_name con_check -placement1 c1p_m1 -placement2 c2p_m2

```

Assume that the *spice.spi* file contains the results of the [export\\_connectivity](#) command with the SPICE option specified:

```

.SUBCKT TOPCELL_3DIC
Xc1p 4 3 2 1 TOPCELL
Xc2p 4 3 2 1 TOPCELL
.ENDS TOPCELL_3DIC

.SUBCKT TOPCELL net1 net2 net3 net4
.ENDS TOPCELL

```

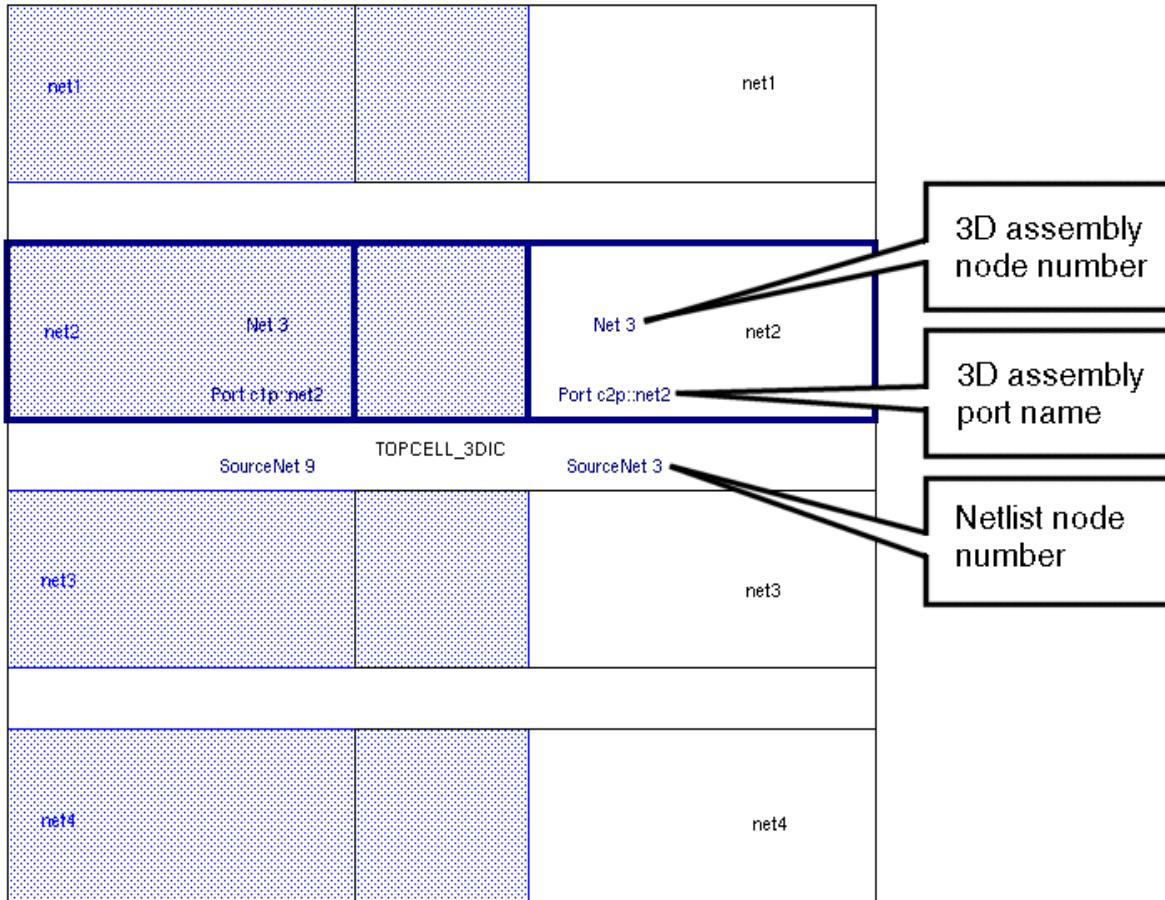
Since the connectivity information in this file matches the connectivity specified with the connect statement, this file does not produce errors. However, suppose that the .SUBCKT TOPCELL\_3DIC definition is modified as follows:

```
.SUBCKT TOPCELL_3DIC
Xc1p 4 9 2 1 TOPCELL
Xc2p 4 3 2 1 TOPCELL
.ENDS TOPCELL_3DIC
```

Now, since an incorrect node number has been introduced, the connected command produces two results that identify the offending polygons. [Figure B-3](#) shows these results highlighted in Calibre DESIGNrev.

Note that the properties attached to each polygon identify the 3D assembly node number, the port name, and the netlist node number.

**Figure B-3. Calibre DESIGNrev Output**



## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

[System Netlist Generator Flow and Invocation](#)

## **svrf\_specs**

Includes a file containing SVRF statements in the run.

---

### **Caution**

---

 The specification file should only be used to include SVRF statements that control layout read behavior, such as Layout Input Exception Severity. Any other SVRF statements (for example, statements that perform layer operations) are not recommended and can cause unintended behavior.

---

### **Usage**

**svrf\_specs** *svrf\_file*

### **Arguments**

- *svrf\_file*

Required path to a file containing SVRF specification statements.

### **Examples**

```
svrf_specs ./3dprocess.svrf
```

## warning\_severity

Option for the standard syntax.

Specifies the severity level for warning messages.

### Usage

```
warning_severity -warning {message_id | *} -severity {0 | 1 | 2} [-count count]
```

### Arguments

- **-warning {*message\_id* | \*}**

Specifies the warning message by the ID. You can also specify \* to include all warning messages.

- **-severity {0 | 1 | 2}**

Sets the severity level of the message. The following keywords are supported:

0 — Disables the warning message. The tool does not print this message.

1 — Enables the warning message (the default).

2 — Configures the warning message as an error. The tool immediately exits from arun after printing the contents of the message.

- **-count *count***

Sets an upper limit on the number of messages printed when the -severity is set to 1. This must be an integer greater than 0.

### Examples

```
warning_severity -warning 3DSTACK_WARNING_108 -severity 1 -count 20
```

## Assembly Commands

Assembly commands allow you to build a physical model of your stacked IC.

For standard Calibre 3DSTACK syntax rule files, the following arguments enable the generation of the assembly cross-section view:

- Apply either the -z\_origin or-level options in the [place\\_chip](#) and [place\\_layer](#) commands. These arguments are mutually exclusive. The -z\_origin option specifies the height of the die from the bottom of the stack. The -level option indicates the order of the die in the stack.
- Apply the -thickness option in the [layout](#) command. This option defines the thickness of the die in microns. If this is not specified, you can use the [CALIBRE\\_3DSTACK\\_DEFAULT\\_THICKNESS](#) environment variable to set a global thickness for all dies. If neither the environment variable nor the -thickness option is specified, the thickness of the placement level is calculated based on the minimum width and height of the dies in the same level.

**Table B-3. Assembly Commands**

| Name                                      | Description                                                                                                                                                                                                       |
|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">anchor</a>                    | Defines an anchor name and location for a chip. This is helpful during the assembly of the stack.                                                                                                                 |
| <a href="#">attach_text</a>               | Attaches text to a placed layer for connectivity extraction and checking.                                                                                                                                         |
| <a href="#">connect (Standard Syntax)</a> | Specifies electrical connections among input layer objects.                                                                                                                                                       |
| <a href="#">derived_connect</a>           | Specifies electrical connectivity for layers that you want to use with connectivity-based SVRF operations, such as Net Area Ratio. This is equivalent to the die -wb_connect -use_in_svrf extended syntax option. |
| <a href="#">ignore_pin</a>                | Excludes specified pins from a Calibre 3DSTACK run.                                                                                                                                                               |
| <a href="#">ignore_trailing_chars</a>     | Removes specified characters from the end of text labels.                                                                                                                                                         |
| <a href="#">import_net_map</a>            | Imports a text file that contains a list of net mapping statements.                                                                                                                                               |
| <a href="#">import_pin_map</a>            | Imports a text file that contains a list of pin mapping statements.                                                                                                                                               |
| <a href="#">import_text_labels</a>        | Imports text objects from a file and applies them to a specified chip.                                                                                                                                            |
| <a href="#">layer</a>                     | Defines an original or derived layer in the assembly.                                                                                                                                                             |
| <a href="#">layout</a>                    | Imports a GDS or OASIS database that contains a chip used in the assembly.                                                                                                                                        |

**Table B-3. Assembly Commands (cont.)**

| Name           | Description                                                                                                                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| layout_case    | Defines the case-sensitivity of the top-level assembly layout.                                                                                                                                                                |
| layout_primary | Defines the name of the top-level cell for the 3D assembly.                                                                                                                                                                   |
| map_placement  | Associates layout placements in the assembly to placements specified in the source netlist file.                                                                                                                              |
| net_map        | Specifies nets that can use a different name during connectivity checking. This command is useful for waiving known connectivity errors by adjusting net names at runtime.                                                    |
| pin_map        | Specifies pins that can use a different name during connectivity checking. This command is useful for waiving known connectivity errors by adjusting pin names at runtime. This operation does not rename pins in the layout. |
| pin_swap       | Specifies that certain pins on a chip are interchangeable during connectivity checking.                                                                                                                                       |
| place_chip     | Defines the placement name, orientation, and scaling factor for a chip.                                                                                                                                                       |
| place_layer    | Defines the placement name, orientation, and scaling factor for a layer.                                                                                                                                                      |
| rename_text    | Edits existing layout text in the assembly.                                                                                                                                                                                   |

## anchor

Defines an anchor name and location for a chip. This is helpful during the assembly of the stack.

### Usage

```
anchor -name anchor_name
 -chip chip_name -x_origin x_offset -y_origin y_offset
 -layer layer_name
 -text text
```

### Arguments

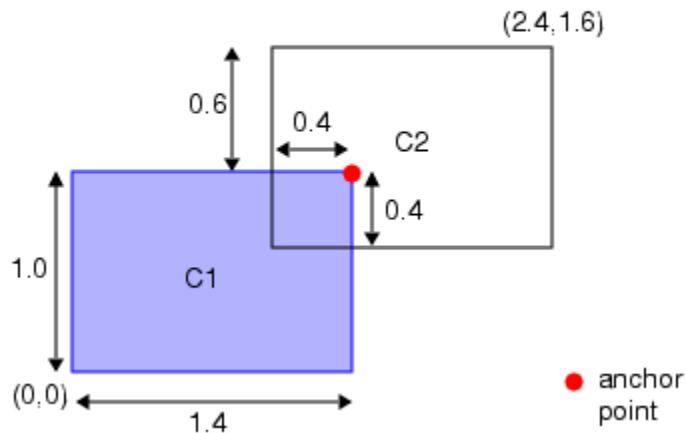
- **-name *anchor\_name***  
Required argument and value set that specifies the name of the anchor.
- **-chip *chip\_name***  
Required argument and value set that specifies the name of the chip to which to assign the anchor. Cannot be specified with **-text**.
- **-x\_origin *x\_offset***  
Required argument and value set that specifies the offset from the origin along the x-axis.  
The value of *x\_offset* is in microns.
- **-y\_origin *y\_offset***  
Required argument and value set that specifies the offset from the origin along the y-axis.  
The value of *y\_offset* is in microns.
- **-layer *layer\_name***  
Required argument set that specifies the name of the layer on a chip.
- **-text *text***  
Required argument that specifies the text label name in the *layer\_name* layer to identify the anchor location. This argument is mutually exclusive with -x\_origin and -y\_origin. If there are multiple layers defined in the rule file with the same name, then you must specify the -chip option to identify the layer from which the *text* is retrieved.

### Description

Assigns an anchor name and location to a chip, which can be used in the [place\\_chip](#) or [place\\_layer](#) commands for simplified alignment of a chip stack.

Assume that you want to place chips C1 and C2 in the configuration shown in the following figure:

**Figure B-4. Anchoring Example**



One way this can be accomplished is by defining two anchors as follows:

```
anchor -name a1 -chip C1 -x_origin 1.4 -y_origin 1.0
anchor -name a2 -chip C2 -x_origin 0.4 -y_origin 0.4
```

where anchor a1 is relative to the origin of chip C1, and anchor a2 is relative to the origin of chip C2.

The chips can then be placed as follows:

```
layer -layer 0 -chip C1 -layer_number 0
layer -layer 1 -chip C2 -layer_number 0

place_chip -placement C1p -chip C1 -x_origin 0 -y_origin 0
place_chip -placement C2p -chip C2 -anchor_placement C1p -anchor_from a2 \
-anchor_to a1
```

## Examples

```
anchor -name interposer_mem -chip interposer -x_origin 30.0 \
-y_origin 14.0
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## attach\_text

Attaches text to a placed layer for connectivity extraction and checking.

### Usage

```
attach_text -placement placed_layer -text_placement placed_text_layer
[-net_text] [-text_depth depth] [-no_update]
```

### Arguments

- **-placement *placed\_layer***

Required argument and value set that specifies the name of a placed layer.

- **-text\_placement *placed\_text\_layer***

Required argument and value set that specifies the name of a text layer. This layer does not need to be unique, and can be the same layer specified for *placed\_layer*.

- **-net\_text**

Optional argument that specifies that the text attached to the layer represents net names instead of pin names (the text labels of this layer are not interpreted as pins of the die or package). This option allows Calibre 3DSTACK to extract layout net names from the text labels of whitebox connectivity layers and check them against net names in source netlist.

