The Liberty User Guides and Reference Manual Suite includes the following documents:

- Liberty Release Notes
- Liberty User Guide Volume 1
- Liberty User Guide Volume 2
- Liberty Reference Manual

Note:
The contents of Liberty User Guide Volume 2 have not changed since the 2007.03 release.

You can use Adobe Reader version 7 or later to view the Liberty User Guides and Reference Manual Suite.

To navigate through the Liberty User Guides and Reference Manual Suite, you can use the View > Go To menu item and select the appropriate option.
These release notes present the latest information about Liberty version 2017.06. New modeling syntax and enhancements are described in the following sections:

- New LVF Models
- Partial Voltage Swing in Timing Arcs

For detailed information about these enhancements, see the *Liberty User Guide, Vol. 1.*
New LVF Models

The following new LVF models are introduced:

- **OCV Models for Retain Arc Delay and Transition**
- **Statistical Moments-Based LVF Models For Ultra-Low Voltage Designs**

**OCV Models for Retain Arc Delay and Transition**

A retain arc models how long an output port retains its current logic value after a voltage rise or fall at a related input port. The Liberty syntax now supports new Liberty Variation Format (LVF) models in timing groups for on-chip variation (OCV) in delay and transition times of retain arcs.

To model the retain arc OCV delay, use the new `ocv_sigma_retaining_rise` and `ocv_sigma_retaining_fall` groups. Each of these groups specify a lookup table for the delay variation at the standard deviation ($\sigma$) value from the nominal retain arc rise and fall delays, respectively. Use the `sigma_type` attribute to define the type of arrival time in the lookup table.

To model the retain arc OCV transition, use the new `ocv_sigma_retain_rise_slew` and `ocv_sigma_retain_fall_slew` groups. Each of these groups specify a lookup table for the transition variation at the standard deviation ($\sigma$) value from the nominal retain arc rise and fall transitions, respectively.

**Statistical Moments-Based LVF Models For Ultra-Low Voltage Designs**

The LVF models for OCV now support asymmetric, biased, or non-Gaussian distributions of timing variation. This is useful to accurately model ultra-low voltage libraries. To capture the shape of a biased timing variation distribution, the LVF models use the statistical moments including the mean, standard deviation, and skewness of the distribution.

The syntax includes Liberty groups with lookup tables to store the moments-based variation values of the cell delays, cell transitions, cell timing constraints, retain arc delays, and retain arc transitions at one-sigma ($\sigma$) deviation. These lookup tables can be scalar, one, two, or three-dimensional and are defined under the `timing` group together with the nominal timing tables. The lookup table template for these groups are defined at the library-level using the `lu_table_template` group.
LVF OCV Mean Shift Groups

The new `ovc_mean_shift_cell_rise`, `ovc_mean_shift_cell_fall`, `ovc_mean_shift_rise_transition`, `ovc_mean_shift_fall_transition`, `ovc_mean_shift_retaining_rise`, `ovc_mean_shift_retaining_fall`, `ovc_mean_shift_retain_rise_slew`, `ovc_mean_shift_retain_fall_slew`, `ovc_mean_shift_rise_constraint`, and `ovc_mean_shift_fall_constraint` lookup tables specify the offset value from the mean of the timing variation distribution to the nominal value.

LVF OCV Standard Deviation Groups

The new `ovc_std_dev_cell_rise`, `ovc_std_dev_cell_fall`, `ovc_std_dev_rise_transition`, `ovc_std_dev_fall_transition`, `ovc_std_dev_retaining_rise`, `ovc_std_dev_retaining_fall`, `ovc_std_dev_retain_rise_slew`, `ovc_std_dev_retain_fall_slew`, `ovc_std_dev_rise_constraint`, and `ovc_std_dev_fall_constraint` look up tables store the values of standard deviation of the timing variation distribution.

LVF OCV Skewness Groups

The new `ovc_skewness_cell_rise`, `ovc_skewness_cell_fall`, `ovc_skewness_rise_transition`, `ovc_skewness_fall_transition`, `ovc_skewness_retaining_rise`, `ovc_skewness_retaining_fall`, `ovc_skewness_retain_rise_slew`, `ovc_skewness_retain_fall_slew`, `ovc_skewness_rise_constraint`, and `ovc_skewness_fall_constraint` lookup tables specify the skewness values of the timing variation distribution.

Partial Voltage Swing in Timing Arcs

You can now specify the partial voltage swing for a specific timing arc. To do so, define the `output_signal_level_low` and `output_signal_level_high` attributes in the timing group. These attributes specify the actual output voltages of an output pin after a transition through a timing arc.

You specify these attributes in the timing group when the following occur together:

- The cell output exhibits partial voltage swing (and not rail-to-rail swing).
- The voltages are different in different timing arcs.
Copyright Notice

© 2004, 2006-2009, 2011-2017 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

This document is duplicated with the permission of Synopsys, Inc., for the use of Open Source Liberty users.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx. All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com
Contents

1. Sample Library Description
   
   General Syntax ................................................................. 1-2
   Statements .............................................................................. 1-3
      Group Statements ............................................................... 1-3
      Attribute Statements .......................................................... 1-4
         Simple Attributes ............................................................ 1-4
         Complex Attributes ......................................................... 1-4
   Define Statements .................................................................... 1-5
   Reducing Library File Size ...................................................... 1-5

2. Building a Technology Library
   
   Creating Library Groups ......................................................... 2-2
   library Group ........................................................................... 2-2
   Using General Library Attributes .............................................. 2-2
      technology Attribute ........................................................... 2-2
      delay_model Attribute .......................................................... 2-3
      bus_naming_style Attribute .................................................. 2-3
   Delay and Slew Attributes ....................................................... 2-3
      input_threshold_pct_fall Simple Attribute .............................. 2-5
      input_threshold_pct_rise Simple Attribute .............................. 2-6
      output_threshold_pct_fall Simple Attribute ............................. 2-6
      output_threshold_pct_rise Simple Attribute ............................ 2-7
      slew_derate_from_library Simple Attribute ............................. 2-7
      slew_lower_threshold_pct_fall Simple Attribute ....................... 2-7
slew_lower_threshold_pct_rise Simple Attribute ........................................... 2-8
slew_upper_threshold_pct_fall Simple Attribute .......................................... 2-8
slew_upper_threshold_pct_rise Simple Attribute ........................................... 2-8

Defining Units ................................................................. 2-9
time_unit Attribute ............................................................ 2-9
voltage_unit Attribute .......................................................... 2-10
current_unit Attribute ........................................................... 2-10
pulling_resistance_unit Attribute ..................................................... 2-10
capacitive_load_unit Attribute ....................................................... 2-11
leakage_power_unit Attribute ......................................................... 2-11

3. Building Environments

Library-Level Default Attributes ....................................................... 3-2
Setting Default Cell Attributes .......................................................... 3-2
default_cell_leakage_power Simple Attribute .......................................... 3-2
Setting Default Pin Attributes ............................................................. 3-2
Setting Wire Load Defaults ................................................................. 3-3
default_wire_load Attribute ............................................................... 3-3
default_wire_load_capacitance Attribute .............................................. 3-3
default_wire_load_resistance Attribute ................................................. 3-3
default_wire_load_area Attribute ......................................................... 3-4
Setting Other Environment Defaults ..................................................... 3-4
default_operating_conditions Attribute ............................................ 3-4
default_connection_class Attribute .................................................... 3-4
Examples of Library-Level Default Attributes ......................................... 3-5

Defining Operating Conditions ............................................................ 3-5
operating_conditions Group ............................................................... 3-5

Defining Power Supply Cells ............................................................... 3-7
power_supply group .............................................................. 3-7

Defining Wire Load Groups ................................................................. 3-7
wire_load Group .............................................................. 3-8
wire_load_table Group ............................................................. 3-9

Specifying Delay Scaling Attributes .................................................. 3-10
Pin and Wire Capacitance Factors ......................................................... 3-10
Scaling Factors Associated With the Nonlinear Delay Model ........................ 3-10
4. Defining Core Cells

<table>
<thead>
<tr>
<th>Definition</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining cell Groups</td>
<td>4-2</td>
</tr>
<tr>
<td>cell Group</td>
<td>4-2</td>
</tr>
<tr>
<td>area Attribute</td>
<td>4-3</td>
</tr>
<tr>
<td>cell_footprint Attribute</td>
<td>4-3</td>
</tr>
<tr>
<td>clock_gating_integrated_cell Attribute</td>
<td>4-4</td>
</tr>
<tr>
<td>Setting Pin Attributes for an Integrated Cell</td>
<td>4-5</td>
</tr>
<tr>
<td>Setting Timing for an Integrated Cell</td>
<td>4-6</td>
</tr>
<tr>
<td>contention_condition Attribute</td>
<td>4-6</td>
</tr>
<tr>
<td>is_macro_cell Attribute</td>
<td>4-6</td>
</tr>
<tr>
<td>pad_cell Attribute</td>
<td>4-7</td>
</tr>
<tr>
<td>pin_equal Attribute</td>
<td>4-7</td>
</tr>
<tr>
<td>pin_opposite Attribute</td>
<td>4-7</td>
</tr>
<tr>
<td>type Group</td>
<td>4-7</td>
</tr>
<tr>
<td>cell Group Example</td>
<td>4-8</td>
</tr>
<tr>
<td>mode_definition Group</td>
<td>4-9</td>
</tr>
<tr>
<td>Group Statement</td>
<td>4-10</td>
</tr>
<tr>
<td>mode_value Group</td>
<td>4-10</td>
</tr>
<tr>
<td>Defining pin Groups</td>
<td>4-11</td>
</tr>
<tr>
<td>pin Group</td>
<td>4-11</td>
</tr>
<tr>
<td>General pin Group Attributes</td>
<td>4-12</td>
</tr>
<tr>
<td>capacitance Attribute</td>
<td>4-13</td>
</tr>
<tr>
<td>clock_gate_clock_pin Attribute</td>
<td>4-13</td>
</tr>
<tr>
<td>clock_gate_enable_pin Attribute</td>
<td>4-14</td>
</tr>
<tr>
<td>clock_gate_obs_pin Attribute</td>
<td>4-14</td>
</tr>
<tr>
<td>clock_gate_out_pin Attribute</td>
<td>4-14</td>
</tr>
<tr>
<td>clock_gate_test_pin Attribute</td>
<td>4-14</td>
</tr>
<tr>
<td>complementary_pin Simple Attribute</td>
<td>4-15</td>
</tr>
<tr>
<td>connection_class Simple Attribute</td>
<td>4-15</td>
</tr>
<tr>
<td>direction Attribute</td>
<td>4-17</td>
</tr>
<tr>
<td>driver_type Attribute</td>
<td>4-17</td>
</tr>
<tr>
<td>fall_capacitance Attribute</td>
<td>4-21</td>
</tr>
<tr>
<td>fault_model Simple Attribute</td>
<td>4-22</td>
</tr>
<tr>
<td>inverted_output Attribute</td>
<td>4-23</td>
</tr>
<tr>
<td>is_analog Attribute</td>
<td>4-23</td>
</tr>
<tr>
<td>pin_func_type Attribute</td>
<td>4-24</td>
</tr>
<tr>
<td>rise_capacitance Attribute</td>
<td>4-24</td>
</tr>
<tr>
<td>steady_state_resistance Attributes</td>
<td>4-25</td>
</tr>
</tbody>
</table>
test_output_only Attribute .................................................. 4-25

Describing Design Rule Checks ............................................ 4-26
  fanout_load Attribute ......................................................... 4-26
  max_fanout Attribute .......................................................... 4-27
  min_fanout Attribute .......................................................... 4-27
  max_transition Attribute ...................................................... 4-28
  max_trans Group ................................................................. 4-28
  min_transition Attribute ...................................................... 4-30
  max_capacitance Attribute .................................................... 4-31
  max_cap Group ................................................................. 4-31
  min_capacitance Attribute .................................................... 4-33
  cell_degradation Group ....................................................... 4-33

Describing Clocks ............................................................... 4-34
  clock Attribute ............................................................... 4-35
  min_period Attribute .......................................................... 4-35
  min_pulse_width_high and min_pulse_width_low Attributes .......... 4-35

Defining Bused Pins ............................................................ 4-39
  type Group ................................................................. 4-39
  bus Group ................................................................. 4-41
  bus_type Attribute .......................................................... 4-41
  Pin Attributes and Groups .................................................. 4-42

Defining Signal Bundles ....................................................... 4-46
  bundle Group ............................................................... 4-46
  members Attribute ............................................................ 4-47
  pin Attributes ............................................................... 4-47

Defining Layout-Related Multibit Attributes................................. 4-48

Defining Multiplexers .......................................................... 4-49
  Library Requirements ........................................................ 4-50

Defining Decoupling Capacitor Cells, Filler Cells, and Tap Cells .......... 4-51
  Syntax ................................................................. 4-52

Cell-Level Attributes ........................................................ 4-52
  is_decap_cell .............................................................. 4-52
  is_filler_cell .............................................................. 4-52
## 5. Defining Sequential Cells

### Using Sequential Cell Syntax

- Describing a Flip-Flop
  - Using the `ff` Group
    - `clocked_on` and `clocked_on_also` Attributes
    - `next_state` Attribute
    - `nextstate_type` Attribute
    - `clear` Attribute
    - `preset` Attribute
    - `clear_preset_var1` Attribute
    - `clear_preset_var2` Attribute
    - `power_down_function` Attribute
    - `ff` Group Examples
  - Describing a Single-Stage Flip-Flop
  - Describing a Master-Slave Flip-Flop
- Using the `function` Attribute
- Describing a Multibit Flip-Flop
- Describing a Latch
  - `latch` Group
    - `enable` and `data_in` Attributes
    - `data_in_type` Attribute
    - `clear` Attribute
    - `preset` Attribute
    - `clear_preset_var1` Attribute
    - `clear_preset_var2` Attribute
    - Determining a Latch Cell’s Internal State
  - Describing a Multibit Latch
  - `latch_bank` Group
- Describing Sequential Cells With the Statetable Format
  - `statetable` Group
  - Using Shortcuts
  - Partitioning the Cell Into a Model
  - Defining an Output pin Group
    - `state_function` Attribute
    - `internal_node` Attribute
6. Defining Test Cells

Describing a Scan Cell................................................................. 6-2
test_cell Group ........................................................................ 6-2
  Pins in the test_cell Group ..................................................... 6-3
test_output_only Attribute ........................................................ 6-3
Describing a Multibit Scan Cell..................................................... 6-3
  Multibit Scan Cell With Parallel Scan Chain .............................. 6-4
    Example .............................................................................. 6-6
  Multibit Scan Cell With Internal Scan Chain .............................. 6-10
    Examples ............................................................................. 6-13
Scan Cell Modeling Examples ....................................................... 6-17
  Simple Multiplexed D Flip-Flop ................................................. 6-17
  Multibit Cells With Multiplexed D Flip-Flop and Enable ............... 6-19
  LSSD Scan Cell ....................................................................... 6-24
Scan-Enabled LSSD Cell ................................................................. 6-27
  Functional Model of the Scan-Enabled LSSD Cell ......................... 6-28
  Timing Model of the Scan-Enabled LSSD Cell ............................... 6-29
  Scan-Enabled LSSD Cell Model Example .................................... 6-30
  Scan-Enabled LSSD Cell With Asynchronous Inputs ..................... 6-32
    Multibit Scan-Enabled LSSD Cell ............................................ 6-34
Clocked-Scan Test Cell ................................................................. 6-36
  Scan D Flip-Flop With Auxiliary Clock ...................................... 6-38

7. Timing Arcs

Understanding Timing Arcs.......................................................... 7-3
  Combinational Timing Arcs ....................................................... 7-3
  Sequential Timing Arcs ............................................................ 7-4
Modeling Method Alternatives .................................................... 7-4
Defining the timing Group .......................................................... 7-5
  Naming Timing Arcs Using the timing Group. ........................... 7-6
    Timing Arc Between a Single Pin and a Single Related Pin .......... 7-6
    Timing Arcs Between a Pin and Multiple Related Pins .......... 7-7
    Timing Arcs Between a Bundle and a Single Related Pin .......... 7-8
    Timing Arcs Between a Bundle and Multiple Related Pins ....... 7-9
    Timing Arcs Between a Bus and a Single Related Pin ............ 7-10
    Timing Arcs Between a Bus and Multiple Related Pins .......... 7-11
    Timing Arcs Between a Bus and a Related Bus Equivalent ...... 7-13

  Delay Model. ................................................................. 7-15
    delay_model Attribute ................................................. 7-16
  Defining the NLDM Template .............................................. 7-16
  Creating Lookup Tables .................................................. 7-19
  Defining the Scalable Polynomial Delay Model Template ............ 7-21

  timing Group Attributes ................................................. 7-23
    related_pin Simple Attribute ........................................ 7-23
    related_bus_pins Simple Attribute .................................. 7-24
    timing_sense Simple Attribute ...................................... 7-25
    timing_type Simple Attribute ....................................... 7-26
    mode Complex Attribute ............................................. 7-30