To perform checking on net layers, apply the *-net\_mismatch* option of the [connected](#) command.

- **-text\_depth *depth***

Optional argument that specifies the hierarchical depth of the text on the *placed\_text\_layer*. The *depth* value is specified as one or more positive integers (multiple values are specified using braces {}). Note that you cannot specify more than one **attach\_text** command with the same *placed\_text\_layer* but different *depth* values.

- **-no\_update**

Optional argument that specifies the input layers are not included in the extracted layout netlist.

### Description

Associates a text layer with a placed layer to be used for connectivity data extraction from the 3D assembly and for connectivity checking (using the [connected](#) command).

If either the *placed\_layer* or *placed\_text\_layer* values reference a placement that is defined using [place\\_chip](#), the values must be specified in the following format:

```
<placement_name>_<layer>
```

If these values reference a placement defined using [place\\_layer](#), the placement name is specified directly.

The `-no_update` argument should be used when associating text with a source placement layer for use with a [locations](#) check. This prevents the source layer from being included in the extracted layout netlist.

## Examples

### Example 1

```
attach_text -placement m_mtsv -text_placement m_mttx
attach_text -placement l_ltsv -text_placement l_lttx

connected -check_name connectivity_check -placement1 m_mtsv \
-placement2 l_ltsv
```

### Example 2

```
attach_text -placement m_mtsv -text_placement m_mttx -text_depth 1
attach_text -placement l_ltsv -text_placement l_lttx -text_depth {1 2}
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## connect (Standard Syntax)

Specifies electrical connections among input layer objects.

### Usage

#### Syntax 1

**connect *layer1* [*layerN* ...]**

#### Syntax 2

**connect *layer1* *layer2* [*layerN* ...] BY *layerC***

### Arguments

- ***layer1***

A required value that specifies an original polygon layer.

- ***layer2***

A required value that specifies an original polygon layer. This value is required in Syntax 2, but is not used in Syntax 1.

- **BY *layerC***

A required value that specifies an original polygon layer, which is the contact layer. This value is required in Syntax 2, but is not used in Syntax 1.

### Description

Specifies electrical connections among input layer objects. Syntax 1 specifies an electrical connection among *layer1* through *layerN* objects (typically shapes or paths) that abut or overlap, regardless of any other objects present on unspecified layers. Syntax 2 (connect BY) specifies electrical connections among *layer1* and *layerC* objects, and the first available *layer2* through *layerN* object, in that order.

If either the *layer1*, *layer2*, or *layerC* values reference a placement that is defined using [place\\_chip](#), the values must be specified in the following format:

<placement\_name>\_<layer>

If these values reference a placement defined using [place\\_layer](#), the placement name is specified directly.

Refer to the [Connect](#) statement in the *Standard Verification Rule Format (SVRF) User's Manual* for more information and examples on the usage of this command.

---

#### Note

 The connect *layer1*, *layer2*, and BY *layerC* arguments do not accept derived layers.

---

## Examples

```
connect clp_m1 clp_m2 BY clp_v1
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## **derived\_connect**

Specifies electrical connectivity for layers that you want to use with connectivity-based SVRF operations, such as Net Area Ratio. This is equivalent to the die -wb\_connect -use\_in\_svrf extended syntax option.

### **Usage**

#### **Syntax 1:**

**derived\_connect layer1 [layerN ...]**

#### **Syntax 2:**

**derived\_connect layer1 layer2 [layerN ...] BY layerC**

### **Arguments**

- **layer1**

A required value that specifies an original polygon layer.

- **layer2**

A required value that specifies an original polygon layer. This value is required in Syntax 2, but is not used in Syntax 1.

- **BY layerC**

A required value that specifies an original polygon layer, which is the contact layer. This value is required in Syntax 2, but is not used in Syntax 1.

### **Description**

The operation and syntax is the same as connect, but it enables the established connectivity layers to be used with connectivity-based SVRF operations in the tvf\_block command.

Refer to the Calibre 3DSTACK “[connect \(Standard Syntax\)](#)” on page 394 and the [Connect](#) statement in the *Standard Verification Rule Format (SVRF) User’s Manual* for more information and examples on the usage of this command.

### **Examples**

```
derived_connect c1p_m1 c1p_m2 BY c1p_v1
```

## ignore\_pin

Excludes specified pins from a Calibre 3DSTACK run.

### Usage

**ignore\_pin -placement *placement\_name* -pin '{'*expression*'}'**

### Arguments

- **-placement *placement\_name***

Required argument and value set that specifies a name for a placed chip.

- **-pin '{'*expression*'}'**

Required argument and value set that specifies an expression that matches one or more pin names. The ***expression*** can be any Tcl regular expression and is not case-sensitive. The opening '{' and closing '}' braces are required.

### Description

Excludes the specified pin(s) from a Calibre 3DSTACK analysis. The **ignore\_pin** command only applies to layout pins. The source schematic is considered golden and not impacted.

---

#### Note

 To list all successfully ignored pins in the report, include “-report\_ignored\_pins YES” in the report command.

---

### Examples

```
ignore all pin instances of placement p1 that contain the substring
"VSS" in their names.

ignore_pin -placement p1 -pin {vss}

ignore all pin instances of placement p1 with the name "VDD".

ignore_pin -placement p1 -pin {^VDD$}

ignore all pin instances of placement p1 that contain either the
substrings "VSS" or "VDD" in their names.

ignore_pin -placement p1 -pin {vss|vdd}
```

### Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## **ignore\_trailing\_chars**

Removes specified characters from the end of text labels.

### **Usage**

**`ignore_trailing_chars {character...}`**

### **Arguments**

- ***character...***

Required list of characters to remove from the end of all text labels.

### **Description**

Removes up to one character from the end of text labels during connectivity extraction. Trailing characters, such as “:”, are sometimes used on text labels to indicate implied connections between polygons that are not physically connected.

In the 3DSTACK assembly, it is possible for placement labels to inherit trailing characters. For example, the VSS and CLK nets may also be VSS: and CLK:. These are interpreted as different nets, even though they are the same.

Use this command to ignore trailing characters so that the nets are connected during connectivity comparison.

### **Examples**

The following text label pairs are used in a design:

**Table B-4. Nets With Trailing Characters In Design**

| Net | Standard Text Label | Label With Implied Connectivity |
|-----|---------------------|---------------------------------|
| VDD | VDD                 | VDD:                            |
| VSS | VSS                 | VSS.                            |
| CLK | CLK                 | CLK::                           |
| ADR | ADR                 | ADR,                            |

Use the following command to merge the desired nets:

```
ignore_trailing_chars ":", "
```

The following nets are extracted during connectivity analysis as a result of the command:

```
VDD VSS CLK CLK: ADR
```

Note, that only the last character (“：“) was removed from the CLK net. Even though that character is specified, Calibre 3DSTACK only ignores the last character in the string, so the CLK and CLK: text labels are still considered different nets.

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## import\_net\_map

Imports a text file that contains a list of net mapping statements.

### Usage

**import\_net\_map -file *filename***

### Arguments

- **-file *filename***

Required argument that specifies the path to a file that contains a list of net mapping statements.

### Description

The file specified by the -file argument set contains one net mapping statement per line. Each line contains two space-separated entries as follows:

```
original_net_name new_net_name
...
```

Where the **original\_net\_name** argument is an existing net name in the design and the **new\_net\_name** is the desired name.

---

#### Note

---

 The same net cannot be renamed to different values.

---

### Examples

In this example, chip\_1 and chip\_2 have nets called “global\_reset”. These nets are connected to a net named “reset” on chip\_3. This is logically correct, but this could result in misleading LVS errors. In this case, you can rename the nets by specifying a file that contains the correct mapping:

```
import_nets -file ./import_nets.txt
```

where the contents of *import\_nets.txt* is the following:

```
global_reset reset
```

## import\_pin\_map

Imports a text file that contains a list of pin mapping statements.

### Usage

**import\_pin\_map -file *filename***

### Arguments

- **-file *filename***

Required argument that specifies the path to a file that contains a list of pin mapping statements.

### Description

The file specified by the -file argument set contains one pin mapping statement per line. Each line contains two space-separated entries as follows:

```
original_pin_name new_pin_name
...

```

Where the *original\_pin\_name* argument is an existing pin name in the design and the *new\_pin\_name* is the desired name.

---

#### Note

 The same pin cannot be renamed to different values.

---

### Examples

In this example, chip\_1 and chip\_2 have pins called “global\_reset”. These pins are connected to a pin named “reset” on chip\_3. This is logically correct, but this could result in misleading LVS errors. In this case, you can rename the pins by specifying a file that contains the correct mapping:

```
import_pins -file ./import_pins.txt
```

where the contents of *import\_pins.txt* is the following:

```
global_reset reset
```

## import\_text\_labels

Imports text objects from a file and applies them to a specified chip.

### Usage

```
import_text_labels -chip chip_name -file file_name
 [-order column_list] [-xsi {NET_NAME | PIN_NUMBER | PIN_NAME}]
 [-top_level_coords]
```

### Arguments

- **-chip *chip\_name***  
Required argument and value set that specifies the chip name being referenced (chip names are defined using the [layout](#) command).
- **-file *file\_name***  
Required argument and value set that specifies the path to a file. This file must contain only Layout Text statements (one per line). Each Layout Text statement is specified in the following form:

```
LAYOUT TEXT text_name x y layer text_type cell_name
```

where:

- *text\_name* — required name of a text object.
- *x y* — required pair of floating-point coordinates in user units specifying the location of the text label. These coordinates are in the cell context of *cell\_name*.
- *layer* — required original layer number specifying the text layer.
- *text\_type* — required non-negative integer that indicates a text object's text type.
- *cell\_name* — required name of a cell in which the text object is placed.

For example:

```
LAYOUT TEXT BACKUP_1 -0.27 0.28 10215 1 MTR_TOP2_ip_flip
LAYOUT TEXT BACKUP_1_GROUP -0.27 0.28 10131 0 MTR_TOP2_ip_flip
LAYOUT TEXT tx1_n -0.85 1.26 10215 1 MTR_TOP2_ip_flip
LAYOUT TEXT tx1_n_GROUP -0.85 1.26 10131 0 MTR_TOP2_ip_flip
```

---

#### Note

 You can also specify an XSI netlist as the *file\_name* argument, which does not follow this format.

---

- **-order *column\_list***

Optional argument set that specifies the order of the columns in the spreadsheet file, if the *file\_name* is an XSI netlist. The allowed values are the following: REF\_DES,

PIN\_NUMBER, PIN\_X, PIN\_Y, PIN\_NAME, NET\_NAME, COMPONENT\_NAME, and IGNORE. The default order is as follows:

```
NET_NAME IGNORE IGNORE COMPONENT_NAME PIN_NUMBER REF_DES
```

- -xsi {NET\_NAME | PIN\_NUMBER | PIN\_NAME}

Optionally specifies the column in the XSI netlist from which to retrieve text labels. The default is the NET\_NAME column.

- -top\_level\_coords

Specifies that the coordinates for text labels in the *file\_name* belong to the top level.

## Description

Adds text objects to a chip. This command must appear after the [layer](#) and [layout](#) commands in the chip stack rule file. Any layers declared in the *file\_name* that do not exist in the layout file are created.

## Examples

```
import_text_labels -chip stack_die -file lvsText_dieWrapper.txt
```

## layer

Defines an original or derived layer in the assembly.

### Usage

```
layer -layer layer_name -chip chip_name
 { {-layer_number layer_number['.data_type']} ... | -svrf '{' layer_derivation'}' [-show]
 [-level level_number] [-top | -bottom] [-top_only] [-texttype]
```

### Arguments

- **-layer *layer\_name***

Required argument and value set that specifies the name of the layer.

- **-chip *chip\_name***

Required argument and value set that specifies the chip name being referenced (chip names are defined using the [layout](#) command).

- **-layer\_number *layer\_number*['.data\_type']**

Required argument and value set that specifies the layer number and (optionally) the datatype. Optional additional **-layer\_number** arguments are allowed for geometric and text data. This option must not be specified with the **-svrf** argument.

- **-svrf '{' *layer\_derivation*'}'**

Required argument and value set that specifies a layer derivation using SVRF commands. The layer derivation must be enclosed in braces. This option must not be specified with the **-layer\_number** argument.

The *layer\_derivation* body is similar to that of a rule check statement. It must have at least one standalone layer operation, which is the output layer. If two output layers are specified, they are merged. Local layer definitions may be used in the layer derivation.

See the “Description” section for additional information.

- **-show**

Optional argument that writes derived layers to the generated assembly view of your stack. If this option is not specified, no derived layers are included in the assembly layout view. This option must be specified with the **-svrf** argument set.

- **-level *level\_number***

Optional argument and value set that specifies the level of the layer in the 3D stack. Positive and negative integer values are allowed. Use this argument to assist in drawing the system nets and 3D views of the 3D assembly. Layer level numbers are reported in the 3DSTACK rule file.

- **-top**

Optional argument that indicates that the layer is physically placed on the top side of the die. This argument is used to determine the die bump layer placement in the cross-section view.

- -bottom

Optional argument that indicates that the layer is physically placed on the bottom side of the die. This argument is used to determine the die bump layer placement in the cross-section view.

- -top\_only

Optional keyword that specifies to only use the top-level geometry that belongs to the specified layer. When this option is specified, geometry that belongs to all cells other than the top cell on this layer is ignored. The -top\_only option of the [layout](#) command takes precedence over this command option.