  Describing Three-State Timing Arcs .................................... 7-34
    Describing Three-State-Disable Timing Arcs ....................... 7-34
    Describing Three-State-Enable Timing Arcs ....................... 7-35

  Describing Edge-Sensitive Timing Arcs ................................ 7-36

  Describing Clock Insertion Delay ....................................... 7-36

  Describing Intrinsic Delay ............................................. 7-37
    In the CMOS Piecewise Linear Delay Model ......................... 7-38
    In the Scalable Polynomial Delay Model ........................... 7-38

  Describing Transition Delay ............................................ 7-38
    Defining Delay Arcs With Lookup Tables .......................... 7-38
    Modeling Transition Time Degradation .............................. 7-43

  Modeling Load Dependency .............................................. 7-46
    In the CMOS Scalable Polynomial Delay Model ..................... 7-47

  Describing Slope Sensitivity ........................................... 7-49

  Describing State-Dependent Delays .................................... 7-49
    when Simple Attribute ............................................. 7-50
sdf_cond Simple Attribute .................................................. 7-52
Setting Setup and Hold Constraints ........................................ 7-54
rise_constraint and fall_constraint Groups ............................... 7-55
In the Scalable Polynomial Delay Model ................................. 7-57
Identifying Interdependent Setup and Hold Constraints .............. 7-58
Setting Nonsequential Timing Constraints ............................... 7-58
Setting Recovery and Removal Timing Constraints ...................... 7-60
  Recovery Constraints .................................................... 7-60
  Removal Constraint ..................................................... 7-62
Setting No-Change Timing Constraints .................................... 7-63
  In the CMOS Scalable Polynomial Delay Model ..................... 7-67
Setting Skew Constraints ................................................... 7-68
Setting Conditional Timing Constraints .................................. 7-69
  when and sdf_cond Simple Attributes ................................ 7-70
  when_start Simple Attribute ......................................... 7-70
  sdf_cond_start Simple Attribute ...................................... 7-70
  when_end Simple Attribute ........................................... 7-71
  sdf_cond_end Simple Attribute ........................................ 7-71
  sdf_edges Simple Attribute ........................................... 7-71
  min_pulse_width Group ................................................ 7-72
    min_pulse_width Example ............................................ 7-72
    constraint_high and constraint_low Simple Attributes .......... 7-72
    when and sdf_cond Simple Attributes ................................ 7-72
  minimum_period Group ................................................. 7-72
    minimum_period Example ............................................ 7-72
    constraint Simple Attribute ...................................... 7-73
    when and sdf_cond Simple Attributes ................................ 7-73
    min_pulse_width and minimum_period Example ..................... 7-73
Using Conditional Attributes With No-Change Constraints .......... 7-75

Impossible Transitions .................................................... 7-76

Examples of NLDM Libraries ................................................. 7-76
  CMOS Piecewise Linear Delay Model ................................. 7-76
  Library With Timing Constraints .................................... 7-79
  CMOS Scalable Polynomial Delay Model .............................. 7-82
  Library With Clock Insertion Delay .................................. 7-86
Describing a Transparent Latch Clock Model ............................................ 7-86
Driver Waveform Support ................................................................. 7-88
   Library-Level Tables, Attributes, and Variables ............................... 7-90
      normalized_voltage Variable ..................................................... 7-90
      normalized_driver_waveform Group ........................................... 7-90
      driver_waveform_name Attribute .............................................. 7-91
   Cell-Level Attributes ............................................................... 7-91
      driver_waveform Attribute ...................................................... 7-91
      driver_waveform_rise and driver_waveform_fall Attributes .............. 7-91
   Pin-Level Attributes ............................................................... 7-92
      driver_waveform Attribute ...................................................... 7-92
      driver_waveform_rise and driver_waveform_fall Attributes .............. 7-92
   Driver Waveform Example .......................................................... 7-92
Sensitization Support ................................................................. 7-94
   sensitization Group .................................................................... 7-94
      pin_names Attribute ............................................................... 7-95
      vector Attribute ................................................................. 7-95
   Cell-Level Attributes ............................................................... 7-95
      sensitization_master Attribute .................................................. 7-96
      pin_name_map Attribute .......................................................... 7-96
   timing Group Attributes ............................................................. 7-96
      sensitization_master Attribute .................................................. 7-96
      pin_name_map Attribute .......................................................... 7-96
      wave_rise and wave_fall Attributes .......................................... 7-97
      wave_rise_sampling_index and wave_fall_sampling_index Attributes ... 7-98
      wave_rise_time_interval and wave_fall_time_interval Attributes ........ 7-98
   timing Group Syntax ................................................................. 7-99
NAND Cell Example ........................................................................... 7-100
Complex Macro Cell Example .......................................................... 7-101
Phase-Locked Loop Support ........................................................... 7-103
   Phase-Locked Loop Syntax ........................................................... 7-104
   Cell-Level Attribute ................................................................. 7-104
      is_pll_cell Attribute ............................................................. 7-104
   Pin-Level Attributes ................................................................. 7-104
      is_pll_reference_pin Attribute .................................................. 7-105
      is_pll_feedback_pin Attribute ................................................... 7-105
      is_pll_output_pin Attribute ...................................................... 7-105
   Phase-Locked Loop Example ........................................................ 7-105
8. **Modeling Power and Electromigration**

Modeling Power Terminology ................................................................. 8-2
- Static Power ......................................................................................... 8-2
- Dynamic Power ................................................................................... 8-2
  - Internal Power .................................................................................. 8-2
  - Switching Power .............................................................................. 8-3

Switching Activity .................................................................................. 8-4

Modeling for Leakage Power ................................................................. 8-4

Representing Leakage Power Information ............................................. 8-4
  - cell_leakage_power Simple Attribute ............................................... 8-4
  - Using the leakage_power Group for a Single Value ......................... 8-5
    - mode Complex Attribute .............................................................. 8-5
    - when Simple Attribute ............................................................... 8-6
    - value Simple Attribute ............................................................... 8-8
  - leakage_power_unit Simple Attribute .............................................. 8-8
  - default_cell_leakage_power Simple Attribute ................................. 8-9
  - Example ............................................................................................ 8-9

Threshold Voltage Modeling .................................................................. 8-12

Modeling for Internal and Switching Power ........................................ 8-13
  - Modeling Internal Power Lookup Tables ......................................... 8-14

Representing Internal Power Information ............................................. 8-15
  - Specifying the Power Model ............................................................. 8-15
  - Using Lookup Table Templates ...................................................... 8-15
    - power_lut_template Group ............................................................ 8-16
    - Scalar power_lut_template Group ................................................. 8-18

Defining Internal Power Groups ........................................................... 8-18
  - Naming Power Relationships, Using the internal_power Group .......... 8-18
    - Power Relationship Between a Single Pin and a Single Related Pin ... 8-19
    - Power Relationships Between a Single Pin and Multiple Related Pins .................................................. 8-19
    - Power Relationships Between a Bundle and a Single Related Pin .... 8-20
    - Power Relationships Between a Bundle and Multiple Related Pins ........................................................................ 8-21
    - Power Relationships Between a Bus and a Single Related Pin ....... 8-23
    - Power Relationships Between a Bus and Multiple Related Pins .... 8-24
    - Power Relationships Between a Bus and Related Bus Pins .......... 8-26

internal_power Group ............................................................................. 8-26
9. **Advanced Low-Power Modeling**

Power and Ground (PG) Pins ........................................ 9-3
Example Libraries With Multiple Power Supplies .......... 9-3
Partial PG Pin Cell Modeling ................................. 9-6
mode Attribute in minimum_period and min_pulse_width Groups .................. 9-32
Defining PG Modes in Macro Cells .................................................. 9-33
Examples ......................................................................................... 9-33
Cell With Different Power States ....................................................... 9-34
Power State Transition ...................................................................... 9-35
Macro Cell With PG Modes ............................................................... 9-37
Retention Cell Leakage Power in Different Power Modes ................. 9-40
Feedthrough Signal Pin Modeling ...................................................... 9-46
Multipin Feedthroughs .................................................................... 9-46
short Attribute .................................................................................. 9-47
Single-Pin Feedthrough ................................................................. 9-48
Silicon-on-Insulator (SOI) Cell Modeling .......................................... 9-50
Library-Level Attribute ..................................................................... 9-52
is_soi Attribute ................................................................................ 9-52
Cell-Level Attribute .......................................................................... 9-52
is_soi Attribute ................................................................................ 9-52
Level-Shifter Cells in a Multivoltage Design .................................. 9-52
Operating Voltages .......................................................................... 9-53
Level Shifter Functionality ............................................................... 9-53
Basic Level-Shifter Syntax ............................................................... 9-53
Cell-Level Attributes ........................................................................ 9-54
is_level_shifter Attribute ................................................................. 9-54
level_shifter_type Attribute ............................................................. 9-54
input_voltage_range Attribute ....................................................... 9-55
output_voltage_range Attribute ..................................................... 9-55
ground_input_voltage_range Attribute .......................................... 9-56
ground_output_voltage_range Attribute ......................................... 9-56
Pin-Level Attributes .......................................................................... 9-56
std_cell_main_rail Attribute ............................................................. 9-56
level_shifter_data_pin Attribute ....................................................... 9-56
level_shifter_enable_pin Attribute .................................................... 9-56
input_voltage_range and output_voltage_range Attributes .......... 9-57
ground_input_voltage_range Attribute .......................................... 9-57
ground_output_voltage_range Attribute ......................................... 9-57
input_signal_level Attribute ........................................................... 9-57
power_down_function Attribute ....................................................... 9-57
alive_during_power_up Attribute ....................................................... 9-58
Enable Level-Shifter Cell ................................................................. 9-58
Fine-Grained Switch Support for Macro Cells .......................... 9-102
Macro Cell With Fine-Grained Switch Syntax ............................. 9-103
Cell-Level Attributes ...................................................... 9-103
pg_pin Group ..................................................................... 9-104
Switch-Cell Modeling Examples .............................................. 9-104
Simple Coarse-Grain Header Switch Cell ................................. 9-104
Complex Coarse-Grain Header Switch Cell ............................... 9-106
Complex Coarse-Grain Switch Cell With an Internal Switch Pin ...... 9-108
Complex Coarse-Grain Switch Cell With Parallel Switches ......... 9-110
Retention Cell Modeling ...................................................... 9-113
Modes of Operation ......................................................... 9-116
Retention Cell Modeling Syntax ............................................. 9-116
Cell-Level Attributes, Groups, and Variables ......................... 9-119
  retention_cell Simple Attribute ...................................... 9-119
  ff, latch, ff_bank, and latch_bank Groups ........................... 9-119
  retention_condition Group .............................................. 9-119
  clock_condition Group .................................................. 9-120
  preset_condition and clear_condition Groups ...................... 9-121
  reference_pin_names Variable ........................................ 9-121
  variable1 and variable2 Variables ..................................... 9-122
  bits Variable ............................................................. 9-122
Pin-Level Attributes ......................................................... 9-122
  retention_pin Complex Attribute .................................... 9-122
  function Attribute ...................................................... 9-122
  reference_input Attribute ............................................ 9-123
  save_action and restore_action Attributes ......................... 9-123
  restore_edge_type Attribute ......................................... 9-123
  save_condition and restore_condition Attributes .................. 9-124
Retention Cell Model Examples ............................................ 9-124
Always-On Cell Modeling .................................................. 9-142
Always-On Cell Syntax .................................................... 9-142
always_on Simple Attribute ............................................. 9-142
Always-On Simple Buffer Example ....................................... 9-143
Macro Cell Modeling ....................................................... 9-144
Macro Cell Isolation Modeling ............................................ 9-144
Pin-Level Attributes ....................................................... 9-147
Modeling Macro Cells With Internal PG Pins ......................... 9-147
Macro Cell With Fine-Grained Internal Power Switches .............. 9-150
10. Composite Current Source Modeling

Modeling Cells With Composite Current Source Information .......................... 10-2

Representing Composite Current Source Driver Information ........................ 10-2

Composite Current Source Lookup Tables ...................................................... 10-2

Defining the output_current_template Group ................................................... 10-2

  output_current_template Syntax ..................................................................... 10-2
  Template Variables for CCS Driver Models ................................................... 10-3

Defining the Lookup Table Output Current Groups ........................................... 10-3

  output_current_rise Syntax ........................................................................... 10-3

vector Group ....................................................................................................... 10-4

  reference_time Simple Attribute ................................................................... 10-4

Template Variables for CCS Driver Models ....................................................... 10-4

vector Group Example ......................................................................................... 10-4

Representing Composite Current Source Receiver Information ...................... 10-5

Comparison Between Two-Segment and Multisegment Receiver Models .......... 10-5

Two-Segment Receiver Capacitance Model ....................................................... 10-6

Syntax .................................................................................................................. 10-6

Defining the receiver_capacitance Group at the Pin Level ................................ 10-7

  Defining the lu_table_template Group ......................................................... 10-8

  Conditional Data Modeling .............................................................................. 10-9

  Examples ............................................................................................................ 10-9

Defining the Receiver Capacitance Groups at the Timing Level ...................... 10-13

  Conditional Data Modeling .............................................................................. 10-14

  Defining the lu_table_template Group ......................................................... 10-14

  Timing-Level receiver_capacitance Example ............................................... 10-15

Multisegment Receiver Capacitance Model ...................................................... 10-15
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library-Level Groups and Attributes</td>
<td>10-16</td>
</tr>
<tr>
<td>lu_table_template Group</td>
<td>10-16</td>
</tr>
<tr>
<td>receiver_capacitance_rise_threshold_pct and</td>
<td></td>
</tr>
<tr>
<td>receiver_capacitance_fall_threshold_pct Attributes</td>
<td>10-17</td>
</tr>
<tr>
<td>Pin and Timing Level Groups</td>
<td>10-18</td>
</tr>
<tr>
<td>receiver_capacitance Group</td>
<td>10-18</td>
</tr>
<tr>
<td>Conditional Data Modeling</td>
<td>10-19</td>
</tr>
<tr>
<td>Example</td>
<td>10-19</td>
</tr>
<tr>
<td>CCS Retain Arc Support</td>
<td>10-21</td>
</tr>
<tr>
<td>CCS Retain Arc Syntax</td>
<td>10-22</td>
</tr>
<tr>
<td>ccs_retain_rise and ccs_retain_fall Groups</td>
<td>10-23</td>
</tr>
<tr>
<td>vector Group</td>
<td>10-23</td>
</tr>
<tr>
<td>reference_time Attribute</td>
<td>10-23</td>
</tr>
<tr>
<td>Compact CCS Retain Arc Syntax</td>
<td>10-23</td>
</tr>
<tr>
<td>compact_ccs_retain_rise and compact_ccs_retain_fall Groups</td>
<td>10-24</td>
</tr>
<tr>
<td>base_curves_group Attribute</td>
<td>10-25</td>
</tr>
<tr>
<td>index_1, index_2, and index_3 Attributes</td>
<td>10-25</td>
</tr>
<tr>
<td>values Attribute</td>
<td>10-25</td>
</tr>
<tr>
<td>Composite Current Source Driver and Receiver Model Example</td>
<td>10-25</td>
</tr>
<tr>
<td>11. Advanced Composite Current Source Modeling</td>
<td></td>
</tr>
<tr>
<td>Modeling Cells With Advanced Composite Current Source Information</td>
<td>11-2</td>
</tr>
<tr>
<td>Compact CCS Timing Model</td>
<td>11-2</td>
</tr>
<tr>
<td>Modeling With CCS Timing Base Curves</td>
<td>11-2</td>
</tr>
<tr>
<td>Compact CCS Timing Model Syntax</td>
<td>11-4</td>
</tr>
<tr>
<td>base_curves Group</td>
<td>11-6</td>
</tr>
<tr>
<td>compact_lut_template Group</td>
<td>11-6</td>
</tr>
<tr>
<td>compact_ccs_{rise</td>
<td>fall} Group</td>
</tr>
<tr>
<td>Compact CCS Timing Library Example</td>
<td>11-7</td>
</tr>
<tr>
<td>Variation-Aware Timing Modeling Support</td>
<td>11-9</td>
</tr>
<tr>
<td>Variation-Aware Compact CCS Timing Driver Model</td>
<td>11-9</td>
</tr>
<tr>
<td>timing_based_variation Group</td>
<td>11-10</td>
</tr>
<tr>
<td>va_compact_ccs_rise and va_compact_ccs_fall Groups</td>
<td>11-11</td>
</tr>
<tr>
<td>timing_based_variation and pin_based_variation Groups</td>
<td>11-12</td>
</tr>
<tr>
<td>Variation-Aware CCS Timing Receiver Model</td>
<td>11-14</td>
</tr>
<tr>
<td>timing_based_variation and pin_based_variation Groups</td>
<td>11-15</td>
</tr>
<tr>
<td>va_parameters Complex Attribute</td>
<td>11-15</td>
</tr>
<tr>
<td>nominal_va_values Complex Attribute</td>
<td>11-15</td>
</tr>
</tbody>
</table>
va_receiver_capacitance1_rise,
va_receiver_capacitance1_fall,
va_receiver_capacitance2_rise,
and va_receiver_capacitance2_fall Groups ................................................. 11-16
va_values Attribute ..................................................................................... 11-16
Variation-Aware Timing Constraint Modeling ........................................... 11-16
timing_based_variation Group ................................................................. 11-17
va_parameters Complex Attribute ......................................................... 11-17
nominal_va_values Complex Attribute .................................................. 11-18
va_rise_constraint and va_fall_constraint Groups ...................................... 11-18
va_values Attribute ..................................................................................... 11-18
Conditional Data Modeling for Variation-Aware Timing Receiver Models .... 11-18
when Attribute .......................................................................................... 11-20
mode Attribute .......................................................................................... 11-20
Variation-Aware Compact CCS Retain Arcs ........................................... 11-25
va_compact_ccs_retain_rise and va_compact_ccs_retain_fall Groups ...... 11-26
va_values Attribute ..................................................................................... 11-26
values Attribute ......................................................................................... 11-26
Variation-Aware Syntax Examples .......................................................... 11-26