---

#### Note

---

 When multiple -layer\_number arguments are specified in the argument list, the layers are merged (ORed) together at the design compilation stage.

---

- -texttype

Optionally defines the derived layer as a TEXTTYPE layer.

## Description

Defines the name of an original or derived layer. Definitions for empty layers are allowed. A warning is given for empty layers.

If you create derived layers with the **-svrf** argument, Calibre 3DSTACK generates a layout file that includes the derived layers. The layout file is used in assembly generation and further processing. The generated layout file is named **chip\_name.gds** or **chip\_name.oas** and is saved in the working directory with a unique suffix. The generated layout file uses the default precision of 1000. The transcript includes the statement “Creating new layout file for **chip\_name**”.

## Examples

### Example 1

```
layer -layer base_bottom_interface_shapes -chip base -layer_number 52
```

### Example 2

This example demonstrates **layer** usage with multiple layer\_number arguments.

```
layer -layer base_bottom_interface_shapes -chip base \
-layer_number 51 -layer_number 51.1
```

### Example 3

This example uses the -svrf argument to supply a layer derivation.

---

```
layer -layer CUPB -chip die -layer_number 220.170
layer -layer CUPB_pin -chip die -layer_number 220.175
layer -layer CUPB_merged -chip die -svrf {
 CUPB OR CUPB_pin
}
layer -layer die_extent -chip die -svrf {
 MERGE (EXTENT DRAWN CUPB CUPB_pin)
}
```

Local layer definitions may be used in the layer derivation. For example, the last layer derivation in the previous example can be written this way:

```
layer -layer die_extent -chip die -svrf {
 temp_x = EXTENT DRAWN CUPB CUPB_pin
 MERGE temp_x
}
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

# layout

Imports a GDS or OASIS database that contains a chip used in the assembly.

---

## Note

 If physical precisions differ between layouts, the generated assembly file uses the least common denominator of the precisions of the input layouts.

For example, if two input layouts have precisions of 1/2000 and 1/3000, the output file is generated with a precision of 1/6000.

---

## Usage

```
layout -chip_name chip_name -primary top_cell_name -path layout_path
 -system {GDS | OASIS} [-original_extent] [-top_only] [-exclude cells] [-interposer]
 [-thickness value] [-netlist spice_file]
```

## Arguments

- **-chip\_name *chip\_name***

Required argument and value set that specifies a chip name that can be referenced in other commands. If this command is used multiple times, each ***chip\_name*** must be unique.

- **-primary *top\_cell\_name***

Required argument and value set that specifies the name of the top cell in the design.

- **-path *layout\_path***

Required argument and value set that specifies the path to the layout file. The ***layout\_path*** can be relative or absolute. Calibre 3DSTACK treats any input file with the extension .gz or .Z as a compressed file. The gunzip application must be available in your environment.

- **-system {GDS | OASIS}**

Required argument and value set that specifies the format of the layout file.

- **-original\_extent**

Optional argument that generates a new layer that represents the total extent of the original die. This is used for die spacing checks. The extent is based off of all the original layers in the specified layout file. The die extent is written to a new *placement\_name\_EXTENT* layer in the *3dstack\_assembly.oas* file. When geometrical checks are specified for placements of a chip with **-original\_extent**, the new layer is considered.

- **-top\_only**

Optional keyword that specifies to only import the top-level geometry of the die. When this option is specified, geometry that belongs to all cells other than the top cell is ignored. This option takes precedence over the **-top\_only** option of the [layer](#) command.

- **-exclude *cells***

Optional argument and list of cells to exclude from the layout.

- **-interposer**

Optional argument that specifies that the layout is an interposer. This is used for 2.5D ICs when you want to generate a hierarchical netlist using the [export\\_connectivity](#) command with the -hier option. You can apply this option more than once. If you specify more than one interposer, Calibre 3DSTACK ignores the export\_connectivity -hier option and generates a flat assembly netlist.

- **-thickness *value***

Optional argument set that defines the thickness of the die in microns. Calibre 3DSTACK uses the thickness values to generate a cross-sectional assembly view of the stack. See “[Assembly Views](#)” on page 35 for details.

If this option is not specified, you can use the `CALIBRE_3DSTACK_DEFAULT_THICKNESS` environment variable to set a global thickness for all dies. If neither the environment variable nor the -thickness option is specified, the thickness of the placement level is calculated based on the minimum width and height of the dies in the same level.

- **-netlist *spice\_file***

Optional keyword and path to a SPICE netlist that contains the subcircuit definition for the specified layout. If you specify this option, Calibre 3DSTACK automatically calls the System Netlister tool to build a complete netlist during a verification run. If the `top_cell_name` and the subcircuit cell name are different, the layout and netlist cell correspondence is determined by the map\_placement command.

## Examples

### Example

Import three chips into the assembly.

```
layout -chip_name interposer -primary interposer \
 -path ${dsn_dir}/interposer.gds -system GDS -original_extent

layout -chip_name controller -primary controller \
 -path ${dsn_dir}/controller.gds -system GDS -original_extent

layout -chip_name ram -primary ram -path ${dsn_dir}/ram.gds -system GDS \
 -original_extent
```

### Example

Import the `controller.gds` layout into the assembly, but do not include the PLL and cap\_array cells.

```
layout -chip_name controller -primary controller \
 -path ${dsn_dir}/controller.gds -system GDS -original_extent \
 -exclude {PLL cap_array}
```

### Example

Declare a chip as an interposer and export a hierarchical SPICE netlist of the assembly.

```
layout -chip_name interposer -primary interposer -path interposer.gds \
 -system GDS -original_extent -interposer

Declare other chip layouts, connectivity, and assembly operations
...
export_connectivity -file 3dstack_hier.sp -format SPICE -hier
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## layout\_case

Defines the case-sensitivity of the top-level assembly layout.

### Usage

```
layout_case {no | yes}
```

### Arguments

- no | yes

Optional value that controls the case-sensitivity handling of the assembly layout. If you specify yes, the layout net and pin names are handled in case-sensitive manner. The default is no.

### Examples

```
layout_case yes
```

## layout\_primary

Defines the name of the top-level cell for the 3D assembly.

### Usage

**layout\_primary** *name*

### Arguments

- *name*

Required value that specifies the name of the top-level cell for the 3D assembly. The default name is “TOPCELL\_3DI” if this command does not appear in the chip stack rule file.

### Examples

```
layout_primary TOPCELL_3DSTACK
```

### Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## map\_placement

Associates layout placements in the assembly to placements specified in the source netlist file.

### Usage

**map\_placement -placement *layout\_placement\_name* -source *source\_placement\_name***

### Arguments

- **-placement *layout\_placement\_name***

Required parameter and value set that specifies the layout placement of a cell.

- **-source *source\_placement\_name***

Required parameter and value set that specifies the source placement of a cell. Hierarchical names are supported if the source netlist is hierarchical; use the forward slash (/) as the hierarchy separator. For example:

- A source netlist has a top cell named 3DSTACK\_TOP that contains one placement, named INTERPOSER, of the sub-circuit INTERPOSER\_CHIP.
- The INTERPOSER\_CHIP has two placements, named DIE1 and DIE2.

In order to map to DIE1, you specify the hierarchical *source\_placement\_name* as INTERPOSER/DIE1.

### Description

This command maps the layout placement to the source netlist placement imported with the [source\\_netlist](#) command. The mapping information is used during connectivity checking (LVS) if the names used in your rule file and source netlist are different.

This command is necessary when the source names in your netlist file cannot be changed, but must match the placement names in the assembly operations given by your Calibre 3DSTACK rule file. It also is necessary to use **map\_placement** when forcing certain names to the assembly's cells.

Cell mapping from layout to source is implicit. That is, if a placement from the layout is mapped to the placement from the source, then the layout placement cell (the chip) is implicitly mapped to the source placement cell.

---

#### Note

 The map\_placement command is order dependent. It must be specified after the [place\\_chip](#) command in the 3DSTACK rule file.

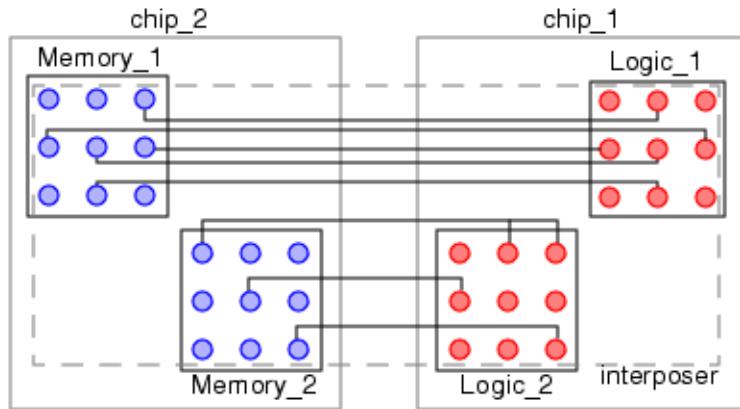
---

### Examples

This example shows the commands for mapping the layouts shown in [Figure B-5](#). In this example, blocks Memory\_1 and Memory\_2 are the same layout. Blocks Logic\_1 and Logic\_2 are also the same layout. In the source schematic, the memory block is named memory\_block\_1

and memory\_block\_2. In order to associate the layout placements named Memory\_1 and Memory\_2 with the source netlist names for LVS, the map\_placement command is used. The same associations are made with the logic blocks.

**Figure B-5. Multiple Chips Using the Same Memory Cell**



These statements map the layout placements to the source placements:

```
#mapping memory blocks
map_placement -placement Memory_1 -source memory_block_1
map_placement -placement Memory_2 -source memory_block_2

#mapping logic blocks
map_placement -placement Logic_1 -source logic_block_1
map_placement -placement Logic_2 -source logic_block_2
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## net\_map

Specifies nets that can use a different name during connectivity checking. This command is useful for waiving known connectivity errors by adjusting net names at runtime.

### Usage

**net\_map -nets *list\_of\_nets* -to *net\_name***

### Arguments

- **-nets *list\_of\_nets***

Required list of nets to rename.

- **-to *net\_name***

Required name for the list of specified nets. All nets in the *list\_of\_nets* argument are renamed to the specified value.

### Description

The net mapping operation only occurs in memory during connectivity checking. Net mapping is written to the Calibre 3DSTACK report file.

Multiple net mapping statements are allowed (many-to-one):

```
net_map -nets {VDD_a} -to VDD
net_map -nets {VDD_m} -to VDD
```

However, you cannot apply the `net_map` statement multiple times to the same net (one-to-many). For example, the following is *not* allowed:

```
net_map -nets {VDD_a} -to VDD
net_map -nets {VDD_a} -to VDD_analog
```

### Examples

In this example, chip\_1 and chip\_2 have nets called “global\_reset”. These nets are connected to a net named “reset” on chip\_3. This is logically correct, but this could result in misleading LVS errors. In this case, you can map the nets during connectivity checking by specifying the following command:

```
net_map -nets {global_reset} -to reset
```

## pin\_map

Specifies pins that can use a different name during connectivity checking. This command is useful for waiving known connectivity errors by adjusting pin names at runtime. This operation does not rename pins in the layout.

---

### Note

---

 If you want to change text labels in the design, see “[rename\\_text](#)” on page 424.

---

### Usage

`pin_map -pins list_of_pins -to pin_name`

### Arguments

- **-pins *list\_of\_pins***  
Required list of pins to rename.
- **-to *pin\_name***  
Required name for the list of specified pins. All pins in the *list\_of\_pins* argument are renamed to the specified value.

### Description

This command behaves similar to `rename_text`, except that the pin names in the layout and source are not physically altered. The pin mapping operation only occurs in memory during connectivity checking.

---

### Note

---

 If the specified pin name does not exist in the layout, then the mapping operation is ignored.

---

Multiple pin mapping statements are allowed (many-to-one):

```
pin_map -pins {VDD_a} -to VDD
pin_map -pins {VDD_m} -to VDD
```

However, you cannot apply the `pin_map` statement multiple times to the same pin (one-to-many). For example, the following is *not* allowed:

```
pin_map -pins {VDD_a} -to VDD
pin_map -pins {VDD_a} -to VDD_analog
```

### Examples

In this example, `chip_1` and `chip_2` have pins called “`global_reset`”. These pins are connected to a pin named “`reset`” on `chip_3`. This is logically correct, but this could result in misleading LVS

errors. In this case, you can map the pins during connectivity checking by specifying the following command:

```
pin_map -pins {global_reset} -to reset
```

## pin\_swap

Specifies that certain pins on a chip are interchangeable during connectivity checking.

### Usage

`pin_swap -chip chip_name -pins list_of_pins`

### Arguments

- **-chip *chip\_name***

Required name of a chip used in your assembly. The pin swapping operation applies to all placements of the specified chip.

- **-pins *list\_of\_pins***

Required list of pins to mark as interchangeable.

### Description

Use this command to mark certain pins on a chip as swappable (logically equivalent) during connectivity checking. Multiple pin\_swap statements are allowed for the same chip, but the same pin cannot be specified in more than one pin\_swap statement.

### Examples

Set the power pins on the ram chip as equivalent:

```
pin_swap -chip myram -pins {vdd vdd_a vdd_bus vdd_cp}
```

## place\_chip

Defines the placement name, orientation, and scaling factor for a chip.

### Usage