12. Nonlinear Signal Integrity Modeling

Modeling Noise Terminology .................................................................. 12-2
Noise Calculation ...................................................................................... 12-2
Noise Immunity .......................................................................................... 12-2
Noise Propagation ..................................................................................... 12-3
Modeling Cells for Noise .......................................................................... 12-3
I-V Characteristics and Drive Resistance ............................................. 12-3
Noise Immunity .......................................................................................... 12-5
Using the Hyperbolic Model ................................................................. 12-8
Noise Propagation ..................................................................................... 12-8
Representing Noise Calculation Information .......................................... 12-10
I-V Characteristics Lookup Table Model ............................................. 12-10
iv_lut_template Group .............................................................................. 12-10
Defining the Lookup Table Steady-State Current Groups ..................... 12-11
I-V Characteristics Curve Polynomial Model ......................................... 12-12
poly_template Group .............................................................................. 12-12
Defining Polynomial Steady-State Current Groups ............................... 12-13
Using Steady-State Resistance Simple Attributes .................................. 12-15
Chapter 1: Contents

1. Using I-V Curves and Steady-State Resistance for tied_off Cells ........................................ 12-15
   Defining tied_off Attribute Usage .................................................................................. 12-16

Representing Noise Immunity Information ........................................................................... 12-17
   Noise Immunity Lookup Table Model ........................................................................... 12-17
   noise_lut_template Group ............................................................................................. 12-17
   Defining the Noise Immunity Table Groups ................................................................. 12-18
   Noise Immunity Polynomial Model ............................................................................... 12-19
   poly_template Group ...................................................................................................... 12-20
   Defining the Noise Immunity Polynomial Groups ....................................................... 12-21

Input Noise Width Ranges at the Pin Level .......................................................................... 12-22
   Defining the input_noise_width Range Limits ............................................................. 12-22
   Defining the Hyperbolic Noise Groups ......................................................................... 12-23

Representing Propagated Noise Information ....................................................................... 12-25
   Propagated Noise Lookup Table Model ...................................................................... 12-25
   propagation_lut_template Group .................................................................................. 12-25
   Defining the Propagated Noise Table Groups .............................................................. 12-26
   Propagated Noise Polynomial Model ........................................................................... 12-28
   poly_template Group ...................................................................................................... 12-29
   Defining Propagated Noise Groups for Polynomial Representation ............................ 12-30

Examples of Modeling Noise ............................................................................................... 12-32
   Scalable Polynomial Model Noise Example .................................................................. 12-32
   Nonlinear Delay Model Library With Noise Information .............................................. 12-39

13. Composite Current Source Signal Integrity Modeling

CCS Signal Integrity Modeling Overview .............................................................................. 13-2
    CCS Signal Integrity Modeling Syntax .......................................................................... 13-2
    Library-Level Groups and Attributes .......................................................................... 13-7
    lu_table_template Group .............................................................................................. 13-7
    variable_1, variable_2, variable_3, and variable_4 Attributes ...................................... 13-7
    Pin-Level Groups and Attributes ................................................................................ 13-7
    ccsn_first_stage and ccsn_last_stage Groups ............................................................ 13-7
    is_needed Attribute ...................................................................................................... 13-8
    is_inverting Attribute ................................................................................................. 13-8
    stage_type Attribute .................................................................................................. 13-9
    miller_cap_rise and miller_cap_fall Attributes ............................................................ 13-9
    output_signal_level and input_signal_level Attributes ........................................... 13-9
    dc_current Group ....................................................................................................... 13-9
output_voltage_rise and output_voltage_fall Groups ........................................ 13-10
propagated_noise_high and propagated_noise_low Groups ....................... 13-10
when Attribute ................................................................. 13-10
CCS Noise Library Example ........................................................................ 13-10
Conditional Data Modeling in CCS Noise Models ..................................... 13-13
when Attribute ............................................................................... 13-14
mode Attribute ................................................................................ 13-15
CCS Noise Modeling for Unbuffered Cells With a Pass Gate ...................... 13-16
Syntax for Unbuffered Output Latches ....................................................... 13-17
Pin-Level Attributes ................................................................................ 13-18
is_unbuffered Attribute ......................................................................... 13-18
has_pass_gate Attribute ........................................................................ 13-19
ccsn_first_stage Group .......................................................................... 13-19
is_pass_gate Attribute ........................................................................... 13-19
CCS Noise Modeling for Multivoltage Designs ......................................... 13-19
Referenced CCS Noise Modeling .............................................................. 13-22
Modeling Syntax ..................................................................................... 13-23
Cell-Level Attributes ................................................................................ 13-24
driver_waveform Attribute ...................................................................... 13-24
driver_waveform_rise and driver_waveform_fall Attributes ....................... 13-24
Pin-Level Groups .................................................................................... 13-24
input_crb and output_crb Groups ............................................................ 13-24
Timing-Level Attributes ............................................................................ 13-26
active_input_crb Attribute ...................................................................... 13-26
active_output_crb Attribute .................................................................... 13-26
propagating_crb Attribute ...................................................................... 13-26
Examples ................................................................................................. 13-27

14. Composite Current Source Power Modeling

Composite Current Source Power Modeling .................................................. 14-2
Cell Leakage Current ................................................................................... 14-2
Gate Leakage Modeling in Leakage Current .............................................. 14-3
gate_leakage Group ................................................................................. 14-4
input_low_value Attribute ........................................................................ 14-4
input_high_value Attribute ....................................................................... 14-4
Intrinsic Parasitic Models ......................................................................... 14-5
Voltage-Dependent Intrinsic Parasitic Models .......................................... 14-7
Library-Level Group .................................................................................. 14-9
1. On-Chip Variation (OCV) Modeling

OCV Model Overview ........................................ 15-2
Advanced OCV Modeling ................................. 15-2

Library-Level Groups and Attributes .................. 15-3
  ocv_table_template Group ............................ 15-3
  ocv_derate Group ...................................... 15-4
  default_ocv_derate_group Attribute ............... 15-5
  ocv_arc_depth Attribute ............................. 15-5

Cell-Level Attribute ................................. 15-5
  ocv_derate_group Attribute ......................... 15-5

Advanced OCV Library Example ..................... 15-5

LVF Models For Cell Delay, Transition, and Constraint .......................... 15-7
16. Defining I/O Pads

Special Characteristics of I/O Pads .......................... 16-2
Identifying Pad Cells ................................................................. 16-2
  is_pad Attribute ................................................................. 16-2
  driver_type Attribute. ......................................................... 16-3
Defining Units for Pad Cells .................................................. 16-6
  Capacitance ................................................................. 16-6
  Resistance ................................................................. 16-6
  Voltage ................................................................. 16-6
  Current ................................................................. 16-7
Describing Input Pads. ............................................................ 16-7
  hysteresis Attribute ......................................................... 16-7
Describing Output Pads .......................................................... 16-8
  Drive Current ................................................................. 16-8
  Slew-Rate Control ............................................................. 16-8
Modeling Wire Load for Pads .................................................. 16-10
Programmable Driver Type Support in I/O Pad Cell Models .......... 16-11
  Syntax ................................................................. 16-11
  Programmable Driver Type Functions .................................... 16-12
    Example ................................................................. 16-13
Pad Cell Examples ............................................................... 16-15
  Input Pads ................................................................. 16-15
  Output Pads ................................................................. 16-18
  Bidirectional Pad ........................................................... 16-19
    Cell with contention_condition and x_function ..................... 16-21

17. Library Characterization Configuration

  The char_config Group .......................................................... 17-2
  Library Characterization Configuration Syntax ......................... 17-2
Common Characterization Attributes ..................................... 17-5
  driver_waveform Attribute ............................................... 17-7
  driver_waveform_rise and driver_waveform_fall Attributes ........ 17-8
  input_stimulus_transition Attribute .................................... 17-8
  input_stimulus_interval Attribute ...................................... 17-8
  unrelated_output_net_capacitance Attribute ......................... 17-9
  default_value_selection_method Attribute ....................... 17-9
default_value_selection_method_rise and
default_value_selection_method_fall Attributes ........................................ 17-9
merge_tolerance_abs and merge_tolerance_rel Attributes. ......................... 17-10
merge_selection Attribute ........................................................................ 17-11

CCS Timing Characterization Attributes ......................................................... 17-11
ccs_timing_segment_voltage_tolerance_rel Attribute ................................... 17-11
ccs_timing_delay_tolerance_rel Attribute ..................................................... 17-12
ccs_timing_voltage_margin_tolerance_rel Attribute ..................................... 17-12

CCS Receiver Capacitance Attributes ............................................................ 17-12

Input-Capacitance Characterization Attributes ............................................ 17-12

capacitance_voltage_lower_threshold_pct_rise and
capacitance_voltage_lower_threshold_pct_fall Attributes ......................... 17-13
capacitance_voltage_upper_threshold_pct_rise and
capacitance_voltage_upper_threshold_pct_fall Attributes. ....................... 17-13
1

Sample Library Description

This chapter familiarizes you with the basic format and syntax of a library description. The chapter starts with an example that shows the general syntax of a library. The structure of the library description is also discussed. These topics are covered in the following sections:

This chapter includes the following sections:

- General Syntax
- Statements
- Reducing Library File Size
General Syntax

Example 1-1 shows the general syntax of a library description. The first statement names the library. The statements that follow are library-level attributes that apply to the entire library. These statements define library features such as the technology type, definitions, and defaults. Every cell in the library has a separate cell description.

Example 1-1  General Syntax of the Library Description

```plaintext
library (name) {
    technology (name) ; /* library-level attributes */
    delay_model : generic_cmos | table_lookup | cmos2 |
                  piecewise_cmos | dcm | polynomial ;
    bus_naming_style : string;
    routing_layers (string) ;
    time_unit : unit ;
    voltage_unit : unit ;
    current_unit : unit ;
    pulling_resistance_unit : unit ;
    capacitive_load_unit (value, unit) ;
    leakage_power_unit : unit ;
    piece_type : type ;
    piece_define ("list") ;
    define_cell_area (area_name, resource_type) ;

    /* default values for environment definitions */
    operating_conditions (name) {
        /* operating conditions */
    }
    timing_range (name) {
        /* timing information */
    }
    wire_load (name) {
        /* wire load information */
    }
    wire_load_selection () {
        /* area/group selections */
    }
    power_lut_template (name) {
        /* power lookup table template information */
    }
    cell (name1) { /* cell definitions */
        /* cell information */
    }
    cell (name2) { /* cell information */
    }
    scaled_cell (name1) {
        /* alternate scaled cell information */
    }
    ...
```
Statements

Statements are the building blocks of a library. All library information is described in one of the following types of statements:

- Group statements
- Attribute statements
- Define statements

A statement can continue across multiple lines. A continued line ends with a backslash (\).

Group Statements

A group is a named collection of statements that defines a library, a cell, a pin, a timing arc, and so forth. Braces ({}), which are used in pairs, enclose the contents of the group.

Syntax

```plaintext
group_name (name) {
    ... statements ...
}
name
```

A string that identifies the group. Check the individual group statement syntax definition to verify if name is required, optional, or null.

You must type the group name and the { symbol on the same line.

Example 1-2 defines a pin group A.

Example 1-2   Group Statement Specification

```plaintext
pin(A) {
    ... pin group statements ...
}```
Attribute Statements

An attribute statement defines characteristics of specific objects in the library. Attributes applying to specific objects are assigned within a group statement defining the object, such as a cell group or a pin group.

In this user guide, attribute refers to all attributes unless the manual specifically states otherwise. For clarity, this manual distinguishes different types of attribute statements according to syntax. All simple attributes use the same general syntax; complex attributes have different syntactic requirements.

Simple Attributes

This is the syntax of a simple attribute:

```
attribute_name : attribute_value ;
```

You must separate the attribute name from the attribute value with a space, followed by a colon and another space. Place attribute statements on a single line.

Example 1-3 adds a direction attribute to pin group A shown in Example 1-2.

```
Example 1-3 Simple Attribute Specification
pin(A) {
  direction : output ;
}
```

For some simple attributes, you must enclose the attribute value in quotation marks:

```
attribute_name : "attribute_value" ;
```

Example 1-4 adds the X + Y function to the pin example.

```
Example 1-4 Defining the Function of a Pin
pin (A) {
  direction : output ;
  function : "X + Y" ;
}
```

Complex Attributes

This is the syntax of a complex attribute statement. Include one or more parameters in parentheses.

```
attribute_name (parameter1, [parameter2, parameter3 ...] ) ;
```
The following example uses the `line` complex attribute to define a line on a schematic symbol. This line is drawn from coordinates (1,2) to coordinates (6,8):

```
line (1, 2, 6, 8);
```

---

**Define Statements**

You can create new simple attributes with the `define` statement.

**Syntax**

```
define (attribute_name, group_name, attribute_type) ;
```

- **attribute_name**
  - The name of the new attribute you are creating.

- **group_name**
  - The name of the group statement in which this attribute is specified.

- **attribute_type**
  - The type of attribute you are creating: Boolean, string, integer, or floating point.

For example, to define a new string attribute called `bork`, which is valid in a `pin` group, use

```
define (bork, pin, string) ;
```

You give the new attribute a value using the simple attribute syntax:

```
bork : "nimo" ;
```

---

**Reducing Library File Size**

Large library files can compromise disk capacity and memory resources. To reduce file size and improve file management, the syntax allows you to combine multiple source files by referencing the files from within the source file containing the `library` group description. During library compilation, the referenced information is retrieved, included at the point of reference, and then the compilation continues.

Use the `include_file` attribute to reference information in another file for inclusion during library compilation. Be sure the directory of the included file is defined in your search path—the `include_file` attribute takes only the file name as its value; no path is allowed.

**Syntax**

```
include_file (file_name_id) ;
```
Example

cell( ) {
    area : 0.1 ;
    ...
    include_file (memory_file) ;
    ...
}

where memory_file contains the memory group statements.

Limitations

The include_file attribute has these requirements:

- Recursive include_file statements are not allowed; that is, the source files that you include cannot also contain include_file statements.
- If the included file is not in the current directory, then the location of the included file must be defined in your search path.
- Multiple file names are not allowed in an include_file statement. However, there is no limit to the number of include_file statements you can have in your main source file.
- An included file cannot substitute for a group value statement. For example, the following is not allowed:

  cell ( ) {
      area : 0.1 ;
      ...
      pin_equal : include_file (source_file) ;
  }

- The include_file attribute cannot substitute or cross the group boundary. For example, the following is not allowed:

  cell ( A ) include (source_file)

where source_file is the following:

{  
    attribute : value ;
    attribute : value ;
    ...
}
A library description identifies the characteristics of a technology library and the cells it contains. The library group contains the entire library description. This chapter describes the library group-level attributes of a CMOS technology library. The following concepts and tasks are explained in this chapter:

- Creating Library Groups
- Using General Library Attributes
- Delay and Slew Attributes
- Defining Units
- Setting Minimum Cell Requirements
Creating Library Groups

The **library group** contains the entire library description. Each library source file must have only one **library group**. Attributes that apply to the entire library are defined at the **library group** level, at the beginning of the library description.

**library Group**

The **library group** statement defines the name of the library you want to describe. This statement must be the first executable line in your library.

**Example**

```
library (my_library) {
...
}
```

Using General Library Attributes

These attributes apply generally to the technology library:

- **technology**
- **delay_model**
- **bus_naming_style**

**technology Attribute**

This attribute identifies the technology used in the library. Valid value is **cmos**. The **technology attribute** must be the first attribute defined and is placed at the top of the listing.

**Syntax**

```
technology (name) ;
```

**Example**

```
library (my_library) {
    technology (cmos);
    ...
}
**delay_model Attribute**

This attribute indicates the delay model to use in delay calculations. Valid value is `table_lookup`.

The `delay_model` attribute must follow the `technology` attribute; if the `technology` attribute is not present, the `delay_model` attribute must be the first attribute in the library.

**Syntax**

```
delay_model : value ;
```

**Example**

```
library (my_library) {
    delay_model : table_lookup;
    ...
}
```

---

**bus_naming_style Attribute**

This attribute defines the naming convention for buses in the library.

**Syntax**

```
bus_naming_style : "string";
```

**string**

Contains alphanumeric characters, braces, underscores, dashes, or parentheses. Must contain one `%s` symbol and one `%d` symbol. The `%s` and `%d` symbols can appear in any order with at least one nonnumeric character in between.

The colon character is not allowed in a `bus_naming_style` attribute value because the colon is used to denote a range of bus members. You construct a complete bused-pin name by using the name of the owning bus and the member number. The owning bus name is substituted for the `%s`, and the member number replaces the `%d`.

---

**Delay and Slew Attributes**

This section describes the attributes to set the values of the input and output pin threshold points that are used to model delay and slew.

Delay is the time it takes for the output signal voltage, which is falling from 1 to 0, to fall to the threshold point set with the `output_threshold_pct_fall` attribute after the input signal voltage, which is falling from 1 to 0, has fallen to the threshold point set with the `input_threshold_pct_fall` attribute (see Figure 2-1).
Delay is also the time it takes for the output signal, which is rising from 0 to 1, to rise to the threshold point set with the `output_threshold_pct_rise` attribute after the input signal, which is rising from 0 to 1, has risen from 0 to the threshold point set with the `input_threshold_pct_rise` attribute.

**Figure 2-1  Delay Modeling for Falling Signal**

Slew is the time it takes for the voltage value to fall or rise between two designated threshold points on an input, an output, or a bidirectional port. The designated threshold points must fall within a voltage falling from 1 to 0 or rising from 0 to 1.

Use the following attributes to enter the two designated threshold points to model the time for voltage falling from 1 to 0:

- `slew_lower_threshold_pct_fall`
- `slew_upper_threshold_pct_fall`

Use the following attributes to enter the two designated threshold points to model the time for voltage rising from 0 to 1:

- `slew_lower_threshold_pct_rise`
- `slew_upper_threshold_pct_rise`
Figure 2-2  Slew Modeling Example

Specifying Delay and Slew Attributes in Voltage Range

When partial voltage swing is defined in a pin or a timing group, the nonlinear delay model (NLDM) data in that group is for the partial swing. You must apply the following threshold and trip point attributes only in the voltage range specified using the output_signal_level_low and the output_signal_level_high attributes. For more information, see “output_signal_level_low and output_signal_level_high Attributes” on page 9-15.”

input_threshold_pct_fall Simple Attribute

Use the input_threshold_pct_fall attribute to set the value of the threshold point on an input pin signal falling from 1 to 0. This value is used to model the delay of a signal transmitting from an input pin to an output pin.

Syntax

input_threshold_pct_fall : trip_pointvalue ;
trip_point

A floating-point number between 0.0 and 100.0 that specifies the threshold point of an input pin signal falling from 1 to 0. The default is 50.0.

Example

input_threshold_pct_fall : 60.0 ;

input_threshold_pct_rise Simple Attribute

Use the input_threshold_pct_rise attribute to set the value of the threshold point on an input pin signal rising from 0 to 1. This value is used to model the delay of a signal transmitting from an input pin to an output pin.

Syntax

input_threshold_pct_rise : trip_pointvalue ;

trip_point

A floating-point number between 0.0 and 100.0 that specifies the threshold point of an input pin signal rising from 0 to 1. The default is 50.0.

Example

input_threshold_pct_rise : 40.0 ;

output_threshold_pct_fall Simple Attribute

Use the output_threshold_pct_fall attribute to set the value of the threshold point on an output pin signal falling from 1 to 0. This value is used to model the delay of a signal transmitting from an input pin to an output pin.

Syntax

output_threshold_pct_fall : trip_pointvalue ;

trip_point

A floating-point number between 0.0 and 100.0 that specifies the threshold point of an output pin signal falling from 1 to 0. The default is 50.0.

Example

output_threshold_pct_fall : 40.0 ;
output_threshold_pct_rise Simple Attribute

Use the `output_threshold_pct_rise` attribute to set the value of the threshold point on an output pin signal rising from 0 to 1. This value is used to model the delay of a signal transmitting from an input pin to an output pin.

Syntax

```plaintext
output_threshold_pct_rise : trip_pointvalue ;
```

`trip_point`

A floating-point number between 0.0 and 100.0 that specifies the threshold point of an output pin signal rising from 0 to 1. The default is 50.0.

Example

```plaintext
output_threshold_pct_rise : 40.0 ;
```

slew_derate_from_library Simple Attribute

Use the `slew_derate_from_library` attribute to specify how the transition times found in the library need to be derated to match the transition times between the characterization trip points.

Syntax

```plaintext
slew_derate_from_library : deratenvalue ;
```

`derate`

A floating-point number between 0.0 and 1.0. The default is 1.0.

Example

```plaintext
slew_derate_from_library : 0.5 ;
```

slew_lower_threshold_pct_fall Simple Attribute

Use the `slew_lower_threshold_pct_fall` attribute to set the value of the lower threshold point that is used to model the delay of a pin falling from 1 to 0.

Syntax

```plaintext
slew_lower_threshold_pct_fall : trip_point_value ;
```
trip_point

A floating-point number between 0.0 and 100.0 that specifies the lower threshold point that is used to model the delay of a pin falling from 1 to 0. The default is 20.0.

Example

slew_lower_threshold_pct_fall : 30.0 ;

slew_lower_threshold_pct_rise Simple Attribute

Use the `slew_lower_threshold_pct_rise` attribute to set the value of the lower threshold point that is used to model the delay of a pin rising from 0 to 1.

Syntax

slew_lower_threshold_pct_rise : `trip_point`_`value` ;

trip_point

A floating-point number between 0.0 and 100.0 that specifies the lower threshold point that is used to model the delay of a pin rising from 0 to 1. The default is 20.0.

Example

slew_lower_threshold_pct_rise : 30.0 ;

slew_upper_threshold_pct_fall Simple Attribute

Use the `slew_upper_threshold_pct_fall` attribute to set the value of the upper threshold point that is used to model the delay of a pin falling from 1 to 0.

Syntax

slew_upper_threshold_pct_fall : `trip_point`_`value` ;

trip_point

A floating-point number between 0.0 and 100.0 that specifies the upper threshold point that is used to model the delay of a pin falling from 1 to 0. The default is 80.0.

Example

slew_upper_threshold_pct_fall : 70.0 ;

slew_upper_threshold_pct_rise Simple Attribute

Use the `slew_upper_threshold_pct_rise` attribute to set the value of the upper threshold point that is used to model the delay of a pin rising from 0 to 1.
Syntax

slew_upper_threshold_pct_rise : trip_point\text{value} ;

\textit{trip\_point}

A floating-point number between 0.0 and 100.0 that specifies the upper threshold point that is used to model the delay of a pin rising from 0 to 1. The default is 80.0.

Example

slew_upper_threshold_pct_rise : 70.0 ;

---

**Defining Units**

Use these library-level attributes to define units:

- \textit{time\_unit}
- \textit{voltage\_unit}
- \textit{current\_unit}
- \textit{pulling\_resistance\_unit}
- \textit{capacitive\_load\_unit}
- \textit{leakage\_power\_unit}

The unit attributes identify the units of measure, such as nanoseconds or picofarads, used in the library definitions.

---

**time\_unit Attribute**

Use this attribute to identify the physical time unit used in the generated library.

Syntax

\textit{time\_unit} : \textit{unit} ;

\textit{unit}

Valid values are 1ps, 10ps, 100ps, and 1ns. The default is 1ns.

Example

time_unit : "10ps";
voltage_unit Attribute

Use this attribute to scale the contents of the input_voltage and output_voltage groups. Additionally, the voltage attribute in the operating_conditions group represents values in the voltage units.

Syntax

```
voltage_unit : unit ;
```

`unit`

Valid values are 1mV, 10mV, 100mV, and 1V. The default is 1V.

Example

```
voltage_unit : "100mV";
```

current_unit Attribute

This attribute specifies the unit for the drive current that is generated by output pads. The pulling_current attribute for a pull-up or pull-down transistor also represents its values in this unit.

Syntax

```
current_unit : valueenum ;
```

`value`

The valid values are 1uA, 10uA, 100uA, 1mA, 10mA, 100mA, and 1A. No default exists for the current_unit attribute if the attribute is omitted.

Example

```
current_unit : "1mA";
```

pulling_resistance_unit Attribute

The pulling_resistance_unit attribute defines pulling resistance unit values for pull-up and pull-down devices.

Syntax

```
pulling_resistance_unit : "unit" ;
```
unit

Valid unit values are 1ohm, 10ohm, 100ohm, and 1kohm. No default exists for pulling_resistance_unit if the attribute is omitted.

Example
pulling_resistance_unit : "10ohm";

capacitive_load_unit Attribute

This attribute specifies the unit for all capacitance values within the technology library, including default capacitances, max_fanout capacitances, pin capacitances, and wire capacitances.

Syntax
capacitive_load_unit (valuefloat,unitenum) ;

value
A floating-point number.

unit
Valid values are ff and pf.

Example
capacitive_load_unit(1,pf);

leakage_power_unit Attribute

This attribute indicates the units of the power values in the library. If this attribute is missing, the leakage-power values are expressed without units.

Syntax
leakage_power_unit : valueenum ;

value
Valid values are 1mW, 100mW, 10mW, 1mW, 100nW, 10nW, 1nW, 100pW, 10pW, and 1pW.

Example
leakage_power_unit : 100uW;
Building Environments


The environment attributes include various attributes and tasks, covered in the following sections:

- Library-Level Default Attributes
- Defining Operating Conditions
- Defining Wire Load Groups
- Specifying Delay Scaling Attributes
Library-Level Default Attributes

Global defaults are set at the library level. You can override many of these defaults.

Setting Default Cell Attributes

The following attributes are defaults that apply to all cells in a library.

**default_cell_leakage_power Simple Attribute**

Indicates the default leakage power for those cells that do not have the cell_leakage_power attribute. This attribute must be a nonnegative floating-point number. If it is not defined, this attribute defaults to 0.0.

**Example**

default_cell_leakage_power : 0.5;

Setting Default Pin Attributes

Default pin attributes apply to all pins in a library and deal with timing. How you define default timing attributes in your library depends on the timing delay model you use.

These are the defaults that apply to all pins in a library.

default_inout_pin_cap : value float;

Sets a default for capacitance for all I/O pins in the library.

default_input_pin_cap : value float;

Sets a default for capacitance for all input pins in the library.

default_output_pin_cap : value float;

Sets a default for capacitance for all output pins in the library.

default_max_fanout : value float;

Sets a default for max_fanout for all output pins in the library.

default_max_transition : value float;

Sets a default for max_transition for all output pins in the library.

default_fanout_load : value float;

Sets a default for fanout_load for all input pins in the library.
The following example shows the default pin attributes in a CMOS library:

Example 3-1   Default Pin Attributes for a CMOS Library

library (example) {
  ...  
  /* default pin attributes */  
  default_inout_pin_cap          :  1.0 ; 
  default_input_pin_cap          :  1.0 ; 
  default_output_pin_cap         :  0.0 ; 
  default_fanout_load            :  1.0 ; 
  default_max_fanout             :  10.0 ; 
  default_max_transition         :  15.0 ; 
  ...  
}

Setting Wire Load Defaults

Use the following library-level attributes to set wire load defaults.

default_wire_load Attribute

Assigns the defaults to the wire_load group, unless you assign a different value for wire_load before compiling the design.

Syntax

default_wire_load : wire_load_name;

Example

default_wire_load : WL1;

default_wire_load_capacitance Attribute

Specifies a value for the default wire load capacitance.

Syntax

default_wire_load_capacitance : value;

Example

default_wire_load_capacitance : .05;

default_wire_load_resistance Attribute

Specifies a value for the default wire load resistance.
Syntax
default_wire_load_resistance : value;

Example
default_wire_load_resistance : .067;

default_wire_load_area Attribute
Specifies a value for the default wire load area.

Syntax
default_wire_load_resistance : value;

Example
default_wire_load_area : 0.33;

Setting Other Environment Defaults
Use the following library-level attributes to set other environment defaults.

default_operating_conditions Attribute
Assigns a default operating_conditions group name for the library. It must be specified after all operating_conditions groups. If this attribute is not used, nominal operating conditions apply. See “Defining Operating Conditions” on page 3-5.

Syntax
default_operating_conditions : operating_condition_name;

Example
default_operating_conditions : WCCOM1;

default_connection_class Attribute
Sets a default for connection_class for all pins in a library.

Example
default_connection_class : name1 [name2 name3 ...];
Examples of Library-Level Default Attributes

In Example 3-2, the wire_load and operating_conditions group statements illustrate the requirement that group names that are referred to by the default attributes, such as WL1 and OP1, must be defined in the library.

Example 3-2 Setting Library-Level Default Attributes for a CMOS Library

```
library (example) {
    ...
    /* default cell attributes */
    default_cell_leakage_power : 0.2;
    /* default pin attributes */
    default_inout_pin_cap : 1.0;
    default_input_pin_cap : 1.0;
    default_output_pin_cap : 0.0;
    default_fanout_load : 1.0;
    default_max_fanout : 10.0;
    wire_load (WL1) {
        ...
    }
    operating_conditions (OP1) {
        ...
    }
    default_wire_load : WL1;
    default_operating_conditions : OP1;
    default_wire_load_mode : enclosed;
    ...
}
```

Defining Operating Conditions

The following section explains how to define and determine various operating conditions for a technology library.

operating_conditions Group

An operating_conditions group is defined in a library group.

Syntax

```
library (lib_name) {
    operating_conditions (name) {
        ...
    }
```
conditions description ...
}
}

name

Identifies the set of operating conditions. Names of all operating_conditions groups and wire_load groups must be unique within a library.

The operating_conditions groups are useful for testing timing and other characteristics of your design in predefined simulated environments. The following attributes are defined in an operating_conditions group:

process : multiplier ;

The scaling factor accounts for variations in the outcome of the actual semiconductor manufacturing steps, typically 1.0 for most technologies. The multiplier is a floating-point number from 0 through 100.

process_label : "name" ;

The process name of the current process. The value is a string.

temperature : value ;

The ambient temperature in which the design is to operate. The value is a floating-point number.

voltage : value ;

The operating voltage of the design, typically 5 volts for a CMOS library. The value is a floating-point number from 0 through 1,000, representing the absolute value of the actual voltage.

Note:
Define voltage units consistently.

tree_type : model ;

The definition for the environment interconnect model.

The model is one of the following three models:

- best_case_tree
  Models the case in which the load pin is physically adjacent to the driver. In the best case, all wire capacitance is incurred but none of the wire resistance must be overcome.
Defining Power Supply Cells

Use the `power_supply` group to model multiple power supply cells.

**power_supply group**

The `power_supply` group captures all nominal information about voltage variation. It is defined before the `operating_conditions` group and before the `cell` groups. All the power supply names defined in the `power_supply` group exist in the `operating_conditions` group. Define the `power_supply` group at the library level.

**Syntax**

```plaintext
power_supply () {
    default_power_rail : string;
    power_rail (string, float);
    ... 
}
```

**Example**

```plaintext
power_supply () {
    default_power_rail : VDD0;
    power_rail (VDD1, 5.0);
    power_rail (VDD2, 3.3);
}
```

Defining Wire Load Groups

Use the `wire_load group` and the `wire_load_selection` group to specify values for the `capacitance_factor`, `resistance_factor`, `area_factor`, `slope`, and `fanout_length` you want to apply to the wire delay model for different sizes of circuitry.
wire_load Group

The wire_load group has an extended fanout_length complex attribute. Define the wire_load group at the library level.

Syntax
wire_load(name){
    resistance : value ;
    capacitance : value ;
    area : value ;
    slope : value ;
    fanout_length(fanout_int, length_float, \
                    average_capacitance_float, standard_deviation_float, \
                    number_of_nets_int);
}

In a wire_load group, you define the estimated wire length as a function of fanout. You can also define scaling factors to derive wire resistance, capacitance, and area from a given length of wire.

You can define any number of wire_load groups in a technology library, but all wire_load groups and operating_conditions groups must have unique names.

You can define the following simple attributes in a wire_load group:
resistance : value ;
    Specifies a floating-point number representing wire resistance per unit length of interconnect wire.

capacitance : value ;
    Specifies a floating-point number representing capacitance per unit length of interconnect wire.

area : value ;
    Specifies a floating-point number representing the area per unit length of interconnect wire.

slope : value ;
    Specifies a floating-point number representing slope. This attribute characterizes linear fanout length behavior beyond the scope of the longest length described by the fanout_length attributes.

You can define the following complex attribute in a wire_load group:
fanout_length (fanout_int, length_float, average_capacitance_float \ standard_deviation_float, number_of_nets_int);

fanout_length is a complex attribute that defines values that represent fanout and length. The fanout value is an integer; length is a floating-point number.

When you create a wire load manually, define only fanout and length.