```
place_chip -placement placement_name -chip chip_name
{ {-x_origin x_offset -y_origin y_offset} | {-anchor_placement anchor_placement_name
-anchor_from_name -anchor_to_name} }
[-rotate angle] [-magnify factor]
[-flip {x | y}] [-level value | -z_origin value]
```

### Arguments

- **-placement *placement\_name***

Required argument and value set that specifies a unique placement name that can be referenced with other commands in the following format:

<*placement\_name*>\_<*layer*>

- **-chip *chip\_name***

Required argument and value set that specifies the chip name being referenced (chip names are defined using the [layout](#) command).

- **-x\_origin *x\_offset***

Argument and value set that specifies the offset from the origin along the x-axis. The value of *x\_offset* is in microns.

- **-y\_origin *y\_offset***

Argument and value set that specifies the offset from the origin along the y-axis. The value of *y\_offset* is in microns.

- **-anchor\_placement *anchor\_placement\_name* -*anchor\_from\_name* -*anchor\_to\_name***

Argument and value set that aligns the locations of *from\_name* and *to\_name* (relative to the origins of their respective chip names) at the location of the specified *anchor\_placement\_name*. Mutually exclusive with **x\_origin** and **y\_origin**. See the description section under the [anchor](#) command for an example.

- **-rotate *angle***

Optional argument and value set that specifies a rotation *angle* in degrees, which must be either 0 or a multiple of 90 (positive or negative). The layer is rotated about the origin (after *x\_offset* and *y\_offset* values are applied) by the specified angle. Positive angle values yield a counter-clockwise rotation; negative angle values yield a clockwise rotation.

- **-magnify *factor***

Optional argument and value set that specifies a magnification factor by which to expand or shrink the chip. Coordinate data on each layer is multiplied by the specified *factor*. The *factor* must be a positive number.

**Caution**

If you apply a magnification factor, objects may move up to 1 dbu due to grid snapping. This can affect the results from geometrical checks.

- **-flip {x | y}**

Optional argument that flips the chip across the x-axis or y-axis.

- **-level *value***

Optional argument set that specifies the vertical level or tier number to which the placement belongs, where the *value* is an integer greater than 0. This option enables the generation of a cross-section assembly view and is mutually exclusive with -z\_origin. See “[Assembly Views](#)” on page 35 for details.

If there is more than one placement on the same level, then the thickness of the level is the maximum of all placements. For example, if you have an interposer with a stack of two ram dies on top and a single controller die also on top of the interposer, the following commands correctly specify the level value:

```
place_chip -placement int_place -chip interposer -level 1
place_chip -placement cont_place -chip controller -level 2
place_chip -placement ram1_place -chip ram -level 2
place_chip -placement ram2_place -chip ram -level 3
```

- **-z\_origin *value***

Optional argument set that specifies the height of the die from the bottom of the stack, where the *value* is a distance in microns. This option enables the generation of a cross-section assembly view and is mutually exclusive with -z\_origin. See “[Assembly Views](#)” on page 35 for details.

## Description

Defines the placement name, orientation, and scaling factor for a chip.

If **anchor\_placement\_name** references a placement that is defined using [place\\_chip](#), the value must be specified in the following format:

```
<placement_name>_<layer>
```

If **anchor\_placement\_name** references a placement defined using [place\\_layer](#), the *placement name* is specified directly. Refer to the [anchor](#) command for a detailed description of its usage.

## Examples

```
place_chip -placement place_base_bottom_interface_shapes -flip x \
-chip chip1 -x_origin 0 -y_origin 0

place_chip -placement i -chip interposer -x_origin 0.0 -y_origin 0.0

place_chip -placement m -chip memory -anchor_placement i_tsv \
-anchor_from anchor_mem -anchor_to interposer_mem -flip x

place_chip -placement l -chip logic -anchor_placement i_tsv \
-anchor_from anchor_logic -anchor_to interposer_logic -flip x
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

# place\_layer

Defines the placement name, orientation, and scaling factor for a layer.

## Usage

```
place_layer -placement placement_name
 {-layer layer_name { {-x_origin x_offset -y_origin y_offset}
 | {-anchor_placement anchor_placement_name -anchor_from from_name
 -anchor_to to_name} } | -svrf {'svrf_operations'} [-show] }
 [-rotate angle] [-magnify factor] [-flip {x | y}] [-level value] [-z_origin value]
```

## Arguments

- **-placement *placement\_name***

Required argument and value set that specifies a unique placement name that can be referenced with other commands.

- **-layer *layer\_name***

Argument and value set that specifies the name of a layer. You must specify this argument or **-svrf**.

- **-svrf {'svrf\_operations'}**

Argument and value set that specifies a set of SVRF operations to derive an output layer. You must specify this argument or **-layer**. The operations define an output layer or a merge of output layers. Any placement layers defined by either place\_chip or place\_layer that use the **-layer** option can be used in the SVRF operations.

The **-svrf** operation adds a new layout to your design that uses the name specified in the **-placement** option. The **-svrf** operation also adds a new layer to the design with the name *placement-name\_placement-name*. For example, if the placement name is “derived1” then the generated layer name is “derived1\_derived1”. Finally, a layer placement of the new derived layer is placed at the origin (0,0).

See the “Examples” section for more details.

- **-show**

Optional argument that writes derived layers to the generated assembly view of your stack. If this option is not specified, no derived layers are included in the assembly layout view. This option must be specified with the **-svrf** argument set.

- **-x\_origin *x\_offset***

Required argument and value set that specifies the offset from the origin along the x-axis. The value of **x\_offset** is in microns. This argument set only applies to **-layer**.

- **-y\_origin *y\_offset***

Required argument and value set that specifies the offset from the origin along the y-axis. The value of **y\_offset** is in microns. This argument set only applies to **-layer**.

- **-anchor\_placement anchor\_placement\_name -anchor\_from from\_name -anchor\_to to\_name**

Argument and value set that aligns the locations of *from\_name* and *to\_name* (relative to the origins of their respective chip names) at the location of the specified *anchor\_placement\_name*. Mutually exclusive with **x\_origin** and **y\_origin**. See the description section under the [anchor](#) command for an example. This argument set only applies to **-layer**.

- **-rotate angle**

Optional argument and value set that specifies a rotation *angle* in degrees, which must be either 0 or a multiple of 90 (positive or negative). The layer is rotated about the origin (after **x\_offset** and **y\_offset** values are applied) by the specified angle. Positive angle values yield a counter-clockwise rotation; negative angle values yield a clockwise rotation.

- **-magnify factor**

Optional argument and value set that specifies a magnification factor by which to expand or shrink the layer. Coordinate data on the layer is multiplied by the specified *factor*. The *factor* must be a positive number.

- **-flip {x | y}**

Optional argument that flips the layer across the x-axis or y-axis.

- **-level value**

Optional argument set that specifies the vertical level or tier number to which the placement belongs, where the *value* is an integer greater than 0. This option enables the generation of a cross-section assembly view and is mutually exclusive with **-z\_origin**. See “[Assembly Views](#)” on page 35 for details.

If there is more than one placement on the same level, then the thickness of the level is the maximum of all placements.

- **-z\_origin value**

Optional argument set that specifies the height of the layer from the bottom of the stack, where the *value* is a distance in microns. This option enables the generation of a cross-section assembly view and is mutually exclusive with **-z\_origin**. See “[Assembly Views](#)” on page 35 for details.

## Description

Defines the placement name, orientation and scaling factor for a layer.

If **anchor\_placement\_name** references a placement that is defined using [place\\_chip](#), the value must be specified in the following format:

```
<placement_name>_<layer>
```

If **anchor\_placement\_name** references a placement defined using [place\\_layer](#), the *placement name* is specified directly. Refer to the [anchor](#) command for a detailed description of its usage.

## Examples

### Example

Use the -layer argument to place the bottom interface layer at the origin and flip it over the x-axis.

```
place_layer -placement place_base_bottom_interface_shapes \
 -layer base_bottom_interface_shapes -x_origin 0 -y_origin 0 -flip x
```

### Example

Use the -svrf argument to derive a new layer by merging two existing layers and place it at the origin. The internal command uses the derived layer in a spacing rule check.

```
place_layer -placement P3_1 -svrf { P01_10 OR P02_11 }
place_layer -placement P3_2 -svrf { P11_21 OR P12_22 }

place_layer -placement P3_3 -layer 32 -x_origin 0 -y_origin 0
...
internal -check_name int_ -placement P3_2 -constraint "< 35.0"
...
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## rename\_text

Edits existing layout text in the assembly.

### Usage

**rename\_text {-chip *chip\_name* | -placement *placed\_layer*} -rename “*expression*”**

### Arguments

- **-chip *chip\_name***  
Required argument and value set that specifies a chip name.
- **-placement *placed\_layer***  
Required argument and value set that specifies a placed layer.
- **-rename “*expression*”**  
Required argument and value set that specifies a GNU regular expression that defines a pattern to be matched and replaced.

### Description

Enables editing of layout port names. The format of the *expression* follows the syntax of [Layout Rename Text](#) described in the *Standard Verification Rule Format (SVRF) Manual*. [Table B-5](#) contains a list of regular expression examples.

**Table B-5. Regular Expression Examples**

| Regular Expression | Input   | Output | Notes                                 |
|--------------------|---------|--------|---------------------------------------|
| /A/B/              | AAA     | BAA    | Replaces the first instance of A      |
| /A/B/2             | AAA     | ABA    | Replaces the second instance of A     |
| /A/B/g             | AAA     | BBB    | Global replacement.                   |
| /A.*B/R/           | ABAABBC | RC     | Longest left-most string is replaced. |
| /^VDD.*/VCC/       | VDD:1   | VCC    | ^ searches from beginning of text.    |

If the *placed\_layer* value references a placement that is defined using [place\\_chip](#), the value must be specified in the following format:

<chip\_placement\_name>\_<layer>

If the value references a placement defined using [place\\_layer](#), the placement name is specified directly.

## Examples