You must define at least one pair of fanout and length points per wire load model. You can define as many additional pairs as necessary to characterize the fanout-length behavior you want.

interconnect_delay (template_name)
{values(float,...float,...float,...float,...) ; }

The interconnect_delay group specifies the lookup table template and the wire delay values.

Specify the interconnect_delay values.

To overwrite the default index values, specify the new index values before the interconnect_delay values, as shown here.

wire_load_table Group

You can use the wire_load_table group to estimate accurate connect delay. Compared to the wire_load group, this group is more flexible, because wire capacitance and resistance no longer have to be strictly proportional to each other. In some cases, this results in more-accurate connect delay estimates.

Syntax

wire_load_table(name_string) {
  fanout_length(fanout_int, length_float);
  fanout_capacitance(fanout_int, capacitance_float);
  fanout_resistance(fanout_int, resistance_float);
  fanout_area(fanout_int, area_float);
}

In the wire_load group, the fanout_capacitance, fanout_resistance, and fanout_area values represent per-length coefficients. In the wire_load_table group the values are exact.
Specifying Delay Scaling Attributes

These k-factors (attributes that begin with k_) are multipliers that scale defined library values, taking into consideration the effects of changes in process, temperature, and voltage.

To model the effects of process, temperature, and voltage variations on circuit timing, use the following:

- k-factors that apply to the entire library and are defined at the library level.
- User-selected operating conditions that override the values in the library for an individual cell.

Pin and Wire Capacitance Factors

The pin and wire capacitance factors scale the capacitance of a pin or wire according to process, temperature, and voltage variations. Each attribute gives the multiplier for a certain portion of the capacitance of a pin or a wire. In the following syntax, multiplier is a floating-point number:

- $k_{\text{process pin cap}} : \text{multiplier}$
  - Scaling factor applied to pin capacitance to model process variation.
- $k_{\text{process wire cap}} : \text{multiplier}$
  - Scaling factor applied to wire capacitance to model process variation.
- $k_{\text{temp pin cap}} : \text{multiplier}$
  - Scaling factor applied to pin capacitance to model temperature variation.
- $k_{\text{temp wire cap}} : \text{multiplier}$
  - Scaling factor applied to wire capacitance to model temperature variation.
- $k_{\text{volt pin cap}} : \text{multiplier}$
  - Scaling factor applied to pin capacitance to model voltage variation.
- $k_{\text{volt wire cap}} : \text{multiplier}$
  - Scaling factor applied to wire capacitance to model voltage variation.

Scaling Factors Associated With the Nonlinear Delay Model

The CMOS nonlinear delay model scaling factors scale the delay based on the variation in process, temperature, and voltage. The following scaling factors apply to the CMOS nonlinear delay model:
• k_process_cell_rise
• k_temp_cell_rise
• k_volt_cell_rise
• k_process_cell_fall
• k_temp_cell_fall
• k_volt_cell_fall
• k_process_rise_propagation
• k_temp_rise_propagation
• k_volt_rise_propagation
• k_process_fall_propagation
• k_temp_fall_propagation
• k_volt_fall_propagation
• k_process_rise_transition
• k_temp_rise_transition
• k_volt_rise_transition
• k_process_fall_transition
• k_temp_fall_transition
• k_volt_fall_transition
Defining Core Cells

Cell descriptions are a major part of a technology library. They provide information about the area, function, and timing of each component in an ASIC technology.

Defining core cells for CMOS technology libraries involves the following concepts and tasks described in this chapter:

- Defining cell Groups
- Defining Bused Pins
- Defining Signal Bundles
- Defining Layout-Related Multibit Attributes
- Defining Multiplexers
- Defining Decoupling Capacitor Cells, Filler Cells, and Tap Cells
Defining cell Groups

A cell group defines a single cell in the technology library.

This section discusses the attributes in a cell group.

For information about groups within a cell, see the following sections in this chapter:

• “Defining Bused Pins” on page 4-39
• “Defining Signal Bundles” on page 4-46

See Chapter 6, “Defining Test Cells,” for a test cell with a test_cell group. See Example 4-1 on page 4-8 for an example cell description.

cell Group

The cell group statement gives the name of the cell being described. It appears at the library group level, as shown here:

library (lib_name) {
  ...
  cell( name ) {
    ... cell description ...
  }
  ...
}

Use a name that corresponds to the name the ASIC vendor uses for the cell. When naming cells, remember that names are case-sensitive. For example, the cell names, AND2, and2, and And2 are all different. Cell names beginning with a number must be enclosed in quotation marks.

To create the cell group for the AND2 cell, use this syntax:

cell( AND2 ) {
  ... cell description ...
}

To describe a CMOS cell group, you use the type group and these attributes:

• area
• bundle (See “Defining Signal Bundles” on page 4-46.)
• bus (See “Defining Bused Pins” on page 4-39.)
• cell_footprint
• clock_gating_integrated_cell
• contention_condition
• is_clock_gating_cell
• map_only
• pad_cell
• pad_type
• pin_equal
• pin_opposite
• preferred

---

**area Attribute**

This attribute specifies the cell area.

**Example**

```
area : 2.0;
```

For unknown or undefined (black box) cells, the `area` attribute is optional. Unless a cell is a pad cell, it should have an `area` attribute. Pad cells should be given an area of 0.0, because they are not used as internal gates.

---

**cell_footprint Attribute**

This attribute assigns a footprint class to a cell.

**Example**

```
cell_footprint : 5MIL ;
```

Characters in the string are case-sensitive.

Use this attribute to assign the same footprint class to all cells that have the same layout boundary. Cells with the same footprint class are considered interchangeable and can be swapped during in-place optimization.
clock_gating_integrated_cell Attribute

An integrated clock-gating cell is a cell that you or your library developer creates to use especially for clock gating. The cell integrates the various combinational and sequential elements of a clock gate into a single cell that is compiled into gates and located in the technology library.

Consider using an integrated clock-gating cell if you are experiencing timing problems caused by the introduction of randomly chosen logic on your clock line.

Use the `clock_gating_integrated_cell` attribute to specify a value that determines the integrated cell functionality to be used by the clock-gating tools.

**Syntax**

```
clock_gating_integrated_cell : generic|valueid;
```

**generic**

When you specify the value `generic`, the actual type of clock gating integrated cell structure is determined by accessing the function specified on the library pin.

**Note:**
Statetables and state functions should not be used. Use latch groups with function groups instead.

**value**

A concatenation of up to four strings that describe the cell’s functionality to the clock-gating tools:

- The first string specifies the type of sequential element you want. The options are latch-gating logic and none.
- The second string specifies whether the logic is appropriate for rising- or falling-edge-triggered registers. The options are `posedge` and `negedge`.
- The third (optional) string specifies whether you want test-control logic located before or after the latch or not at all. The options for cells set to latch are `precontrol` (before), `postcontrol` (after), or `no entry`. The options for cells set to no gating logic are `control` and `no entry`.
- The fourth (optional) string, which exists only if the third string does, specifies whether you want observability logic or not. The options are `obs` and `no entry`.
Example

clock_gating_integrated_cell : "latch_posedge_precontrol_obs" ;

Table 4-1  Values of the clock_gating_integrated_cell Attribute

<table>
<thead>
<tr>
<th>When the value is</th>
<th>The integrated cell must contain</th>
</tr>
</thead>
<tbody>
<tr>
<td>latch_negedge</td>
<td>Latch-based gating logic.</td>
</tr>
<tr>
<td></td>
<td>Logic appropriate for falling-edge-triggered registers.</td>
</tr>
<tr>
<td>latch_posedge_postcontrol</td>
<td>Latch-based gating logic.</td>
</tr>
<tr>
<td></td>
<td>Logic appropriate for rising-edge-triggered registers.</td>
</tr>
<tr>
<td></td>
<td>Test-control logic located after the latch.</td>
</tr>
<tr>
<td>latch_negedge_precontrol</td>
<td>Latch-based gating logic.</td>
</tr>
<tr>
<td></td>
<td>Logic appropriate for falling-edge-triggered registers.</td>
</tr>
<tr>
<td></td>
<td>Test-control logic located before the latch.</td>
</tr>
<tr>
<td>none_posedge_control_obs</td>
<td>Latch-free gating logic.</td>
</tr>
<tr>
<td></td>
<td>Logic appropriate for rising-edge-triggered registers.</td>
</tr>
<tr>
<td></td>
<td>Test-control logic (no latch).</td>
</tr>
<tr>
<td></td>
<td>Observability logic.</td>
</tr>
</tbody>
</table>

Setting Pin Attributes for an Integrated Cell

The clock-gating tool requires that you set the pins of your integrated cells by using the attributes listed in Table 4-2. Setting some of the pin attributes, such as those for test and observability, is optional.

Table 4-2  Pin Attributes for Integrated Clock-Gating Cells

<table>
<thead>
<tr>
<th>Integrated cell pin name</th>
<th>Data direction</th>
<th>Required attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>in</td>
<td>clock_gate_clock_pin</td>
</tr>
<tr>
<td>enable</td>
<td>in</td>
<td>clock_gate_enable_pin</td>
</tr>
<tr>
<td>test_mode or scan_enable</td>
<td>in</td>
<td>clock_gate_test_pin</td>
</tr>
<tr>
<td>observability</td>
<td>out</td>
<td>clock_gate_obs_pin</td>
</tr>
<tr>
<td>enable_clock</td>
<td>out</td>
<td>clock_gate_out_pin</td>
</tr>
</tbody>
</table>
For details about these pin attributes, see the following sections:

- “clock_gate_clock_pin Attribute” on page 4-13
- “clock_gate_enable_pin Attribute” on page 4-14
- “clock_gate_obs_pin Attribute” on page 4-14
- “clock_gate_out_pin Attribute” on page 4-14
- “clock_gate_test_pin Attribute” on page 4-14

**Setting Timing for an Integrated Cell**

You set both the setup and hold arcs on the enable pin by setting the `clock_gate_enable_pin` attribute for the integrated cell to `true`. The setup and hold arcs for the cell are determined by the edge values you enter for the `clock_gating_integrated_cell` attribute.

**Table 4-3  Edge Values of the `clock_gating_integrated_cell` Attribute With Setup and Hold Arcs**

<table>
<thead>
<tr>
<th>Value</th>
<th>Setup arc</th>
<th>Hold arc</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>latch_posedge</code></td>
<td>rising</td>
<td>rising</td>
</tr>
<tr>
<td><code>latch_negedge</code></td>
<td>falling</td>
<td>falling</td>
</tr>
<tr>
<td><code>none_posedge</code></td>
<td>falling</td>
<td>rising</td>
</tr>
<tr>
<td><code>none_negedge</code></td>
<td>rising</td>
<td>falling</td>
</tr>
</tbody>
</table>

**contention_condition Attribute**

The `contention_condition` attribute specifies the contention conditions for a cell.

Contention is a clash of 0 and 1 signals. In certain cells, it can be a forbidden condition and cause circuits to short.

**Example**

```
contention_condition : "!ap * an" ;
```

**is_macro_cell Attribute**

The `is_macro_cell` attribute identifies whether a cell is a macro cell. If the attribute is set to `true`, the cell is a macro cell. If it is set to `false`, the cell is not a macro cell.
**Example**

is_macro_cell : true;  

---

**pad_cell Attribute**

The `pad_cell` attribute in a cell group identifies the cell as a pad.

**Example**

pad_cell : true;

---

**pin_equal Attribute**

This attribute describes a group of logically equivalent input or output pins in the cell.

---

**pin_opposite Attribute**

This attribute describes functionally opposite (logically inverse) groups of pins in a cell. The `pin_opposite` attribute also incorporates the functionality of `pin_equal`.

**Example**

pin_opposite("Q1 Q2 Q3", "QB1 QB2");

In this example, Q1, Q2, and Q3 are equal; QB1 and QB2 are equal; and the pins of the first group are opposite to the pins of the second group.

**Note:**

Use the `pin_opposite` attribute only in cells without function information or when you want to define required inputs.

---

**type Group**

The `type` group, when defined within a cell, is a type definition local to the cell. It cannot be used outside of the cell.

**Example**

type (bus4) {
    base_type : array;
    data_type : bit;
    bit_width : 4;
    bit_from : 0;
    bit_to : 3;
}
cell Group Example

Example 4-1 shows cell definitions that include some of the CMOS cell attributes described in this section.

Example 4-1 cell Group Example

```liberty
library (cell_example){
  date : "August 14, 2015";
  revision : 2015.03;
  cell (inout){
    pad_cell : true;
    dont_use : true;
    dont_fault : sa0;
    dont_touch : true;
    area : 0;/* pads do not normally consume internal core area */
    cell_footprint : 5MIL;
    pin (A) {
      direction : input;
      capacitance : 0;
    }
    pin (Z) {
      direction : output;
      function : "A";
      timing () {
        ...
      }
    }
  }
  cell(inverter_med){
    area : 3;
    preferred : true;
    pin (A) {
      direction : input;
      capacitance : 1.0;
    }
    pin (Z) {
      direction : output;
      function : "A' ";
      timing () {
        ...
      }
    }
  }
  cell(nand){
    area : 4;
    pin(A) {
      direction : input;
      capacitance : 1;
      fanout_load : 1.0;
    }
  }
}
```
pin(B) {
    direction : input;
    capacitance : 1;
    fanout_load : 1.0;
}

pin (Y) {
    direction : output;
    function : "(A * B)' ";
    timing() {
        ...
    }
}

cell(buff1){
    area : 3;
    pin (A) {
        direction : input;
        capacitance : 1.0;
    }
    pin (Y) {
        direction : output;
        function : "A ";
        timing () {
            ...
        }
    }
}
} /* End of Library */

---

**mode_definition Group**

A **mode_definition** group declares a **mode** group that contains several timing mode values.

**Syntax**

cell(name_string) {
    mode_definition(name_string) {
        mode_value(name1) {
            when : "Boolean expression" ;
            sdf_cond : "sdf_expression_string" ;
        }
        mode_value(name_string) {
            when : "Boolean expression" ;
            sdf_cond :"Boolean expression" ;
        }
    }
}
**Group Statement**

mode_value (name_string) { }

Specifies the condition that a timing arc depends on to activate a path.

**mode_value Group**

The `mode_value` group contains several mode values within a `mode` group. You can optionally put a condition on a mode value. When the condition is true, the `mode` group takes that value.

**Syntax**

mode_value (name_string) { }

**Simple Attributes**

when : "Boolean expression" ;
sdf : "Boolean expression" ;

**when Simple Attribute**

The `when` attribute specifies the condition that a timing arc depends on to activate a path. The valid value is a Boolean expression.

**Syntax**

when : "Boolean expression" ;

**Example**

when: !R;

**sdf_cond Simple Attribute**

The `sdf_cond` attribute supports Standard Delay Format (SDF) file generation and condition matching during back-annotation.

**Syntax**

sdf_cond : "Boolean expression" ;

**Example**

sdf_cond: "R == 0";

**Example 4-2 mode_definition Group Description**

```plaintext
cell(example_cell) {
  ...
  mode_definition(rw) {
    mode_value(read) {
      when : "R";
    }
```
Defining pin Groups

For each pin in a cell, the cell group must contain a description of the pin characteristics. You define pin characteristics in a pin group within the cell group. A pin group often contains a timing group and an internal_power group.

pin Group

You can define a pin group within a cell, test_cell, model, or bus group.

library (lib_name) {
  ... 
  cell (cell_name) {
    ... 
    pin ( name | name_list ) {
      ... pin group description ... 
    }
  }
  cell (cell_name) {
    ... 
    bus (bus_name) {
      ... bus group description ... 
    }
    bundle (bundle_name) {
      ... bundle group description ... 
    }
    pin ( name | name_list ) {
      ... pin group description ... 
    }
  }
}

See “Defining Bused Pins” on page 4-39 for descriptions of bus groups. See “Defining Signal Bundles” on page 4-46 for descriptions of bundle groups.

The pin groups are also valid within test_cell groups. They have different requirements from pin groups in cell, bus, or bundle groups. See “Pins in the test_cell Group” on page 6-3 for specific information and restrictions on describing test pins.
All pin names within a single `cell, bus, or bundle` group must be unique. Names are case-sensitive: pins named `A` and `a` are different pins.

You can describe pins with common attributes in a single `pin` group. If a cell contains two pins with different attributes, two separate `pin` groups are required. Grouping pins with common technology attributes can significantly reduce the size of a cell description that includes many pins.

In the following example, the AND cell has two pins: A and B.

```plaintext
cell (AND) {
  area : 3 ;
  pin (A) {
    direction : input ;
    capacitance : 1 ;
  }
  pin (B) {
    direction : input ;
    capacitance : 1 ;
  }
}
```

Because pins A and B have the same attributes, the cell can also be described as

```plaintext
cell (AND) {
  area : 3 ;
  pin (A,B) {
    direction : input ;
    capacitance : 1 ;
  }
}
```

---

**General pin Group Attributes**

To define a pin, use these general `pin` group attributes:

- `capacitance`
- `clock_gate_clock_pin`
- `clock_gate_enable_pin`
- `clock_gate_obs_pin`
- `clock_gate_out_pin`
- `clock_gate_test_pin`
- `complementary_pin`
- `connection_class`
• direction
• dont_fault
• driver_type
• fall_capacitance
• fault_model
• inverted_output
• is_analog
• pin_func_type
• rise_capacitance
• steady_state_resistance
• test_output_only

capacitance Attribute

The capacitance attribute defines the load of an input, output, inout, or internal pin. The load is defined with a floating-point number, in units consistent with other capacitance specifications throughout the library. Typical units of measure for capacitance include picofarads and standardized loads.

Example

The following example defines the A and B pins in an AND cell, each with a capacitance of one unit.

cell (AND) {
  area : 3 ;
  pin (A,B) {
    direction : input ;
    capacitance : 1 ;
  }
}

If the timing groups in a cell include the output-pin capacitance effect in the intrinsic-delay specification, do not specify capacitance values for the cell’s output pins.

clock_gate_clock_pin Attribute

The clock_gate_clock_pin attribute identifies an input pin connected to a clock signal. Valid values for this attribute are true and false.
Example

```
clock_gate_clock_pin : true;
```

**clock_gate_enable_pin Attribute**

The `clock_gate_enable_pin` attribute identifies an input port connected to an enable signal for nonintegrated clock-gating cells and integrated clock-gating cells.

Valid values for this attribute are `true` and `false`.

Example

```
clock_gate_enable_pin : true;
```

**clock_gate_obs_pin Attribute**

The `clock_gate_obs_pin` attribute identifies an output port connected to an observability signal.

Valid values for this attribute are `true` and `false`.

Example

```
clock_gate_obs_pin : true;
```

**clock_gate_out_pin Attribute**

The `clock_gate_out_pin` attribute identifies an output port connected to an `enable_clock` signal.

Valid values for this attribute are `true` and `false`.

Example

```
clock_gate_out_pin : true;
```

**clock_gate_test_pin Attribute**

The `clock_gate_test_pin` attribute identifies an input port connected to a `test_mode` or `scan_enable` signal.

Valid values for this attribute are `true` and `false`.

Example

```
clock_gate_test_pin : true;
```
complementary_pin Simple Attribute

The complementary_pin attribute supports differential I/O. Differential I/O assumes the following:

• When the noninverting pin equals 1 and the inverting pin equals 0, the signal gets logic 1.
• When the noninverting pin equals 0 and the inverting pin equals 1, the signal gets logic 0.

Syntax
complementary_pin : "string" ;

string

Identifies the differential input data inverting pin whose timing information and associated attributes the noninverting pin inherits. Only one input pin is modeled at the cell level. The associated differential inverting pin is defined in the same pin group.

For details on the fault_model attribute used to define the value when both the complementary pin and the pin that it complements are driven to the same value, see “fault_model Simple Attribute” on page 4-22.

Example
cell (diff_buffer) {
  ...
  pin (A) { /* noninverting pin /
    direction : input ;
    complementary_pin : "DiffA" ;/* inverting pin /
  }
}

connection_class Simple Attribute

The connection_class attribute lets you specify design rules for connections between cells.

Example
connection_class : "internal";

Only pins with the same connection class can be legally connected. For example, you can specify that clock input must be driven by clock buffer cells or that output pads can be driven only by high-drive pad driver cells between the internal logic and the pad. To do this, you assign the same connection class to the pins that must be connected. For the pad example, you attach a given connection class to the pad driver output and the pad input. This attachment makes it invalid to connect another type of cell to the pad.
In Example 4-3, the output_pad cell can be driven only by the pad_driver cell. The pad driver’s input can be connected to internal core logic, because it has the internal connection class.

In Example 4-4, the high_drive_buffer cell can drive internal core cells and pad cells, whereas the low_drive_buffer cell can drive only internal cells.

Example 4-3  Connection Class Example

```plaintext
default_connection_class : "default" ;
cell (output_pad) {
   pin (IN) {
      connection_class : "external_output" ;
   ...
   }
}
cell (pad_driver) {
   pin (OUT) {
      connection_class : "external_output" ;
   ...
   }
   pin (IN) {
      connection_class : "internal" ;
   ...
   }
}
```

Example 4-4  Multiple Connection Classes for a Pin

```plaintext
cell (high_drive_buffer) {
   pin (OUT) {
      connection_class : "internal pad" ;
   ...
   }
}
cell (low_drive_buffer) {
   pin (OUT) {
      connection_class : "internal" ;
   ...
   }
}
cell (pad_cell) {
   pin (IN) {
      connection_class : "pad" ;
   ...
   }
}
cell (internal_cell) {
   pin (IN) {
      connection_class : "internal" ;
   ...
   }
```
direction Attribute

Use this attribute to specify whether the pin being described is an input, output, internal, or bidirectional pin.

Example

direction : output;


driver_type Attribute

Use the optional driver_type attribute to modify the signal on a pin. This attribute specifies a signal mapping mechanism that supports the signal transitions performed by the circuit.

The driver_type attribute tells the application tool to use a special pin-driving configuration for the pin during simulation. A pin without this attribute has normal driving capability by default.

A driver type can be one or more of the following:

pull_up

The pin is connected to DC power through a resistor. If it is a three-state output pin and it is in the Z state, its function is evaluated as a resistive 1 (H). If it is an input or inout pin and the node to which it is connected is in the Z state, it is considered an input pin at logic 1 (H). For a pull-up cell, the pin stays constantly at logic 1 (H).

pull_down

The pin is connected to DC ground through a resistor. If it is a three-state output pin and it is in the Z state, its function is evaluated as a resistive 0 (L). If it is an input or inout pin and the node to which it is connected is in the Z state, it is considered an input pin at logic 0 (L). For a pull-down cell, the pin stays constantly at logic 0 (L).

bus_hold

The pin is a bidirectional pin on a bus holder cell. The pin holds the last logic value present at that pin when no other active drivers are on the associated net. Pins with this driver type cannot have function or three_state statements.

open_drain

The pin is an output pin without a pull-up transistor. Use this driver type only for off-chip output or inout pins representing pads. The pin goes to high impedance (Z) when its function is evaluated as logic 1.
open_source

The pin is an output pin without a pull-down transistor. Use this driver type only for off-chip output or inout pins representing pads. The pin goes to high impedance (Z) when its function is evaluated as logic 0.

resistive

The pin is an output pin connected to a controlled pull-up or pull-down driver with a control port (input). When the control port is disabled, the pull-up or pull-down driver is turned off and has no effect on the pin. When the control port is enabled, a functional value of 0 evaluated at the pin is turned into a weak 0 (L), a functional value of 1 is turned into a weak 1 (H), but a functional value of Z is not affected.

resistive_0

The pin is an output pin connected to DC power through a pull-up driver that has a control port (input). When the control port is disabled, the pull-up driver is turned off and has no effect on the pin. When the control port is enabled, a functional value of 1 evaluated at the pin is turned into a weak 1 (H) but the functional values of 0 and Z are not affected.

resistive_1

The pin is an output pin connected to DC ground through a pull-down driver that has a control port (input). When the control port is disabled, the pull-down driver is turned off and has no effect on the pin. When the control port is enabled, a functional value of 0 evaluated at the pin is turned into a weak 0 (L) but the functional values of 1 and Z are not affected.

Inout pins can have two driver types, one for input and one for output. The only valid combinations are pull_up or pull_down for input and open_drain for output. If you specify only one driver type and it is bus_hold, it is used for both input and output. If the single driver type is not bus_hold, it is used for output. Specify multiple driver types in one entry in this format:

driver_type : "driver_type1 driver_type2" ;

Example

This is an example of a pin connected to a controlled pull-up cell that results in a weak 1 when the control port is enabled.

function : 1;
driver_type : resistive;
three_state_enable : EN;
Interpretation of Driver Types

The driver type specifies one of the following signal modifications:

Resolve the value of Z

These driver types resolve the value of Z on an existing circuit node, implying a constant 0 or 1 signal source. They do not perform a function. Resolution driver types are pull_up, pull_down, and bus_hold.

Transform the signal

These driver types perform an actual function on an input signal, mapping the transition from 0 or 1 to L, H, or Z. Transformation driver types are open_drain, open_source, resistive, resistive_0, and resistive_1.

For output pins, the driver_type attribute is applied after the pin’s functional evaluation. For input pins, this attribute is applied before the signal is used for functional evaluation.

Table 4-4 Signal Mapping and Pin Types for Different Driver Types

<table>
<thead>
<tr>
<th>Driver type</th>
<th>Description</th>
<th>Signal mapping</th>
<th>Applicable pin types</th>
</tr>
</thead>
<tbody>
<tr>
<td>pull_up</td>
<td>Resolution</td>
<td>01Z -&gt; 01H</td>
<td>in, out</td>
</tr>
<tr>
<td>pull_down</td>
<td>Resolution</td>
<td>01Z -&gt; 01L</td>
<td>in, out</td>
</tr>
<tr>
<td>bus_hold</td>
<td>Resolution</td>
<td>01Z -&gt; 01S</td>
<td>inout</td>
</tr>
<tr>
<td>open_drain</td>
<td>Transformation</td>
<td>01Z -&gt; 0ZZ</td>
<td>out</td>
</tr>
<tr>
<td>open_source</td>
<td>Transformation</td>
<td>01Z -&gt; Z1Z</td>
<td>out</td>
</tr>
<tr>
<td>resistive</td>
<td>Transformation</td>
<td>01Z -&gt; LHZ</td>
<td>out</td>
</tr>
<tr>
<td>resistive_0</td>
<td>Transformation</td>
<td>01Z -&gt; 0HZ</td>
<td>out</td>
</tr>
<tr>
<td>resistive_1</td>
<td>Transformation</td>
<td>01Z -&gt; L1Z</td>
<td>out</td>
</tr>
</tbody>
</table>

Signal Mapping:

0 and 1 represent strong logic 0 and logic 1 values
L represents a weak logic 0 value
H represents a weak logic 1 value
Z represents high impedance
S represents the previous state

Interpreting Bus Holder Driver Type

Figure 4-1 01Z to 01S Signal Mapping for bus_hold Driver Type

For bus_hold driver types, a three-state buffer output value of 0 changes the bus value to 0. Similarly, a three-state buffer output value of 1 changes the bus value to 1. However, when the output of the three-state buffer is Z, the bus holds its previous value (S), which can be 0, 1, or Z. In other words, the buffer output value of Z is resolved to the previous value of the bus.

Modeling Pull-Up and Pull-Down Cells

Example 4-5 is the description of the pull-up resistor cell in Figure 4-2.

Example 4-5 Description of a Pull-Up Cell Transistor

Example 4-6 describes an output pin with a pull-up resistor and the bidirectional pin on a bus holder cell.
Example 4-6  Pin Driver Type Specifications

```plaintext
pin(Y) {
  direction : output ;
  driver_type : pull_up ;
  pulling_resistance : 10000 ;
  function : "IO" ;
  three_state : "OE" ;
}
cell (bus_hold) {
  pin(Y) {
    direction : inout ;
    driver_type : bus_hold ;
  }
}
```

Bidirectional pads can often require one driver type for the output behavior and another associated with the input. For this case, you can define multiple driver types in one `driver_type` attribute:

```plaintext
driver_type : "open_drain pull_up" ;
```

Note:

- An n-channel open-drain pad is flagged with `open_drain`, and a p-channel open-drain pad is flagged with `open_source`.

**fall_capacitance Attribute**

Defines the load for an input and inout pin when its signal is falling.

Setting a value for the `fall_capacitance` attribute requires that a value for the `rise_capacitance` also be set, and setting a value for the `rise_capacitance requires that a value for the `fall_capacitance` also be set.

**Syntax**

```plaintext
fall_capacitance : float ;
```

**float**

A floating-point number in units consistent with other capacitance specifications throughout the library. Typical units of measure for `fall_capacitance` include picofarads and standardized loads.

The following example defines the A and B pins in an AND cell, each with a `fall_capacitance` of one unit, a `rise_capacitance` of two units, and a capacitance of two units.

**Example**

```plaintext
cell (AND) {
```
area : 3 ;
vhdl_name : "AND2" ;
pin (A, B) {
    direction : input ;
    fall_capacitance : 1 ;
    rise_capacitance : 2 ;
    capacitance : 2 ;
}
}

fault_model Simple Attribute

The differential I/O feature enables an input noninverting pin to inherit the timing information and all associated attributes of an input inverting pin in the same pin group designated with the complementary_pin attribute.

If you enter a fault_model attribute, you must designate the inverted pin associated with the noninverting pin, using the complementary_pin attribute.

For details on the complementary_pin attribute, see “complementary_pin Simple Attribute” on page 4-15.

Syntax

fault_model : "two-value string" ;

two-value string

Two values that define the value of the differential signals when both inputs are driven to the same value. The first value represents the value when both input pins are at logic 0; the second value represents the value when both input pins are at logic 1. Valid values for the two-value string are any two-value combinations of 0, 1, and x.

If you do not enter a fault_model attribute value, the signal pin value goes to x when both input pins are 0 or 1.

Example

cell (diff_buffer) {
    ...
    pin (A) { /* noninverting pin */
        direction : input ;
        complementary_pin : ("DiffA")
        fault_model : "1x" ;
    }
}
Table 4-5 shows how testing interprets the complementary pin values for this example:

<table>
<thead>
<tr>
<th>Pin A (noninverting pin)</th>
<th>DiffA (complementary_pin)</th>
<th>Resulting signal pin value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>

**inverted_output Attribute**

The `inverted_output` attribute is a Boolean attribute that you can set for any output port. It is a required attribute only for sequential cells.

Set this attribute to false for noninverting output, which is variable1 or IQ for flip-flop or latch groups. Set this attribute to true for inverting output, which is variable2 or IQN for flip-flop or latch groups.

**Example**

```plaintext
pin(Q) {
    function : "IQ";
    internal_node : "IQ";
    inverted_output : false;
}
```

This attribute affects the internal interpretation of the state table format used to describe a sequential cell.

**is_analog Attribute**

The `is_analog` attribute identifies an analog signal pin as analog so it can be recognized by tools. The valid values for `is_analog` are true and false. Set the `is_analog` attribute to true at the pin level to specify that the signal pin is analog.

**Syntax**

The syntax for the `is_analog` attribute is as follows:

```plaintext
cell (cell_name) {
    ...
    pin (pin_name) {
        is_analog: true | false ;
    }
}
```
Example

The following example identifies the pin as an analog signal pin.

```liberty
pin(Analog) {
    direction : input;
    capacitance : 1.0 ;
    is_analog : true;
}
```

**pin_func_type Attribute**

This attribute describes the functions of a pin.

**Example**

```liberty
pin_func_type : clock_enable;
```

With the `pin_func_type` attribute, you avoid the checking and modeling caused by incomplete timing information about the enable pin. The information in this attribute defines the clock as the clock-enabling mechanism (that is, the clock-enable pin). This attribute also specifies whether the active level of the enable pin of latches is high or low and whether the active edge of the flip-flop clock is rising or falling.

**rise_capacitance Attribute**

Defines the load for an input and inout pin when its signal is rising.

Setting a value for the `rise_capacitance` attribute requires that a value for `fall_capacitance` also be set, and setting a value for `fall_capacitance` requires that a value for `rise_capacitance` also be set.

**Syntax**

```liberty
rise_capacitance : float ;
```

**float**

A floating-point number in units consistent with other capacitance specifications throughout the library. Typical units of measure for `rise_capacitance` include picofarads and standardized loads.

The following example defines the A and B pins in an AND cell, each with a `fall_capacitance` of one unit, a `rise_capacitance` of two units, and a capacitance of two units.
Example
cell (AND) {
    area : 3 ;
    vhdl_name : "AND2" ;
    pin (A, B) {
        direction : input ;
        fall_capacitance : 1 ;
        rise_capacitance : 2 ;
        capacitance : 2 ;
    }
}

steady_state_resistance Attributes
When there are multiple drivers connected to an interconnect network driven by library cells and there is no direct current path between them, the driver resistances could take on different values.

Use the following attributes for more-accurate modeling of steady state driver resistances in library cells.

• steady_state_resistance_above_high
• steady_state_resistance_below_low
• steady_state_resistance_high
• steady_state_resistance_low

Example
steady_state_resistance_above_high : 200 ;

test_output_only Attribute
This attribute is an optional Boolean attribute that you can set for any output port described in statetable format.

In ff or latch format, if a port is to be used for both function and test, you provide the functional description using the function attribute. If a port is to be used for test only, you omit the function attribute.

Regardless of ff or latch statetable, the test_output_only attribute takes precedence over the functionality.

In statetable format, however, a port always has a functional description. Therefore, if you want to specify that a port is for test only, you set the test_output_only attribute to true.