```
rename_text -chip chip1 -rename "/^VDD$/PWR/"
```

In this example, the first instance of the string “^VDD\$” in each name in chip1 is renamed to “PWR”.

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

# Standard Rule Syntax

The standard syntax supports the same set of rules as the extended syntax, but the options and usage of the rules is different.

Versions of the documentation prior to 2018.1 include more detailed descriptions of the options for the standard syntax. The following table includes the usage and examples for each command.

**Table B-6. Standard Syntax Rule Application**

| Command                   | Standard Syntax Rule Usage and Examples                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">centers</a>   | <pre>centers -check_name check_name     -placement1 placed_layer1 [-same_die]     [-placement2 placed_layer2] -constraint "constraint"     [-alignment {octagonal_only   orthogonal_only}]     [-overlapping] [-square] [-comment "comment"] [rve_option ...]</pre> <p>Check that pads are evenly spaced by 80 um.</p> <pre>centers -check_name controller_pads \     -placement1 \${cont}_CON_if -constraint "== 80" \     -alignment orthogonal_only -comment "Controller pads \     spaced 80 um in x or y from all pad centers"</pre> |
| <a href="#">connected</a> | <pre>connected -check_name check_name     -placement1 placed_c_layer1 [-placement2 placed_c_layer2]     [-net_mismatch {ALL   MULTI_NAME   MISMATCH   MISSING_NAME}]     [-no_dangling_ports] [-no_extra_ports] [-no_missing_ports]     [-pin_list list_of_pins]     [-text_checks {'ALL'   NONE   [type ...]}]     [-comment "comment"] [rve_option ...]</pre>                                                                                                                                                                           |
| <a href="#">copy</a>      | <pre>copy -check_name check_name -placement placement_name     [-comment "comment"]</pre> <pre>copy -check_name enc_check -placement placement1 \     -comment "EXPORT OPERATION\n Writing placement1 to DFM \     database format."</pre>                                                                                                                                                                                                                                                                                                |

**Table B-6. Standard Syntax Rule Application (cont.)**

| Command          | Standard Syntax Rule Usage and Examples                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dangling_ports   | <pre>dangling_ports -check_name <i>check_name</i>   -placements <i>placement_name_list</i>   [-comment "comment"] [rve_option ...]</pre> <p>Verify that all of the ports on the ram placement are connected to physical geometry:</p> <pre>dangling_ports -check_name ram2_pad_text_rule \   -placements p_ram_2 \   -comment "ERROR: Unconnected ports detected!"</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| dangling_no_text | <pre>dangling_no_text -check_name <i>check_name</i>   {-chips <i>chips_list</i>   {-placement1 <i>name</i> -placement2 <i>name</i>}   [-comment "comment"] [rve_option ...]}</pre> <p>Flag orphan ports that are missing text labels:</p> <pre>dangling_no_text -check_name orphans \   -chips {die1 die2} \   -comment "ERROR: Unconnected ports detected!"</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| density          | <pre>density -check_name <i>check_name</i>   -placements '{placed_layer ...}'   -constraint "<i>constraint_expression</i>"   [-expression "<i>density_expression</i>"]   [-window {wxy   wx wy} [-step {sxy   sx sy}]]   [-window_type {truncate   backup   ignore   wrap}]   [-inside {extent   placed_layer}]   [-centers value] [-comment "comment"] [rve_option ...]</pre> <pre>density -check_name density_tracer_and_interposer \   -placements {i_tsv m_mtsv_trace} -constraint "&lt; 0.1" \   -window 50 -window_type "wrap" -centers 25 -step {10 20} \   -comment "Density check for m-tracer and interposer"</pre> <pre>density -check_name density_tracer_and_interposer \   -placements {i_tsv m_mtsv_trace} -constraint "&lt; 0.1" \   -expression "(AREA(i_tsv) + AREA(m_mtsv_trace)) / AREA()" \   -window 20 -window_type "truncate" -centers 10 \   -comment "density check i_tsv m_mtsv_trace &lt; 0.1"</pre> |
| enclosure        | <pre>enclosure -check_name <i>check_name</i>   -placement1 <i>placement1</i> -placement2 <i>placement2</i>   -constraint "<i>constraint_value</i>"   [-comment "comment"] [rve_option ...]</pre> <p>Check for Through Silicon Vias that are not enclosed within metal pads by at least 0.5 um.</p> <pre>enclosure -check_name TSV_enclosure \   -placement1 pInterp_TSV -placement2 pInterp_bmetall1 \   -constraint "&lt; 0.5 REGION" -comment "TSV is less than 0.5 microns from the edge of pad on layer bmetall1"</pre>                                                                                                                                                                                                                                                                                                                                                                                                      |

**Table B-6. Standard Syntax Rule Application (cont.)**

| Command        | Standard Syntax Rule Usage and Examples                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| external       | <pre>external -check_name check_name     -placement1 placement1 [-placement2 placement2]     -constraint "constraint_value"     [-comment "comment"] [rve_option ...]</pre> <p>Check spacing between two placed layers:</p> <pre>external -check_name external_check     -placement1 p_stack_if1_stack_bot_if \     -placement2 p_base_bot_if -constraint "&lt; 0.4 REGION" \     -comment "Checking &lt; 0.4 REGION between \     layer p_stack_if1_stack_bot_if and layer p_base_bot_if"</pre> <p>Check for sufficient spacing between chip placements. Three chips are placed on an interposer using the following assembly operations:</p> <pre>place_chip -placement Controller -chip CONT -x_origin 40 \     -y_origin 40 place_chip -placement Mem_1 -chip RAM -x_origin 250 \     -y_origin 100 place_chip -placement Mem_2 -chip RAM -x_origin 280 \     -y_origin 40 -rotate 90</pre> <p>The chip instantiations can be directly referenced in a geometrical check as follows:</p> <pre>external -check_name controller_spacing_check \     -placement1 Controller -constraint "&lt; 65 REGION" \     -comment "Chip placement closer than 65 um."</pre> |
| extra_ports    | <pre>extra_ports -check_name check_name     -placements placement_name_list     [-comment "comment"] [rve_option ...]</pre> <p>Verify that the layout pads match the source netlist for one of the placements of the ram chip.</p> <pre>extra_ports -check_name ram2_pad_text_rule \     -placements p_ram_2 \     -comment "ERROR: Extra ports detected!"</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| floating_texts | <pre>floating_texts -check_name check_name     {-chips chip_name_list   -placements placement_name_list }     [-comment "comment"] [rve_option ...]</pre> <p>Verify that all text is correctly attached to geometry on the controller and interposer die.</p> <pre>floating_texts -check_name cont_int_pad_text_rule \     -chips {controller interposer} \     -comment "ERROR: Text not attached to any geometry!"</pre> <p>Verify floating text for only one of the placements of the ram chip.</p> <pre>floating_texts -check_name ram2_pad_text_rule \     -placements p_ram_2 \     -comment "ERROR: Text not attached to any geometry!"</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

**Table B-6. Standard Syntax Rule Application (cont.)**

| Command       | Standard Syntax Rule Usage and Examples                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| internal      | <pre>internal -check_name check_name           -placement1 placed_layer1 [-placement2 placed_layer2]           -constraint "constraint_value"           [-comment "comment"] [rve_option ...]</pre>                                                                                                                                                                                                                                                                                                                            |
|               | <pre>internal -check_name internal_check \           -placement1 p_stack_if1_stack_bot_if \           -placement2 p_base_bot_if \           -constraint "&lt; 0.3 REGION" \           -comment "Checking &lt; 0.3 REGION between layer \           p_stack_if1_stack_bot_if and layer p_base_bot_if"</pre>                                                                                                                                                                                                                     |
| locations     | <pre>locations -check_name check_name -placement1 placement_layer1           -placement2 placement_layer2           [-constraint "constraint_value"]           [-text_only   -overlap_only]           [-no_case] [-comment "comment"] [rve_option ...]</pre>                                                                                                                                                                                                                                                                   |
|               | <pre>locations -check_name locations_check \           -placement1 p_layout_l_pads \           -placement2 p_source_s_pads</pre>                                                                                                                                                                                                                                                                                                                                                                                               |
| missing_ports | <pre>missing_ports -check_name check_name           -placements placement_name_list           [-comment "comment"] [rve_option ...]</pre>                                                                                                                                                                                                                                                                                                                                                                                      |
|               | <p>Verify that the layout contains matching pads found in the source netlist for one of the placements of the ram chip.</p> <pre>missing_ports -check_name ram2_pad_text_rule \           -placements p_ram_2 \           -comment "ERROR: The layout is missing ports!"</pre>                                                                                                                                                                                                                                                 |
| multi_texts   | <pre>multi_texts -check_name check_name           {-chips chip_name_list   -placements placement_name_list }           [-comment "comment"] [rve_option ...]</pre>                                                                                                                                                                                                                                                                                                                                                             |
|               | <p>Verify that each pad on the controller and interposer die only contains a single text label.</p> <pre>multi_texts -check_name cont_int_pad_text_rule \           -chips {controller interposer} \           -comment "ERROR: Pads contain multiple text labels!"</pre> <p>Perform the same check on the pad text for only one of the placements of the ram chip.</p> <pre>multi_texts -check_name ram2_pad_text_rule \           -placements p_ram_2 \           -comment "ERROR: Pads contain multiple text labels!"</pre> |

**Table B-6. Standard Syntax Rule Application (cont.)**

| Command                      | Standard Syntax Rule Usage and Examples                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>no_texts</code>        | <pre><code>no_texts -check_name <i>check_name</i>   {-chips <i>chip_name_list</i>   -placements <i>placement_name_list</i> }   [-comment "comment"] [rve_option ...]</code></pre> <p>Verify that all pads are correctly labeled on the controller and interposer die.</p> <pre><code>no_texts -check_name cont_int_pad_text_rule \   -chips {controller interposer} \   -comment "ERROR: Pads are missing text labels!"</code></pre> <p>Verify pad text for only one of the placements of the ram chip.</p> <pre><code>no_texts -check_name ram2_pad_text_rule \   -placements p_ram_2 \   -comment "ERROR: Pads are missing text labels!"</code></pre>               |
| <code>offgrid_centers</code> | <pre><code>offgrid_centers -check_name <i>check_name</i>   -placement <i>placed_layer</i>   -resolution {<i>value</i>   {<i>x_value</i> <i>y_value</i>}}   [-comment "comment"] [rve_option ...]</code></pre> <p>Check that pads are aligned to a 5 micron square grid:</p> <pre><code>offgrid_centers -check_name offgrid_memory -placement m_mtsv \   -resolution 5 -comment "m_mtsv pads align to 5x5 um grid"</code></pre> <p>Check that pads are aligned to a 5 um by 10 um (x by y) grid:</p> <pre><code>offgrid_centers -check_name offgrid_memory -placement m_mtsv \   -resolution "5 10" -comment "m_mtsv pads must align to \   5x10 um grid"</code></pre> |

**Table B-6. Standard Syntax Rule Application (cont.)**

| Command                       | Standard Syntax Rule Usage and Examples                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">overlap</a>       | <pre>overlap -check_name <i>check_name</i> -placement1 <i>placed_layer1</i>           -placement2 {<i>placed_layer2</i> [<i>placed_layer2</i> ...] }           -constraint "<i>constraint_expression</i>"           [-merge] [-by_area] [-intersection] [-comment "comment"]           [rve_option ...]</pre> <p>Measure the overlap by percentage and output the intersection region.</p> <pre>overlap -check_name overlap_tracer_with_interposer \           -placement1 m_mtsv_trace -placement2 i_tsv \           -constraint "&lt;= 80.0" -intersection \           -comment "TSV and interposer pads must overlap by 80%"</pre> <p>Measure the overlap by area.</p> <pre>overlap -check_name overlap_tracer_with_interposer \           -placement1 m_mtsv_trace -placement2 i_tsv \           -constraint "&lt;= 20.0" -by_area \           -comment "Overlap not at least 20um^2 between tsv and \           interposer pads"</pre> <p>Check that all of the interposer pads have suitable overlap with all of the pads of the chips placed on the interposer.</p> <pre>overlap -check_name interposer_pads \           -placement1 p_interposer \           -placement2 {"p_controller" "p_ram_1" "p_ram_2" "p_ram_3"} \           -constraint "&lt;= 95" \           -comment "Interposer pads must overlap chip pads by 95%!"</pre> <p>Check for completely missing interposer pads. In this case, you can use an overlap percentage constraint. Any pads that are missing are reported.</p> <pre>overlap -check_name interposer_pads \           -placement1 p_interposer \           -placement2 {"p_controller" "p_ram_1" "p_ram_2" "p_ram_3"} \           -constraint "&lt; 90" \           -comment "Interposer pads must overlap chip pads"</pre> |
| <a href="#">same_size</a>     | <pre>same_size -check_name <i>check_name</i>           -placement1 <i>placed_layer</i>           -placement2 <i>placed_layer</i>           [-comment "comment"] [rve_options...]</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <a href="#">select_checks</a> | <pre>select_checks {     -check_names {'name_pattern [name_pattern ...]'}       -placements {'place_name_pattern [place_name_pattern ...]'} } [-check_types {'check_type ...'}]</pre> <p>Execute only the checks that are used in the interposer and any of the ram placements:</p> <pre>select_checks -placements {"interposer" "ram_*"}</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

**Table B-6. Standard Syntax Rule Application (cont.)**

| Command         | Standard Syntax Rule Usage and Examples                                                                                                                                                                                                                                                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| unselect_checks | <pre><code>unselect_checks {     -check_names '{' name_pattern [name_pattern ...] '}',       -placements '{' place_name_pattern[place_name_pattern ...] '}', }     [-check_types '{' check_type ...'}' ]</code></pre> <p>Exclude checks that are used in the interposer and ram placements:</p> <pre><code>unselect_checks -placements {"interposer" "ram_*"}</code></pre> |

## **set\_auto\_rve\_show\_layers**

Specifies for all rule checks (except [tvf\\_block](#)) whether Calibre RVE layer highlights show only layers used to derive the rule check.

### **Usage**

```
set_auto_rve_show_layers {NO | YES }
```

### **Arguments**

- **NO**

Specifies not to apply -set\_rve\_show\_layers AUTO to any checks that do not have an existing Calibre RVE layer specification. This is the default.

- **YES**

Applies -set\_rve\_show\_layers AUTO to all rule checks (except [tvf\\_block](#)) that do not have an existing Calibre RVE layer specification.

### **Description**

Use this optional rule file command to control the “-set\_rve\_show\_layers {layer\_list | AUTO}” option for all rule checks (except [tvf\\_block](#)). When this command is specified with YES in the Calibre 3DSTACK rule file, then -set\_rve\_show\_layers AUTO is applied to all rule checks that do not already specify the -set\_rve\_show\_layers option. The AUTO keyword instructs Calibre RVE to only show the input layers that were used to derive the rule checks.

---

#### **Note**

 In order to use this functionality, you must enable the Calibre RVE option “Only show check-dependent layers while highlighting results (hides other layers)” on the **Setup > Options > Highlighting** pane in the DRC/DFM Highlighting area. See “[RVE Show Layers](#)” in the *Calibre RVE User’s Manual* for complete details.

---

### **Examples**

```
set_auto_rve_show_layers YES
```

### **Related Topics**

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

## **set\_rve\_cto\_file**

Specifies the path to a check text override file that controls how results are highlighted in Calibre RVE.

### **Usage**

**set\_rve\_cto\_file -file *cto\_file***

### **Arguments**

- **-file *cto\_file***

Required argument and value that specifies the path to a check text override file.

### **Description**

Use this optional rule file command to specify the path to a check text override (CTO) file. The *cto\_file* contains DRC RVE check text comments that control how results are highlighted in Calibre RVE. Only one CTO file is allowed.

---

#### **Note**

 The -set\_rve\_\* arguments in your Calibre 3DSTACK rule file take precedence over Calibre RVE options in your CTO file specified for the same rules.

---

The following CTO file specifies two check text comments for a rule named pads\_on\_grid:

```
DRC RVE check text override file for a Calibre 3DSTACK rule file
#
pads_on_grid
RVE Highlight color: red
RVE Show Layers: backside_rdl
```

For complete information on the CTO file and how to load it, see “[DRC RVE Check Text Override File \(CTO File\)](#)” in the *Calibre RVE User’s Manual*.

---

#### **Caution**

 Calibre RVE searches your working directory and loads a CTO file named *3dstack.rdb.cto* if it exists. Do not use **set\_rve\_cto\_file** to specify this file if it exists in your working directory as it is automatically loaded. You may use *3dstack.rdb.cto* as the CTO filename if it exists outside of your working directory.

---

### **Examples**

```
set_rve_cto_file -file ./custom_rve_settings.cto
```

### **Related Topics**

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

**Rule Check Commands**

# tvf\_block

Used only in the standard Calibre 3DSTACK syntax. See the [custom\\_check](#) command for the Extended 3DSTACK+ syntax.

Defines custom checks for the Calibre 3DSTACK rule file.

## Usage

### Standard Calibre 3DSTACK Syntax