Example
pin (my_out) {
direction : output ;
signal_type : test_scan_out ;
test_output_only : true ;
}

Describing Design Rule Checks
To define design rule checks, use the following pin group attributes and group:

- fanout_load attribute
- max_fanout attribute
- min_fanout attribute
- max_transition attribute
- max_trans group
- min_transition attribute
- max_capacitance attribute
- max_cap group
- min_capacitance attribute
- cell_degradation group

fanout_load Attribute
The fanout_load attribute gives the fanout load value for an input pin.

Example
fanout_load : 1.0;

The sum of all fanout_load attribute values for input pins connected to a driving (output) pin must not exceed the max_fanout value for that output pin.
max_fanout Attribute

This attribute defines the maximum fanout load that an output pin can drive.

Example

```liberty
pin(Q) {
  direction : output;
  max_fanout : 10;
}
```

Some designs have limitations on input load that an output pin can drive regardless of any loading contributed by interconnect metal layers. To limit the number of inputs on the same net driven by an output pin, define a `max_fanout` value for each output pin and a `fanout_load` on each input pin in a cell. (See Figure 4-3.)

To determine `max_fanout`, find the smallest loading of any input to a cell in the library and use that value as the standard load unit for the entire library. Usually the smallest buffer or inverter has the lowest input pin loading value. Use some multiple of the standard value for the fanout loads of the other cells.

Although you can use capacitance as the unit for your `max_fanout` and `fanout_load` specifications, it should be used to constrain routability requirements.

min_fanout Attribute

This attribute defines the minimum fanout load that an output or inout pin can drive. The sum of fanout cannot be less than the minimum fanout value.

Example

```liberty
pin(Q) {
  direction : output;
  min_fanout : 2.0;
}
```
**max_transition Attribute**

This attribute defines a design rule constraint for the maximum acceptable transition time of an input or output pin.

**Example**

```python
pin(A) {
    direction : input;
    max_transition : 4.2;
}
```

You can specify `max_transition` at three levels: at the library level, at the pin level, and on the command line.

With an output pin, `max_transition` is used only to drive a net for which the cell can provide a transition time at least as fast as the defined limit.

With an input pin, `max_transition` indicates that the pin cannot be connected to a net that has a transition time greater than the defined limit.

In the following example, the cell that contains pin Q cannot be used to drive a net for which the cell cannot provide a transition time faster than 5.2:

```python
pin(Q) {
    direction : output ;
    max_transition : 5.2 ;
}
```

**max_trans Group**

The `max_trans` group specifies the maximum transition time of an input, output, or inout pin as a function of operating frequency of the cell, input transition time, and output load. Use the `max_trans` group instead of the `max_transition` attribute to include these effects on the maximum transition time. When both the `max_trans` group and `max_transition` attribute are present, the `max_trans` group overrides the `max_transition` attribute.

To model the effect of operating frequency, input transition time, and output load on the maximum transition, define the lookup table template for the `max_trans` group at the library level.

To define the lookup table template, use the `maxtrans_lut_template` group at the library level. The `maxtrans_lut_template` group can have the variables, `variable_1`, `variable_2`, and `variable_3`. The valid values of the `variable_1`, `variable_2`, and `variable_3` variables are `frequency`, `input_transition_time`, and `total_output_net_capacitance` respectively.

The one-dimensional lookup table consists of the maximum transition values for different values of `frequency`. Similarly, the two-dimensional lookup table consists of the maximum transition values for different values of `frequency` and `input_transition_time` and so on.
The following syntax shows the maximum transition model:

```liberty
library (library_name) {
  delay_model : table_lookup;
  ...
  maxtrans_lut_template (template_name1) { /*1-D LUT template*/
    variable_1 : frequency;
    index_1  { "float, ... float" };
  }

  maxtrans_lut_template (template_name2) { /*2-D LUT template*/
    variable_1 : frequency;
    variable_2 : input_transition_time;
    index_1  { "float, ... float" };
    index_2  { "float, ... float" };
  }

  maxtrans_lut_template (template_name3) { /*3-D LUT template*/
    variable_1 : frequency;
    variable_2 : input_transition_time;
    variable_3 : total_output_net_capacitance;
    index_1  { "float, ... float" };
    index_2  { "float, ... float" };
    index_3  { "float, ... float" };
  }

  cell (cell_name) {
    ...
    pin (pin_name) { /* input pin */
      ...
      max_trans (template_name1) {
        index_1 ("float, ... float");
        values ("float, ... float");
      }
      ...
    } /* pin */
    ...
    pin (pin_name) { /* input pin */
      ...
      max_trans (template_name2) { /* output or inout pin */
        index_1 ("float, ... float");
        index_2 ("float, ... float");
        values ("float, ... float");
        values ("float, ... float");
      }
      ...
    }
    ...
    pin (pin_name2) { /* input pin */
      ...
      max_trans (template_name3) { /* output or inout pin */
        index_1 ("float, ... float");
        index_2 ("float, ... float");
        index_3 ("float, ... float");
        values ("float, ... float");
        values ("float, ... float");
      }
      ...
    }
  }
}
```
values ("float, ... float");
}
} /* pin */
} /* cell */
...}
} /* library */

**Maximum Transition Modeling Example**

**Example 4-7  Frequency-Dependent Maximum Transition Model With One-Dimensional Lookup Table**

```liberty
library(my_lib) {
    technology (cmos);
    delay_model : table_lookup;
    ...
    maxtrans_lut_template ( mt ) {
        variable_1 : frequency;
        index_1 ( "100.0000, 200.0000" );
    }
    ...
    cell ( test ) {
        .......
        pin(Z) {
            direction : output;
            function: "A";
            ...
            max_trans(mt) {
                index_1 ( "100.0000, 200.0000, 300.0000" );
                values ( "925.0000, 835.0000, 745.0000" );
            } /* end of max_trans */
            timing () {
                related_pin : "A" ;
                ...
            } /* end of arc */
        } /* end of pin Z */
        pin(A) {
            direction : input;
            max_transition : 5000.000000 ;
            capacitance : 1.0 ;
        } /* end of pin A */
    } /* end of cell */
} /* end of library */
```

**min_transition Attribute**

This attribute defines a design rule constraint for the minimum acceptable transition time of an input, output, or an inout pin.

**Example**

```liberty
pin(A) {
    direction : input;
```
max_capacitance Attribute
This attribute defines the maximum total capacitive load that an output pin can drive. This attribute can be specified only for an output or inout pin.

Example
pin(Q) {
  direction : output;
  max_capacitance : 5.0;
}

You can specify max_capacitance at three levels: at the library level, at the pin level, and on the command line.

max_cap Group
The max_cap group specifies the maximum capacitive load that an output or inout pin can drive as a function of operating frequency of the cell or both operating frequency and input transition time. Use the max_cap group instead of the max_capacitance attribute to include these effects on the maximum capacitance. When both the max_cap group and max_capacitance attribute are present, the max_cap group overrides the max_capacitance attribute.

To model the effect of operating frequency and input transition time on the maximum capacitance, define the lookup table template for the max_cap group at the library level.

To define the lookup table template, use the maxcap_lut_template group at the library level. The maxcap_lut_template group can have the variables, variable_1 and variable_2. The valid values of the variable_1 and variable_2 variables are frequency and input_transition_time, respectively.

The one-dimensional lookup table consists of the maximum capacitance values for different values of frequency. The max_cap group takes the value of the maximum capacitance from the lookup table by using the variable_1 variable.

The two-dimensional lookup table consists of the maximum capacitance values for different values of frequency and input_transition_time. The max_cap group takes the value of the maximum capacitance from the lookup table by using the variable_1 and variable_2 variables.

The following shows the maximum capacitance model syntax:
library (library_name) {
  delay_model : table_lookup;
  ...
maxcap_lut_template (template_name1) { /*1-D lookup table template*/
  variable_1 : frequency;
  index_1  ( "float, ... float" );
}
maxcap_lut_template (template_name2) { /*2-D LUT template */
  variable_1 : frequency;
  variable_2 : input_transition_time;
  index_1  ( "float, ... float" );
  index_2  ( "float, ... float" );
}
cell (cell_name) {
...
  pin (pin_name) {
    max_cap (template_name1) {
      index_1  ( "float, ... float" );
      values ( "float, ... float" );
    }
    ...
  } /* pin */
  ...
} /* cell */
/* library */

Maximum Capacitance Modeling Example

Example 4-8  Frequency-Dependent Maximum Capacitance Model With One-Dimensional Lookup Table

library(example_library) {
  technology (cmos);
  delay_model : table_lookup;
  ...
  maxcap_lut_template ( mc ) {
    variable_1 : frequency;
    index_1  ( "100.0000, 200.0000" );
  }
  ...
  cell ( test ) {
    ...
    pin(2) {
      direction : output ;
      function: "A";
      max_transition : 5000.000000 ;
      min_transition : 0.000000 ;
      min_capacitance : 0.000000 ;
      max_cap(mc) {
        index_1  ( "100.0000, 200.0000, 300.0000" );
        values ( "925.0000, 835.0000, 745.0000" );
      } /* end of max_cap */
      timing () { /* end of max_cap */
        related_pin : "A" ;


... 
} /* end of pin Z */
pin(A) {
  direction : input;
  max_transition : 5000.000000;
  capacitance : 1.0;
} /* end of pin A */
} /* end of cell */
} /* end of library */

**min_capacitance Attribute**

This attribute defines the minimum total capacitive load that an output pin can drive. The capacitance load cannot be less than the minimum capacitance value. This attribute can be specified only for an output or inout pin.

**Example**

```plaintext
pin(Q) {
  direction : output;
  min_capacitance : 1.0;
}
```

**cell_degradation Group**

Use the `cell_degradation` group to describe a cell performance degradation design rule when compiling a design. A cell degradation design rule specifies the maximum capacitive load a cell can drive without causing cell performance degradation during the fall transition.

This description is restricted to functionally related input and output pairs. You can determine the degradation value by switching some inputs while keeping other inputs constant. This causes output discharge. The degradation value for a specified input transition rate is the maximum output loading that does not cause cell degradation.

You can model cell degradation only in libraries using the CMOS nonlinear delay model. Cell degradation modeling uses the same format of templates and lookup tables used to model delay with the nonlinear delay model.

There are two ways to model cell degradation,

1. Create a one-dimensional lookup table template that is indexed by input transition time.

1. The following shows the syntax of the cell degradation template.

```plaintext
lu_table_template(template_name) {
  variable_1 : input_net_transition;
  index_1 ("float, .., float");
}
```

The valid value for `variable_1` is `input_net_transition`. 
The `index_1` values must be greater than or equal to 0.0 and follow the same rules for the lookup table template `index_1` attribute described in “Defining pin Groups” on page 4-11. The number of floating-point numbers in `index_1` determines the size of the table dimension.

This is an example of a cell degradation template.

```plaintext
table_template(deg_constraint) {
  variable_1 : input_net_transition;
  index_1 ("0.0, 1.0, 2.0");
}
```

2. Use the `cell_degradation` group and the cell degradation template to create a one-dimensional lookup table for each timing arc in the cell. You receive warning messages if you define a `cell_degradation` construct for some, but not all, timing arcs in the cell.

The following example shows the `cell_degradation` group:

```plaintext
pin(output) {
  timing() {
    cell_degradation(deg_constraint) {
      index_1 ("0.5, 1.5, 2.5");
      values ("0.0, 2.0, 4.0");
    }
  }
}
```

3. You can describe cell degradation groups only in the following types of timing groups:
   - combinational
   - three_state_enable
   - rising_edge
   - falling_edge
   - preset
   - clear

---

**Describing Clocks**

To define clocks and clocking, use these `pin` group attributes:

- `clock`
- `min_period`
• min_pulse_width_high
• min_pulse_width_low

clock Attribute
This attribute indicates whether or not an input pin is a clock pin.
A true value labels a pin as a clock pin. A false value labels a pin as not a clock pin, even though it might otherwise have such characteristics.

Example
clock : true ;

min_period Attribute
Place the min_period attribute on the clock pin of a flip-flop or a latch to specify the minimum clock period required for the input pin. The minimum period is the sum of the data arrival time and setup time. This time must be consistent with the max_transition time.

Example
min_period : 26.0;

min_pulse_width_high and min_pulse_width_low Attributes
Use these optional attributes to specify the minimum length of time a pin must remain at logic 1 (min_pulse_width_high) or logic 0 (min_pulse_width_low). These attributes can be placed on a clock input pin or an asynchronous clear/preset pin of a flip-flop or latch.

Example
The following example shows both attributes on a clock pin, indicating the minimum pulse width for a clock pin.

    pin(CLK) {
        direction : input ;
        capacitance : 1 ;
        min_pulse_width_high : 3 ;
        min_pulse_width_low : 3 ;
    }

Describing Clock Pin Functions
To define the function of a clock pin, use these pin group attributes:

• function
• three_state
• x_function
• state_function
• internal_node

function Attribute

The function attribute defines the value of an output or inout pin in terms of the cell’s input or inout pins.

Syntax

```
function : "Boolean expression" ;
```

The precedence of the operators is left to right, with inversion performed first, then XOR, then AND, then OR.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>Invert previous expression</td>
</tr>
<tr>
<td>!</td>
<td>Invert following expression</td>
</tr>
<tr>
<td>^</td>
<td>Logical XOR</td>
</tr>
<tr>
<td>*</td>
<td>Logical AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td>space</td>
<td>Logical AND</td>
</tr>
<tr>
<td>+</td>
<td>Logical OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Signal tied to logic 1</td>
</tr>
<tr>
<td>0</td>
<td>Signal tied to logic 0</td>
</tr>
</tbody>
</table>
Grouped Pins in function Statements

Grouped pins can be used as variables in a function statement; see “Defining Bused Pins” on page 4-39 and “Defining Signal Bundles” on page 4-46. In function statements that use bus or bundle names, all the variables in the statements must be either a single pin or buses or bundles of the same width.

Ranges of buses or bundles are valid if the range you define contains the same number of members as the other buses or bundles in the same expression. You can reverse the bus order by listing the member numbers in reverse (high: low) order. Two buses, bundles, or bused-pin ranges with different widths should not appear in the same function statement.

When the function attribute of a cell with group input pins is a combinational-logic function of grouped variables only, the logic function is expanded to apply to each set of output grouped pins independently. For example, if A, B, and Z are defined as buses of the same width and the function statement for output Z is

```
function : "(A & B)" ;
```

the function for Z[0] is interpreted as

```
function : "(A[0] & B[0])" ;
```

Likewise, the function for Z[1] is interpreted as

```
```

If a bus and a single pin are in the same function attribute, the single pin is distributed across all members of the bus. For example, if A and Z are buses of the same width, B is a single pin, and the function statement for the Z output is

```
function : "(A & B)" ;
```

The function for Z[0] is interpreted as

```
function : "(A[0] & B)" ;
```

Likewise, the function for Z[1] is interpreted as

```
```

three_state Attribute

Use this attribute to define a three-state output pin in a cell.

Example 4-9  Three-State Cell Description

```
library(example){
  technology (cmos) ;
  date : "May 14, 2002" ;
  revision : 2002.05;
  ...
  cell(TRI_INV2) {
```
area : 3 ;
pin(A) {
    direction : input ;
    capacitance : 2 ;
}

pin(E) {
    direction : input ;
    capacitance : 2 ;
}

pin(Z) {
    direction : output ;
    function : "A'" ;
    three_state : "E'" ;
    timing() {
        ...
    }
}

x_function Attribute

Use the x_function attribute to describe the X behavior of a pin, where X is a state other than 0, 1, or Z.

The three_state, function, and x_function attributes are defined for output and inout pins and can have shared input. You can assign three_state, function, and x_function to be the function of the same input pins. When these functions have shared input, however, the cell must be inserted manually.

Also, when the values of more than one function equal 1, the three functions are evaluated in this order:

1. x_function
2. three_state
3. function

Example

pin (y) {
    direction: output;
    function : !(ap * an) ;
    x_function : !(ap * an) ;
    three_state : ap * !an ;
}
**state_function Attribute**

Use this attribute to define output logic. Ports in the state_function Boolean expression must be either input, three-state inout, or ports with an internal_node attribute. If the output logic is a function of only the inputs (IN), the output is purely combinational (for example, feed-through output). A port in the state_function expression refers only to the non-three-state functional behavior of that port. An inout port in the state_function expression is treated only as an input port.

**Example**

```
state_function : Q*X;
```

**internal_node Attribute**

Use this attribute to resolve node names to real port names. The internal_node attribute describes the sequential behavior of an output pin. It provides the relationship between the statetable group and a pin of a cell. Each output with the internal_node attribute might also have the optional input_map attribute.

**Example**

```
internal_node : "Q";
```

---

**Defining Bused Pins**

To define bused pins, use these groups:

- **type group**
- **bus group**

You can use a defined bus or bus member in Boolean expressions in the function attribute. An output pin does not need to be defined in a cell before it is referenced.

---

**type Group**

If your library contains bused pins, you must define type groups and define the structural constraints of each bus type in the library.

The type group is defined at the library group level, as follows:

```
library (lib_name) {
    type ( name ) {
        ... type description ...
    }
}
```
name

Identifies the bus type.

A `type` group can one of the following:

- `base_type`
  Only the array base type is supported.