```
tvf_block name '{' body '}' [-export_layers placement_list] [-attach_to_placement placement]
```

## Arguments

- ***name***  
Required argument that specifies the name of the TVF block.
- **'{ *body* '}**  
Required body of TVF code that defines statements for the creation of derived layers.  
Enclosing the TVF code in braces, { }, is required.
- **-export\_layers *placement\_list***  
Optional argument and value set that defines the list of resulting placements that can be used in rule checks.
- **-attach\_to\_placement *placement***  
Optional argument and value set that specifies the placement to which the resulting placements should be attached. If this option is specified and the derived layer is used in geometrical rule checks, then any resulting errors that are applied on this derived layer are reported for the specified *placement*.

## Description

Use this command to define custom checks in the Calibre 3DSTACK rule file. The **tvf\_block** command provides the capability to define derived layers with the TVF language. The layers can then be used for [centers](#), [density](#), [enclosure](#), [external](#), [internal](#), [offgrid\\_centers](#), and [overlap](#) checks. Placements generated from the **tvf\_block** command cannot be used as text placements in the `attach_text` command.

## Syntax Debugging Tips

If you introduce a syntax error inside a `tvf_block` statement, Calibre 3DSTACK reports the error on a line number in an SVRF rule file generated at runtime. Calibre 3DSTACK saves this intermediate SVRF rule file to your run directory with the name `3dstack_deck.svrf.enc`. You can use this file to determine the cause of the syntax error.

The `tvf_block` statements appear in the middle of two encrypted sections of the SVRF file and are preceded with comments that include the originating line number of the statement in your

rule file. This allows you to correlate the runtime syntax error to statements in your rules. The following is an example excerpt from the *3dstack\_deck.svrf.enc* file:

```
' _+^+>/Z6\\^C=A!=KZW%!!! "K_W$)SZE79#S^=Y<%P\!RQ!!"
#ENDCRYPT

/// rules/3dstack.rules:228 - tvf_block Not_COIN1
out_layer = (pInterp_INTERP_FRONT_if XOR pCont_CONTROLLER_if) TOUCH
pCont_CONTROLLER_if

/// rules/3dstack.rules:236 - tvf_block ici
ici {
OR pInterp_INTERP_FRONT_if pCont_CONTROLLER_if
}

#DECRYPT %;5~GPPB_R"BV8-)B?:CAD5^1QP.BHCG?-B>H\K$0B_CZ-
!#ONK0EAD%I$$DHNLV4FF]A!!"
```

## Examples

```
tvf_block creating_logic_to_memory_connects { \
 tvf::SETLAYER l_logic_ltsv_to_memory = l_ltsv NOT i_out_M1_tsv; \
} -export_layers [list l_logic_ltsv_to_memory] -attach_to_placement l
```

## Related Topics

[System and Miscellaneous Commands](#)

[Assembly Commands](#)

[Rule Check Commands](#)

# Appendix C

## Error and Warning Messages

---

Error and warning messages are generated by the Calibre 3DSTACK tool at run time.

If you receive a general error, such as a command not found, but the command is documented, check to make sure the file is formatted for Unix. You can run dos2unix to convert a file from Windows format to Unix.

A frequent error you may encounter is the following:

```
ERROR: Error SYN1 on line <number> of SVRF generated from TVF ...
```

This message originates from the rule file parser and can be caused by items such as the following:

- A space after a line continuation character “\”. For example:

```
die -name mem1 \
 -layout {...} \whitespace_here
 -layer_info {...} \
 ...
 ...
```

- A hyphen “-” instead of an underscore “\_” in layer names or types. In the following example, “die-bump” produces an error because hyphens are not allowed in SVRF layer names. The “bump\_type” type is correct because underscores are permitted.

```
die -name mem1 \
 -layer_info {
 -name die-bump
 -type bump_type
 ...
 } \
 ...
 ...
```

---

### Tip

 You can optionally create additional information about Tcl errors generated from your Calibre 3DSTACK rule files by setting the CALIBRE\_ENABLE\_3DSTACKPLUS\_DECK\_DUMP environment variable to a non-null value. This option will export a file named <rule\_file>\_exported.3dstack+ that enables you to debug the source of Tcl errors in your rules.

---

|                               |            |
|-------------------------------|------------|
| <b>Error Messages .....</b>   | <b>438</b> |
| <b>Warning Messages .....</b> | <b>447</b> |

## Error Messages

Error messages must be corrected to continue a Calibre 3DSTACK run.

Error messages take the following form:

`3DSTACK_ERROR_number: error_message`

---

### Note

---

 If you encounter a layout peek error, try running Calibre 3DSTACK on a machine with more memory.

---

**Table C-1. Calibre 3DSTACK Error Messages**

| Number            | Description                                                         | Possible Causes                                                                                                                                                                                                                         |
|-------------------|---------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_100 | Unable to find the executable file, please check your installation. | Invalid Calibre installation.                                                                                                                                                                                                           |
| 3DSTACK_ERROR_101 | Unable to find layout file.                                         | The value of the -path argument for the die <code>-layout</code> command is incorrect.                                                                                                                                                  |
| 3DSTACK_ERROR_102 | Unable to find 3DSTACK file.                                        | The path to the <code>rule_file_name</code> is incorrect for the 3DSTACK run.                                                                                                                                                           |
| 3DSTACK_ERROR_103 | Unable to find rule file.                                           | The value of the -rule_deck argument for the <code>run</code> command is incorrect.                                                                                                                                                     |
| 3DSTACK_ERROR_104 | Unable to find NetList file.                                        | The value of the -file argument for a <code>source_netlist</code> command is incorrect.                                                                                                                                                 |
| 3DSTACK_ERROR_105 | Unable to find cell.                                                | The value specified for the -primary argument in a die <code>-layout</code> command does not match the top cell name for the design.                                                                                                    |
| 3DSTACK_ERROR_106 | Unable to find layer or placement.                                  | This error is caused by a reference to an invalid layer or placement name.                                                                                                                                                              |
| 3DSTACK_ERROR_107 | Unable to find anchor.                                              | This error is caused by a reference to an invalid anchor name. Ensure that the value of the -name argument for each <code>stack</code> anchor is correct, and that the values of the -anchor_from and -anchor_to arguments are correct. |
| 3DSTACK_ERROR_108 | No placement definition found.                                      | The rule file does not contain any placement operations in the <code>stack</code> command.                                                                                                                                              |
| 3DSTACK_ERROR_109 | Unable to find import file.                                         |                                                                                                                                                                                                                                         |

**Table C-1. Calibre 3DSTACK Error Messages (cont.)**

| <b>Number</b>     | <b>Description</b>                                                                                                    | <b>Possible Causes</b>                                                                                                                                                                                                                                        |
|-------------------|-----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_110 | Unable to find the 3DSTACK assembly file.                                                                             | Assembly file specified with the -use_assembly invocation argument cannot be found.                                                                                                                                                                           |
| 3DSTACK_ERROR_111 | Unknown extension for the 3DSTACK assembly file.                                                                      |                                                                                                                                                                                                                                                               |
| 3DSTACK_ERROR_112 | Unable to find <i>type</i> file <i>file_name</i>                                                                      |                                                                                                                                                                                                                                                               |
| 3DSTACK_ERROR_117 | The -overlapping flag of centers command can be applied only when -placement2 is specified!                           | The -overlapping option of the <a href="#">centers</a> command checks the overlap between pads of two placements. You must specify a second placement for this option to be used.                                                                             |
| 3DSTACK_ERROR_118 | There are text renaming rule(s) for placement(s) <i>placement_list</i> . Chip based text check reporting is disabled! | By default, text errors are reported by chip instead of by placement. If this warning is issued, the text reporting is performed for each placement in the connectivity stack because there is a <a href="#">die -rename_text</a> operation on the placement. |
| 3DSTACK_ERROR_200 | The displayed argument is a mandatory option for the displayed command.                                               | The displayed command is missing a mandatory argument.                                                                                                                                                                                                        |
| 3DSTACK_ERROR_201 | The displayed value is invalid for the displayed option.                                                              | The displayed value is invalid; replace it with a value from the displayed list.                                                                                                                                                                              |
| 3DSTACK_ERROR_202 | The displayed option is invalid for the displayed command.                                                            |                                                                                                                                                                                                                                                               |
| 3DSTACK_ERROR_203 | Wrong number of arguments.                                                                                            | The number of specified arguments does not match the number of expected arguments.                                                                                                                                                                            |
| 3DSTACK_ERROR_204 | Wrong argument.                                                                                                       | The name of the specified argument does not match the name of the expected argument.                                                                                                                                                                          |
| 3DSTACK_ERROR_205 | Incorrect input argument.                                                                                             | The name of the specified input argument does not match the name of the expected argument.                                                                                                                                                                    |

**Table C-1. Calibre 3DSTACK Error Messages (cont.)**

| Number            | Description                                                                                                                                                    | Possible Causes                                                                                                                                                                                         |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_206 | Unknown tag. Please specify a valid tag from the following list.                                                                                               | The displayed option is invalid; replace it with an option from the displayed list.                                                                                                                     |
| 3DSTACK_ERROR_207 | Expecting a non-zero integer number of the displayed argument value.                                                                                           | The syntax of the input file for <a href="#">die -import_text_labels</a> option is incorrect.                                                                                                           |
| 3DSTACK_ERROR_208 | Expecting keyword <code>-export_layers</code> after check body.                                                                                                | The <code>-export_layers</code> keyword must appear after each check body. Refer to <a href="#">Standard (Legacy) Syntax Example 1</a> .                                                                |
| 3DSTACK_ERROR_209 | Expecting a value of the displayed type for the displayed parameter.                                                                                           | The datatype specified for the displayed parameter is incorrect.                                                                                                                                        |
| 3DSTACK_ERROR_210 | Failed to import the netlist.                                                                                                                                  | The input netlist was not specified correctly.<br><br>Check the config <a href="#">-source_netlist</a> command in your rule deck, or the <b>Inputs &gt; Netlist</b> tab in Calibre Interactive 3DSTACK. |
| 3DSTACK_ERROR_211 | Failed to export the netlist.                                                                                                                                  | Writing of the netlist failed.                                                                                                                                                                          |
| 3DSTACK_ERROR_212 | <code>-x_origin</code> , <code>-y_origin</code> and anchor definition are mutually exclusive options.                                                          | The syntax for <a href="#">place_chip</a> or <a href="#">place_layer</a> is incorrect.                                                                                                                  |
| 3DSTACK_ERROR_213 | Expecting: check/<br><code>tvf_block</code> name body<br><code>-export_layers</code> <i>layer_list</i> [ <code>-attach_to_placement</code> <i>placement</i> ]. | The syntax of the rule check is incorrect. Refer to <a href="#">Standard (Legacy) Syntax Example 1</a> .                                                                                                |
| 3DSTACK_ERROR_215 | A required command is missing from the argument list.                                                                                                          |                                                                                                                                                                                                         |
| 3DSTACK_ERROR_216 | Unable to read the netlist.                                                                                                                                    |                                                                                                                                                                                                         |
| 3DSTACK_ERROR_217 | Unable to compile the specified custom check.                                                                                                                  |                                                                                                                                                                                                         |
| 3DSTACK_ERROR_218 | Filtering expression is invalid.                                                                                                                               |                                                                                                                                                                                                         |

**Table C-1. Calibre 3DSTACK Error Messages (cont.)**

| <b>Number</b>     | <b>Description</b>                                                                                           | <b>Possible Causes</b>                                                                                                                                                                                                                                                                       |
|-------------------|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_219 | Identical placements are illegally specified in the connected command.                                       | Identical placements are specified in the <a href="#">connected</a> check command. Correct your rule file to remove the identical instances for this check.                                                                                                                                  |
| 3DSTACK_ERROR_220 | Invalid value <i>value</i> for option <i>option_name</i> . Should not be a 3dstack rule file format keyword. | The highlighted rule check uses a name that is a protected Calibre 3DSTACK file format keyword. Change the name of the rule check.                                                                                                                                                           |
| 3DSTACK_ERROR_221 | <i>placement1</i> and <i>placement2</i> arguments both should be either layer placements or chip placements. | You cannot specify a layer placement and a chip placement in a single geometrical check. The two placement arguments in the command must be of the same type. See <a href="#">enclosure</a> , <a href="#">external</a> , or <a href="#">internal</a> for more details on the correct syntax. |
| 3DSTACK_ERROR_300 | Net, pin or placement already exists in the displayed cell.                                                  |                                                                                                                                                                                                                                                                                              |
| 3DSTACK_ERROR_301 | Layer or placement with the displayed name already exists.                                                   |                                                                                                                                                                                                                                                                                              |
| 3DSTACK_ERROR_302 | Layout with the displayed name already declared.                                                             | There is more than one die <a href="#">-layout</a> command with the same <a href="#">-chip_name</a> value.                                                                                                                                                                                   |
| 3DSTACK_ERROR_303 | Anchor with the displayed name already declared in layout.                                                   | Duplicate <a href="#">anchor</a> -name value for the same chip.                                                                                                                                                                                                                              |
| 3DSTACK_ERROR_304 | The displayed placement already has text-placement attached.                                                 | There is more than one <a href="#">attach_text</a> command with the same <a href="#">-placement</a> value.                                                                                                                                                                                   |
| 3DSTACK_ERROR_305 | Only single occurrences of source_netlist statement are allowed.                                             | There is more than one instance of the config <a href="#">-source_netlist</a> command.                                                                                                                                                                                                       |
| 3DSTACK_ERROR_306 | There are multiple text-labels overlapping with the pads on the displayed layer.                             |                                                                                                                                                                                                                                                                                              |

**Table C-1. Calibre 3DSTACK Error Messages (cont.)**

| Number            | Description                                                                                                     | Possible Causes                                                                                                                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_307 | Multiple definitions of the same placed layer are not allowed.                                                  |                                                                                                                                                                                                                                                                                 |
| 3DSTACK_ERROR_308 | A layout filename already exists that is the same as the filename specified in export_template.                 | The same filename or file path has been specified in the <a href="#">export_template</a> and die <a href="#">-layout</a> commands. Modify one of the commands accordingly. The export_template command is located under <b>Outputs &gt; Export Data</b> in Calibre Interactive. |
| 3DSTACK_ERROR_309 | An undefined check name is specified in the select_checks command.                                              |                                                                                                                                                                                                                                                                                 |
| 3DSTACK_ERROR_310 | The placement already has a specified tracer placement.                                                         |                                                                                                                                                                                                                                                                                 |
| 3DSTACK_ERROR_311 | Placement \$placement_name specified in connected command must be part of a connectivity stack.                 | Placements in a <a href="#">connected</a> check need to have corresponding <a href="#">connect (Standard Syntax)</a> statements.                                                                                                                                                |
| 3DSTACK_ERROR_316 | Source filtering is already defined for the listed items.                                                       | The <a href="#">source_filter</a> command has been applied multiple times to the same components. Correct your source_filter command.                                                                                                                                           |
| 3DSTACK_ERROR_317 | The specified placement should have connectivity information attached in order for the connected check to work. | A placement in your design is missing connectivity information. The <a href="#">connected</a> operation cannot be performed correctly. Verify your <a href="#">connect (Standard Syntax)</a> operations for the specified placement.                                            |
| 3DSTACK_ERROR_318 | The derived placement cannot be used as text placements for an attach_text command.                             | A placement exported from a <a href="#">tvf_block</a> operation has been incorrectly used in an <a href="#">attach_text</a> operation.                                                                                                                                          |
| 3DSTACK_ERROR_319 | Cannot retrieve the specified property from the specified layer.                                                |                                                                                                                                                                                                                                                                                 |

**Table C-1. Calibre 3DSTACK Error Messages (cont.)**

| <b>Number</b>     | <b>Description</b>                                                              | <b>Possible Causes</b>                                                                                                                                                                                                     |
|-------------------|---------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_400 | Missing cell or placement.                                                      | This error can be issued if your design is missing a placement used in a <a href="#">connected</a> rule check. Cross check any placement and connect assembly operations with the connected checks used in your rule file. |
| 3DSTACK_ERROR_401 | The displayed cell already exists in the connectivity database.                 |                                                                                                                                                                                                                            |
| 3DSTACK_ERROR_402 | Incomplete definition of check. Connect check should be defined for two layers. |                                                                                                                                                                                                                            |
| 3DSTACK_ERROR_403 | Connect check prerequisites not met. Please extract connectivity first.         |                                                                                                                                                                                                                            |
| 3DSTACK_ERROR_404 | Prototype cell mismatch.                                                        | A topcell name from a source netlist does not match the value of the die <a href="#">-layout</a> -primary option.                                                                                                          |
| 3DSTACK_ERROR_406 | Connectivity check error.                                                       | The config <a href="#">-source_netlist</a> input file is invalid.                                                                                                                                                          |
| 3DSTACK_ERROR_407 | The displayed pin is connected to different nets.                               |                                                                                                                                                                                                                            |
| 3DSTACK_ERROR_500 | The displayed application is not executable. Please check your installation.    | Check your Calibre installation.                                                                                                                                                                                           |
| 3DSTACK_ERROR_501 | Unable to read directory.                                                       | There was a problem accessing a temporary directory.<br>Check <code>/tmp</code> , <code>/var/tmp</code> , or <code>/usr/tmp</code> .                                                                                       |
| 3DSTACK_ERROR_502 | Given directory is not writable.                                                | Check the permissions on the directory specified with the “ <a href="#">run -directory</a> ” option.                                                                                                                       |
| 3DSTACK_ERROR_503 | The displayed value is a file, not a directory.                                 | Ensure that the “ <a href="#">run -directory</a> ” option points to a directory.                                                                                                                                           |
| 3DSTACK_ERROR_504 | The displayed value is a directory, not a file.                                 | Ensure that the “ <a href="#">run -directory</a> ” option points to a file.                                                                                                                                                |

**Table C-1. Calibre 3DSTACK Error Messages (cont.)**

| Number            | Description                                                                                            | Possible Causes                                                                                                                                      |
|-------------------|--------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_505 | The displayed file is not writable.                                                                    |                                                                                                                                                      |
| 3DSTACK_ERROR_506 | The displayed file is not readable.                                                                    | Verify the location and contents of the file.                                                                                                        |
| 3DSTACK_ERROR_600 | Please provide a version number by invoking <code>set_version</code> command.                          | A <code>set_version</code> command is required. The version must be 1.0.                                                                             |
| 3DSTACK_ERROR_601 | Negative magnify factor, please specify a positive number.                                             | The <code>-magnify</code> value for <code>place_chip</code> and <code>place_layer</code> must be a positive number.                                  |
| 3DSTACK_ERROR_602 | Only orthogonal rotations are allowed in <code>place_chip</code> or <code>place_layer</code> commands. | The <code>-rotate</code> value for <code>place_chip</code> and <code>place_layer</code> must be either 0 or a multiple of 90 (positive or negative). |
| 3DSTACK_ERROR_603 | No layers are defined for the displayed layout.                                                        | No <code>layer</code> statements exist for the displayed chip name.                                                                                  |
| 3DSTACK_ERROR_604 | Poorly formed anchor definition for the displayed placement.                                           | The syntax for the anchor definition in a <code>place_chip</code> or <code>place_layer</code> command is incorrect.                                  |
| 3DSTACK_ERROR_605 | The Calibre run failed. Refer to the run log for further details.                                      | One possible cause is the value of the <code>-directory</code> argument for a <code>run</code> command is incorrect.                                 |
| 3DSTACK_ERROR_606 | No <code>txt-placement</code> is attached to placement <code>placement_layer</code> .                  | An <code>attach_text</code> command is missing for the displayed placement.                                                                          |
| 3DSTACK_ERROR_607 | Syntax error.                                                                                          |                                                                                                                                                      |

**Table C-1. Calibre 3DSTACK Error Messages (cont.)**

| <b>Number</b>     | <b>Description</b>                                                 | <b>Possible Causes</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------------|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_608 | <error_message>                                                    | <p>This error number is used in multiple cases; the message describes the specific case, if possible.</p> <p>One such case that this error can occur is when an input file is not correctly formatted. For example, if you created a file in Windows, it may sometimes include the wrong line encoding (carriage returns instead of line feeds). This can trigger an error such as this:</p> <pre>3DSTACK_ERROR_608: in file file: on line line: invalid command name "command"</pre> <p>The solution for this error is to run dos2unix on the file.</p> <p>Another error is the following:</p> <pre>The name primary is already defined for lefdef in die name.</pre> <p>This is caused by more than one LEF/DEF inputs having the same top cell name defined in the rule file.</p> <p>This error can also be generated from an error condition in Calibre DESIGNrev. If the error is a layout peek error, then try running the design on a machine with more memory.</p> |
| 3DSTACK_ERROR_609 | Syntax Error                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 3DSTACK_ERROR_610 | Unknown Version.                                                   | The version you are currently using is not supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 3DSTACK_ERROR_611 | The check source operation can only be applied for SPICE netlists. | Your config -source_netlist is not SPICE formatted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 3DSTACK_ERROR_700 | The specified data does not exist.                                 | The Calibre 3DSTACK database is missing data. This error is issued in Calibre YieldServer when a handle to a non-existent netlist database object is requested with the 3dstack::get_netlist command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

**Table C-1. Calibre 3DSTACK Error Messages (cont.)**

| Number             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                            | Possible Causes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_ERROR_701  | The specified data already exists.                                                                                                                                                                                                                                                                                                                                                                                                                     | This is issued in Calibre YieldServer if a netlist database object in the Calibre 3DSTACK DFM database already exists with the same name.                                                                                                                                                                                                                                                                                                                                                                  |
| 3DSTACK_ERROR_702  | Missing a prototype cell. The database may be corrupted.                                                                                                                                                                                                                                                                                                                                                                                               | This is issued in Calibre YieldServer if the Calibre 3DSTACK DFM database may have become corrupted.                                                                                                                                                                                                                                                                                                                                                                                                       |
| 3DSTACK_ERROR_1106 | Unable to find package die with the name <package-name>                                                                                                                                                                                                                                                                                                                                                                                                | You specified a package with the -pkg option that does not exist in your design. See <a href="#">config</a> for the extended syntax rules or <a href="#">export_connectivity</a> for 3DSTACK rules.                                                                                                                                                                                                                                                                                                        |
| 3DSTACK_ERROR_1205 | Wrong input, expecting only one interacting pair for RC rule file generation.                                                                                                                                                                                                                                                                                                                                                                          | Use additional export_layout commands with different die names to specify interactions between more than one pair of dies.                                                                                                                                                                                                                                                                                                                                                                                 |
| 3DSTACK_ERROR_1400 | One of the following:<br>Missing section DECLARATION in iRCX file <i>ircx_file</i><br>Missing object DECLARATION in DECLARATION section of iRCX file <i>ircx_file</i><br>Missing SECTION LAYER MAPPING in DECLARATION object of DECLARATION section of iRCX file <i>ircx_file</i><br>Missing section \${layerMappingSection} in iRCX file <i>ircx_file</i><br>Missing layer \${layerNumber} in the layer mapping section of iRCX file <i>ircx_file</i> | One or more of the following issues with iRCX file specified using the <a href="#">-layer_info -ircx</a> option: <ul style="list-style-type: none"> <li>• There is no DECLARATION section.</li> <li>• There is no DECLARATION object in the DECLARATION section.</li> <li>• There is no SECTION[LAYER MAPPING] in the DECLARATION object.</li> <li>• There is no LAYER MAPPING section in the DECLARATION object.</li> <li>• There is a missing layer number and datatype in the layer mapping.</li> </ul> |

# Warning Messages

Warning messages should be reviewed to determine if they indicate a real problem in your design. Calibre 3DSTACK writes all warning messages issued during a run to your working directory in a file named *3dstack.warnings*.

Warning messages take the following form:

`3DSTACK_WARNING_number: warning_message`

**Table C-2. Calibre 3DSTACK Warning Messages**

| Number              | Description                                                                                                                                                                 | Possible Causes                                                                                                                                                                                                                                                                                                                        |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_WARNING_100 | Unable to find specified argument.                                                                                                                                          | An incorrect argument may be specified.                                                                                                                                                                                                                                                                                                |
| 3DSTACK_WARNING_101 | No connectivity is defined for the placement <i>placement_name</i> . Should be part of connectivity stack or selected connected statement. No text attachment is performed. | In order for text tracing and attach operations to occur, the placement must have its connectivity defined. Refer to the <a href="#">connected</a> and <a href="#">attach_text</a> commands for examples.                                                                                                                              |
| 3DSTACK_WARNING_102 | No checks are selected or defined. Only the 3D assembly generation is performed.                                                                                            | This message occurs when you have invoked Calibre 3DSTACK with the <code>-create_assembly <i>assembly_name</i></code> invocation argument. This limits a run to only generate an assembly view file. See “ <a href="#">Calibre 3DSTACK Invocation</a> ” and “ <a href="#">Specifying Assembly Instructions</a> ” for complete details. |

**Table C-2. Calibre 3DSTACK Warning Messages (cont.)**

| Number              | Description                                                                                                                                                                 | Possible Causes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_WARNING_103 | Unable to find placement <i>placement</i> in <i>source_netlist</i> .                                                                                                        | <p>For the standard syntax, the assembly operations (see the map_placement, source_netlist, and layout commands) may not be correctly defined for the object specified in the warning message.</p> <p>For the extended syntax, apply the -source option when specifying dies with the <b>stack</b> command. This option allows you to correspond the chip in the assembly to the instance in your source netlist.</p> <p> <b>Note:</b> If your source netlist is hierarchical, then make sure you check the following items:</p> <ul style="list-style-type: none"> <li>• The -hier option is specified in the config -source command.</li> <li>• The -interposer option is enabled for the interposer die.</li> <li>• The -source option in the stack command contains the full path to the instantiated circuit. For example:</li> </ul> <pre>-source pInterp/pRam0</pre> |
| 3DSTACK_WARNING_104 | Overriding mapping.                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 3DSTACK_WARNING_105 | Multiple pads from tracer layer <i>layer</i> are connected to the same net despite of having different texts attached. Connectivity checking may produce incorrect results. | You may have attached text to two or more pads erroneously. This can produce connectivity errors. Verify any text attach operations.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 3DSTACK_WARNING_106 | The <b>check</b> statement will not be supported in the next version of the 3DSTACK rule file. Please use <b>tvf_block</b> .                                                | The <b>check</b> command is deprecated. You are most likely using an old 3DSTACK rule file in a newer version of Calibre 3DSTACK. Replace all <b>check</b> statements with corresponding <b>tvf_block</b> commands.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

**Table C-2. Calibre 3DSTACK Warning Messages (cont.)**

| Number              | Description                                                                                                                                         | Possible Causes                                                                                                                                                                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_WARNING_107 | source_filter statement(s) can be applied only for source netlist files of flat SPICE format. Ignoring all occurrences for source_filter statement. | The <a href="#">source_filter</a> command can only be used if your config -source_netlist file is formatted in SPICE and the -hier option is not used in the source_netlist command.                                                                  |
| 3DSTACK_WARNING_108 | No matching renaming rules are found for placements or chips. Text renaming is not applied.                                                         | Your <a href="#">rename_text</a> commands may contain errors.                                                                                                                                                                                         |
| 3DSTACK_WARNING_109 | No source_netlist statement is defined in rule file. Nothing to do.                                                                                 | You may have invoked Calibre 3DSTACK with the -cs option, but did not specify a config -source_netlist to verify. See “ <a href="#">Calibre 3DSTACK Invocation</a> ” on page 20 for more details on the -cs option and how to specify a netlist file. |
| 3DSTACK_WARNING_110 | Unable to find chip <i>chip_name</i> in the source netlist. Filtering for the specified chip will not be applied.                                   | The config -source_netlist file does not contain a chip specified in the <a href="#">source_filter</a> command.                                                                                                                                       |
| 3DSTACK_WARNING_111 | The pin <i>pin_name</i> specified in the source_filter command does not exist in the given subcircuit <i>subcircuit_name</i> .                      | You may have incorrectly specified a pin or subcircuit in the <a href="#">source_filter</a> command.                                                                                                                                                  |
| 3DSTACK_WARNING_112 | All nets connected to single pin instances are ignored.                                                                                             |                                                                                                                                                                                                                                                       |
| 3DSTACK_WARNING_113 | Failed to set RVE Link.<br>Reason: <i>text</i>                                                                                                      |                                                                                                                                                                                                                                                       |
| 3DSTACK_WARNING_114 | -set_rve_highlight_index cannot be specified with -set_rve_highlight_color. The latter is ignored.                                                  | You have specified two conflicting Calibre RVE options. Remove one of the options from the check.                                                                                                                                                     |

**Table C-2. Calibre 3DSTACK Warning Messages (cont.)**

| Number              | Description                                                                                                | Possible Causes                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_WARNING_115 | -anchor_tag and -display_text options can be used only with -set_rve_link                                  | The -anchor_tag and -display_text arguments for Calibre RVE check text override commands can only be specified with -set_rve_link. They are not standalone arguments.                                                                                                                                                                                                         |
| 3DSTACK_WARNING_116 | -precision option is only needed for layer derivation, otherwise - it will be ignored                      | The -precision argument for the die <a href="#">-layout</a> rule file command is only required when specifying layer derivation with the -svrf argument to the <a href="#">layer</a> command and if the precision differs from the default of 1000. When layer derivation is not used, precision handling is performed automatically and the -precision argument is not used. |
| 3DSTACK_WARNING_117 | The -overlapping flag of the centers command can be applied only when -placement2 is specified.            | You must specify two placements in the <a href="#">centers</a> command when using the -overlapping option.                                                                                                                                                                                                                                                                    |
| 3DSTACK_WARNING_119 | Hierarchical source_netlist/export_connectivity is allowed for SPICE format only. -hier option is ignored. | The -hier option can only be used in the <a href="#">source_netlist</a> and <a href="#">export_connectivity</a> commands if the netlist format is SPICE.                                                                                                                                                                                                                      |
| 3DSTACK_WARNING_120 | Hierarchical connectivity export is not supported for multi interposer stacks. -hier option is ignored.    | You specified more than one interposer in your assembly (the -interposer option is enabled for two layout or two die commands). Since Calibre 3DSTACK cannot infer the top-level instance, the generated netlist is flat.                                                                                                                                                     |
| 3DSTACK_WARNING_122 | <port> port is dangling in both layout and source netlists.                                                | The tool issues this warning for all pin or port objects that are not connected in either the source or layout. You can disable this message using the config -warning_severity option.                                                                                                                                                                                       |

**Table C-2. Calibre 3DSTACK Warning Messages (cont.)**

| <b>Number</b>       | <b>Description</b>                                                                                                                                            | <b>Possible Causes</b>                                                                                                                                                                                                                          |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_WARNING_306 | Unable to find <package-name> with the name <file-name> . Ignoring...                                                                                         | You specified a package with the -pkg option that does not exist in your design. See <a href="#">config</a> for the extended syntax rules or <a href="#">export_connectivity</a> for 3DSTACK rules.                                             |
| 3DSTACK_WARNING_307 | The package die <package-name> cannot be interposer. Ignoring -pkg for <file-name>                                                                            | You specified an interposer die with the -pkg option. This is not allowed. See <a href="#">config</a> for the extended syntax rules or <a href="#">export_connectivity</a> for 3DSTACK rules.                                                   |
| 3DSTACK_WARNING_308 | There should be exactly one placement of the given package layout <package-name>. Ignoring -pkg for <file-name>                                               | You specified a package with the -pkg option that has more than one placement. There can only be one package in your design. See <a href="#">config</a> for the extended syntax rules or <a href="#">export_connectivity</a> for 3DSTACK rules. |
| 3DSTACK_WARNING_309 | No text attachments are detected for package layout <package-name>. Ignoring -pkg for <file-name>                                                             | You specified a package with the -pkg option that does not have text for connectivity. See <a href="#">config</a> for the extended syntax rules or <a href="#">export_connectivity</a> for 3DSTACK rules.                                       |
| 3DSTACK_WARNING_401 | The pin mapping from <from-pin> to <to-pin> has not been applied for placement <placement-name> as pin <pin-name> doesn't exist in placement <placement-name> | This message occurs when the specified pin does not exist in the layout. Verify your pin mapping statements and case-sensitivity options.                                                                                                       |

**Table C-2. Calibre 3DSTACK Warning Messages (cont.)**

| Number               | Description                                                                                                                                                                                                                      | Possible Causes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_WARNING_1106 |                                                                                                                                                                                                                                  | <p>This warning is issued when the tool is unable to find either:</p> <ul style="list-style-type: none"> <li>• A stack with the specified name for all rule check commands.</li> <li>• A die for the following text check commands           <ul style="list-style-type: none"> <li>• no_texts</li> <li>• floating_texts</li> <li>• multi_texts</li> </ul> </li> </ul>                                                                                                                                                                                                        |
| 3DSTACK_WARNING_1107 | <p>Unable to find placements for the given layer type <i>layer_type</i>. Ignoring...</p>                                                                                                                                         | <p>The layer type was used in a rule check, but the layer type was not placed in the assembly. Review your stack statements to ensure that the specific die or component that contains the layer type was placed in the design.</p> <p>This warning is issued when the tool is unable to find placements for the given layer type for the following text and port check commands:</p> <ul style="list-style-type: none"> <li>• no_texts</li> <li>• floating_texts</li> <li>• multi_texts</li> <li>• extra_ports</li> <li>• missing_ports</li> <li>• dangling_ports</li> </ul> |
| 3DSTACK_WARNING_1117 | <p>New component name &lt;incorrect_value&gt; is detected for &lt;name&gt; should be &lt;correct_value&gt;. Ignoring...</p> <p>-net_mismatch is meaningful only when there is at least one layer with -net_text. Ignoring...</p> | <p>An incorrect name was specified in the COMPONENT_NAME column of the XSI netlist. The tool ignores this name and uses the correct name.</p> <p>Checking for net name mismatches can only be performed if you specify -net_text in the -layer_info option for layers.</p>                                                                                                                                                                                                                                                                                                    |

**Table C-2. Calibre 3DSTACK Warning Messages (cont.)**

| Number               | Description                                                                                                                                                                   | Possible Causes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3DSTACK_WARNING_1305 | In file <i>rules</i> : on line <i>line</i> :<br>-use_net_text is allowed for -text in export_layout only. Ignoring...<br><br>-texted is allowed for -lefdef only. Ignoring... | The -use_net_text argument can only be specified with -text. See “ <a href="#">export_layout</a> ” on page 131.<br><br>The -texted keyword is only optional for LEF/DEF layouts to attach text to layers.                                                                                                                                                                                                                                                                                                           |
| 3DSTACK_WARNING_1306 | No interacting layers are detected for check <i>check_name</i> . Ignoring...                                                                                                  | This warning is likely caused by a mistake in your die layer or stack definitions. Check that the interacting layers are defined on the correct surface of the die (-top or -bottom) and check to ensure that the die placement in the stack is of the correct vertical orientation (apply the -invert keyword to flip the die vertically), and specify -interposer if applicable.<br><br>You can also receive this message if a rule check refers to layer type that is not defined in any -layer_info option set. |
| 3DSTACK_WARNING_1311 | Connectivity check <check1> is a duplicate of <check2>. Ignoring...                                                                                                           | If you specify multiple layer types in different connected checks for the same dies, then the tool ignores the duplicated connected rule checks and only executes one of the checks.                                                                                                                                                                                                                                                                                                                                |
| 3DSTACK_WARNING_1401 | No source netlist defined. All "-source_filter" are ignored...                                                                                                                | A die command contains the -source_filter option, but no source netlist was specified with the config -source_netlist option.                                                                                                                                                                                                                                                                                                                                                                                       |
| WARNING              | There are multiple text-labels overlapping with the pads on layer <i>layer</i> . "connected" check may produce incorrect results!                                             | Overlapping text warnings are generated even if both of the text labels contain identical strings. Calibre 3DSTACK does not distinguish between multiple overlapping labels.                                                                                                                                                                                                                                                                                                                                        |



# Index

---

## — Symbols —

[]<sup>18</sup>  
{}<sup>18</sup>  
|<sup>18</sup>

## — A —

Analysis flow, [39](#)  
Assembly operations, [59](#)

## — B —

Black Box, [150](#), [151](#)  
Bold words, [17](#)

## — C —

Calibre 3DSTACK flow, [16](#)  
Calibre Interactive  
    3DSTACK Options, [59](#)  
    Create Only, [60](#)  
    Create&Use, [59](#)  
    Customization button, [62](#)  
    Customization file, [61](#)  
    Invocation, [23](#)  
    Load, [31](#)  
    Netlist tab, [55](#), [57](#)  
    Outputs, [59](#)  
    Run 3DSTACK, [31](#)  
    Select Checks, [61](#)  
    Show results in RVE, [31](#), [59](#)  
    Source Netlist, [56](#)  
    Use Generated, [60](#)  
    Write 3DSTACK Summary Report File,  
        [31](#), [59](#)

## Calibre RVE

    cross-referencing, [41](#)  
    highlight DRC results, [39](#)  
    Internal Schematic Viewer, [42](#), [52](#)  
    LNC, [45](#)  
    SNC, [45](#)  
    SourceNet, [45](#)

## CALIBRE\_HOME, [17](#)

Chip stack rule file, [12](#), [14](#)

## Commands

    anchor, [390](#)  
    attach\_text, [392](#)  
    centers, [142](#)  
    connect, [394](#), [396](#)  
    connected, [147](#)  
    copy, [158](#)  
    density, [167](#)  
    enclosure, [175](#)  
    export\_connectivity, [115](#), [360](#)  
    export\_template, [369](#)  
    external, [178](#)  
    ignore\_pin, [397](#)  
    import\_text\_labels, [402](#)  
    internal, [188](#), [206](#), [209](#), [435](#)  
    layer, [404](#)  
    layout, [407](#)  
    layout\_primary, [104](#), [411](#), [412](#)  
    map\_placement, [412](#)  
    offgrid\_centers, [199](#)  
    overlap, [201](#)  
    place\_chip, [418](#)  
    place\_layer, [421](#)  
    rename\_text, [424](#)  
    report, [111](#), [374](#)  
    run, [376](#)  
    select\_checks, [206](#), [209](#)  
    set\_version, [378](#)  
    source\_filter, [379](#)  
    source\_netlist, [108](#), [381](#)  
    tvf\_block, [435](#)  
    unselect\_checks, [209](#)

Courier font, [17](#)

## — D —

Double pipes, [18](#)

## — E —

Error messages, [438](#)

---

## — H —

Heavy font, [17](#)

## — I —

Inputs and outputs, [14](#)

Invocation, [20](#)

-3dstack, [21](#)

Calibre Interactive, [23](#)

-compile\_only, [22](#)

-create\_assembly, [21](#)

-help, [21](#)

rule\_file\_name, [22](#)

-run\_dir, [22](#)

-system, [22](#)

-turbo, [21](#)

-use\_assembly, [22](#)

Italic font, [17](#)

## — L —

License requirements, [16](#)

LNC, [148](#)

## — M —

MGC format, [109, 116, 117, 361, 363, 382](#)

MGC\_HOME, [17](#)

Micro bumps, [12](#)

Minimum keyword, [17](#)

## — N —

Notifications, [35](#)

## — P —

Parentheses, [18](#)

Pipes, [18](#)

## — Q —

Quotation marks, [18](#)

## — R —

Requirements, [16](#)

## — S —

Slanted words, [17](#)

SNC, [45, 148](#)

Source net, [45](#)

Source netlist, [57](#)

Square parentheses, [18](#)

System Netlist Generator workflow, [222](#)

System Netlist Configuration File, [235](#)

CONNECTIVITY, [237](#)

EXPORTS, [239](#)

PLACEMENTS, [236](#)

## — T —

TSVs, [11](#)

## — U —

Underlined words, [17](#)

## — V —

Verification

command line, [29, 30](#)

examples, [28](#)

## — W —

Workflow, [16](#)

## **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.