- `data_type`
  Only the bit data type is supported.

- `bit_width`
  An integer that designates the number of bus members. The default is 1.

- `bit_from`
  An integer indicating the member number assigned to the most significant bit (MSB) of successive array members. The default is 0.

- `bit_to`
  An integer indicating the member number assigned to the least significant bit (LSB) of successive array members. The default is 0.

- `doownto`
  A value of true indicates that member number assignment is from high to low instead of low to high. The default is false (low to high).

**Example 4-10  type Group Statement**

```plaintext
type ( BUS4 ) {
  base_type : array ;
  data_type : bit ;
  bit_width : 4 ;
  bit_from : 0 ;
  bit_to : 3 ;
  doownto :false ;
}
```

It is not necessary to use all the `type` group attributes. For example, both the `type` group statements in Example 4-11 are valid descriptions of BUS4 of Example 4-10.

**Example 4-11  Alternative type Group Statements**

```plaintext
type ( BUS4 ) {
  base_type : array ;
  data_type : bit ;
  bit_width : 4 ;
  bit_from : 0 ;
  ```
After you define a type group, you can use the type group in a bus group to describe bused pins.

### bus Group

A bus group describes the characteristics of a bus. You define it in a cell group, as shown here:

```plaintext
library (lib_name) {
    cell (cell_name) {
        area : float ;
        bus (name) {
            ... bus description ...
        }
    }
}
```

A bus group contains the following elements:

- bus_type attribute
- pin groups

In a bus group, use the number of bus members (pins) defined by the bit_width attribute in the applicable type group. You must declare the bus_type attribute first in the bus group.

### bus_type Attribute

The bus_type attribute specifies the bus type. It is a required element of all bus groups. Always declare the bus_type as the first attribute in a bus group.

**Syntax**

```plaintext
bus_type : name ;
```
Pin Attributes and Groups

Pin attributes in a bus or bundle group specify default attribute values for all pins in that bus or bundle. Pin attributes can also appear in pin groups inside the bus or bundle group to define attribute values for specific bus or bundle pins or groups of pins. Values used in pin groups override the default attribute values defined for the bus or bundle.

All pin attributes are valid inside bus and pin groups. See “General pin Group Attributes” on page 4-12 for a description of pin attributes. The direction attribute value of all bus members must be the same.

Use the full name of a pin for the names of pins in a pin group contained in a bus group.

The following example shows a bus group that defines bus A, with defaults for direction and capacitance assigned:

```plaintext
bus (A) {
    bus_type : bus1 ;
    direction : input ;
    capacitance : 3 ;
    ...
}
```

The following example illustrates a pin group that defines a new capacitance attribute value for pin 0 in bus A:

```plaintext
pin (A[0]) {
    capacitance : 4 ;
}
```

You can also define pin groups for a range of bus members. A range of bus members is defined by a beginning value and an ending value, separated by a colon. No spaces can appear between the colon and the member numbers.

The following example illustrates a pin group that defines a new capacitance attribute value for bus members 0, 1, 2, and 3 in bus A:

```plaintext
pin (A[0:3]) {
    capacitance : 4 ;
}
```
For nonbused pins, you can identify member numbers as single numbers or as a range of numbers separated by a colon. Do not define member numbers in a list.

Table 4-7  Comparison of Bused and Single-Pin Formats

<table>
<thead>
<tr>
<th>Pin type</th>
<th>Technology library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bused Pin</td>
<td>pin x[3:0]</td>
</tr>
<tr>
<td>Single Pin</td>
<td>pin x</td>
</tr>
</tbody>
</table>

Example Bus Description

Example 4-12 includes type and bus groups. It also shows the use of bus variables in function, related_pin, pin_opposite, and pin_equal attributes.

Example 4-12  A Complete Bus Description

```liberty
library (ExamBus) {
    date : "May 14, 2002";
    revision : 2002.05;
    bus_naming_style : "%s[%d]"; /* Optional; this is the default */
    type (bus4) {
        base_type : array; /* Required */
        data_type : bit; /* Required if base_type is array */
        bit_width : 4; /* Optional; default is 1 */
        bit_from : 0; /* Optional MSB; defaults to 0 */
        bit_to : 3; /* Optional LSB; defaults to 0 */
        downto : false; /* Optional; defaults to false */
    }
    cell (bused_cell) {
        area : 10;
        bus (A) {
            bus_type : bus4;
            direction : input;
            capacitance : 3;
            pin (A[0:2]) {
                capacitance : 2;
            }
        }
        pin (A[3]) {
            capacitance : 2.5;
        }
    }
    bus (B) {
        bus_type : bus4;
        direction : input;
        capacitance : 2;
    }
    pin (E) {
```
direction : input;
capacitance 2;
}

bus (X) {
  bus_type : bus4;
direction : output;
capacitance : 1;
  pin (X[0:3]) {
    function : "A & B’";
timing() {
      related_pin : "A B’";
      /* A[0] and B[0] are related to X[0],
      A[1] and B[1] are related to X[1], etc. */
    }
  }
}

bus (Y) {
  bus_type : bus4;
direction : output;
capacitance : 1;
  pin (Y[0:3]) {
    function : "B’";
    three_state : "!E’";
timing () {
      related_pin : "A[0:3] B E’";
    }
    internal_power() {
      when: "E’";
      related_pin : B ;
power() {
      ...
    }
  }
}

bus (Z) {
  bus_type : bus4;
direction : output;
  pin (Z[0:1]) {
    function : "!A[0:1]";
timing () {
      related_pin : "A[0:1]";
    }
    internal_power() {
      related_pin : "A[0:1]";
power() {
      ...
    }
}
pin (Z[2]) {
    function "A[2]";
    timing () {
        related_pin : "A[2]";
    }
    internal_power() {
        related_pin : "A[0:1]";
        power() {
            ...
        }
    }
}

pin (Z[3]) {
    function : "!A[3]";
    timing () {
        related_pin : "A[3]";
    }
    internal_power() {
        related_pin : "A[0:1]";
        power() {
            ...
        }
    }
}

pin_opposite("Y[0:1]","Z[0:1]");
/* Y[0] is opposite to Z[0], etc. */

pin_equal("Y[2:3] Z[2:3]");
/* Y[2], Y[3], Z[2], and Z[3] are equal */
cell (bused_cell2) {
    area : 20;
    bus (A) {
        bus_type : bus41;
        direction : input;
        capacitance : 1;
        pin (A[0:3]) {
            capacitance : 2;
        }
        pin (A[3]) {
            capacitance : 2.5;
        }
    }
    bus (B) {
        bus_type : bus4;
        direction : input;
        capacitance : 2;
    }
    pin (E) {
        direction : input;
        capacitance 2;
    }
}
Chapter 4: Defining Core Cells
Defining Signal Bundles

You need certain attributes to define a bundle. A bundle groups several pins that have similar timing or functionality. Bundles are used for multibit cells such as multibit latch, multibit flip-flop, and multibit AND gate.

bundle Group

Define a bundle group in a cell group, as shown:

```plaintext
class library (lib_name) { 
class cell (cell_name) { 
    area : float ; 
    class bundle (name) { 
        ... bundle description ... 
    } 
} 
} 
```

A bundle group contains the following elements:

members attribute

The members attribute must be declared first in a bundle group.

pin attributes

These include direction, function, and three-state.
members Attribute

The members attribute is used in a bundle group to list the pin names of the signals in a bundle. The members attribute must be included as the first attribute in the bundle group. It provides the bundle element names and groups a set of pins that have similar properties. The number of members defines the width of the bundle.

If a bundle has a function attribute defined for it, that function is copied to all bundle members. For example,

```liberty
pin (C) {
  direction : input ;
  ...
}
bundle(A) {
  members(A0, A1, A2, A3);
  direction : output ;
  function : "B' + C";
  ...
}
bundle(B) {
  members(B0, B1, B2, B3);
  direction : input;
  ...
}
```

means that the members of the A bundle have these values:

- A0 = B0’ + C;
- A1 = B1’ + C;
- A2 = B2’ + C;
- A3 = B3’ + C;

Each bundle operand (B) must have the same width as the function parent bundle (A).

pin Attributes

For information about pin attributes, see “General pin Group Attributes” on page 4-12.

Example 4-13 Multibit Latch With Signal Bundles (bundle Group Syntax)

```liberty
cell (latch4) {
  area: 16;
  pin (G) { /* active-high gate enable signal */
    direction : input;
    ...
  }
  bundle (D) { /* data input with four member pins */
    members(D1, D2, D3, D4); /*must be first attribute */
    direction : input;
  }
```

Defining Layout-Related Multibit Attributes

The single_bit_degenerate attribute is a layout-related attribute for multibit cells. The attribute also applies to sequential and combinational cells.

The single_bit_degenerate attribute is for use on multibit bundle or bus cells that are black boxes. The value of this attribute is the name of a single-bit library cell.

Example 4-14  Multibit Cells With single_bit_degenerate Attribute

cell (FDX2) {
  area : 18 ;
  single_bit_degenerate : FDB ;
}
The library description does not include information such as cell height; this must be provided by the library developer.

---

### Defining Multiplexers

A one-hot MUX is a library cell that behaves functionally as a regular MUX logic gate. However, in the case of a one-hot MUX, some inputs are considered dedicated control inputs and others are considered dedicated data inputs. There are as many control inputs as data inputs, and the function of the cell is the logic AND of the $i$th control input with the $i$th data input. For example, a 4-to-1 one-hot MUX has the following function:

$$Z = (D_{0} \& C_{0}) \mid (D_{1} \& C_{1}) \mid (D_{2} \& C_{2}) \mid (D_{3} \& C_{3})$$

One-hot MUXs are generally implemented using pass gates, which makes them very fast and allows their speed to be largely independent of the number of data bits being multiplexed. However, this implementation requires that exactly one control input be active at a time. If no control inputs are active, the output is left floating. If more than one control input is active, there could be an internal drive fight.
Library Requirements

One-hot MUX library cells must meet the following requirements:

- A one-hot MUX cell in the target library should be a single-output cell.
- Its inputs can be divided into two disjoint sets of the same size as follows:
  \[ C = \{C_1, C_2, \ldots, C_n\} \quad \text{and} \quad D = \{D_1, D_2, \ldots, D_n\} \]
  where \( n > 1 \) and is the size of the set. Actual names of the inputs can vary.
- The `contention_condition` attribute must be set on the cell. The value of the attribute is a combinational function, \( f(C) \), of inputs in set \( C \) that defines prohibited combinations of inputs as shown in the following examples (where size \( n \) of the set is 3):
  \[ FC = C_0' \land C_1' \land C_2' \lor C_0 \land C_1 \lor C_0 \land C_2 \lor C_1 \land C_2 \]
  or
  \[ FC = (C_0 \land C_1' \land C_2' \lor C_0' \land C_1 \land C_2' \lor C_0' \land C_1' \land C_2)' \]
- The cell must have a combinational function \( FO \) defined on the output with respect to all its inputs. This function \( FO \) must logically define, together with the contention condition, a base function \( F^* \) that is a sum of \( n \) product terms, where the \( i \)th term contains all the inputs in \( C \), with \( C_i \) high and all others low and exclusively one input in \( D \).

Examples of the defined function are as follows (for \( n = 3 \)):

\[ F^* = C_0 \land C_1' \land C_2' \land D_0 \lor C_0' \land C_1 \land C_2' \land D_1 \lor C_0' \land C_1' \land C_2 \land D_2' \]

or

\[ F^* = C_0 \land C_1' \land C_2' \land D_0' + C_0' \land C_1 \land C_2' \land D_1' + C_0' \land C_1' \land C_2 \land D_2' \]

The function \( FO \) can take many forms, if it satisfies the following condition:

\[ FO \land FC' == F^* \]

when \( FO \) is restricted by \( FC' \), it should be equivalent to \( F^* \). The term \( FO = F^* \) is acceptable; other examples are as follows (for \( n = 3 \)):

\[ FO = (D_0 \land C_0) \lor (D_1 \land C_1) \lor (D_2 \land C_2) \]

or

\[ FO = (D_0' \land C_0) \lor (D_1' \land C_1) \lor (D_2' \land C_2) \]
Note:
When FO is restricted by FC, inverting all inputs in D is equivalent to inverting the output. Inverting only a subset of D yields an incompatible function. It is recommended that you use the simple form described earlier, or $F^\ast$.

The following example shows a cell that is properly specified.

Example
cell(one_hot_mux_example) {
  ... ... 
  contention_condition : "{C0 C1 + C0' C1'}";
  ... ...
  pin(D0) {
    direction : input;
    ... ... 
  } 
  pin(D1) {
    direction : input;
    ... ... 
  } 
  pin(C0) {
    direction : input;
    ... ... 
  } 
  pin(C1) {
    direction : input;
    ... ... 
  } 
  pin(Z) {
    direction : output;
    function : "{C0 D0 + C1 D1}";
    ... ...
  } 
} 

---

Defining Decoupling Capacitor Cells, Filler Cells, and Tap Cells

Decoupling capacitor cells, or decap cells, are cells that have a capacitor placed between the power rail and the ground rail to overcome dynamic voltage drop; filler cells are used to connect the gaps between the cells after placement; and tap cells are physical-only cells that have power and ground pins and do not have signal pins. Cell-level attributes that identify decoupling capacitor cells, filler cells, and tap cells in libraries are supported.
Syntax

The following syntax shows a cell with the is_decap_cell, is_filler_cell and is_tap_cell attributes, which identify decoupling capacitor cells, filler cells, and tap cells, respectively. However, only one attribute can be set to true in a given cell.

```plaintext
cell (cell_name) {
  ...
  is_decap_cell : true | false;
  is_filler_cell : true | false;
  is_tap_cell : true | false;
  ...
} /* End cell group */
```

Cell-Level Attributes

The following attributes can be set at the cell level to identify decoupling capacitor cells, filler cells, and tap cells.

**is_decap_cell**

The is_decap_cell attribute identifies a cell as a decoupling cell. Valid values are true and false.

**is_filler_cell**

The is_filler_cell attribute identifies a cell as a filler cell. Valid values are true and false.

**is_tap_cell**

The is_tap_cell attribute identifies a cell as a tap cell. Tap cells are physical-only cells, which means they have power and ground pins only and not signal pins. Tap cells are well-tied cells that bias the silicon infrastructure of n-wells or p-wells. Valid values for the is_tap_cell attribute are true and false.
Defining Sequential Cells

This chapter describes the peculiarities of defining flip-flops and latches, building upon the cell description syntax given in Chapter 4, “Defining Core Cells.” It describes group statements that apply only to sequential cells and also describes a variation of the function attribute that makes use of state variables.

To design flip-flops and latches, you must understand the following concepts and tasks:

• Using Sequential Cell Syntax
• Using the function Attribute
• Describing a Multibit Flip-Flop
• Describing a Latch
• Describing a Multibit Latch
• Describing Sequential Cells With the Statetable Format
• Flip-Flop and Latch Examples
• Cell Description Examples
Using Sequential Cell Syntax

You can describe sequential cells with the following cell definition formats:

• ff or latch format

  Cells using the ff or latch format are identified by the ff group and latch group.

• statetable format

  Cells using the statetable format are identified by the statetable group. The statetable format supports all the sequential cells supported by the ff or latch format. In addition, the statetable format supports complex sequential cells, such as the following:

  ❍ Sequential cells with multiple clock ports, such as a cell with a system clock and a test scan clock
  ❍ Internal state sequential cells, such as master-slave cells
  ❍ Multistate sequential cells, such as counters and shift registers
  ❍ Sequential cells with combinational outputs
  ❍ Sequential cells with complex clocking and complex asynchronous behavior
  ❍ Sequential cells with multiple simultaneous input transitions
  ❍ Sequential cells with illegal input conditions

  The statetable format contains a complete, expanded set of table rules for which all L and H permutations of table input are explicitly specified.

Some cells cannot be modeled with the statetable format. For example, you cannot use the statetable format to model a cell whose function depends on differential clocks when the inputs change.

Describing a Flip-Flop

To describe an edge-triggered storage device, include a ff group or a statetable group in a cell definition. This section describes how to define a flip-flop by using the ff or latch format. See “Describing Sequential Cells With the Statetable Format” on page 5-23 for the way to define cells using the statetable group.
Using the ff Group

A **ff** group describes either a single-stage or a master-slave flip-flop. The **ff_bank** group represents multibit registers, such as a bank of flip-flops. See “Describing a Multibit Flip-Flop” on page 5-11 for more information about the **ff_bank** group.

**Syntax**

```
library (lib_name) {
  cell (cell_name) {
    ...
    ff (variable1, variable2) {

      clocked_on : "Boolean_expression" ;
      next_state : "Boolean_expression" ;

      clear : "Boolean_expression" ;
      preset : "Boolean_expression" ;

      clear_preset_var1 : value ;
      clear_preset_var2 : value ;
      clocked_on_also : "Boolean_expression" ;
      power_down_function : "Boolean_expression" ;
    }
  }
}
```

*variable1*

The state of the noninverting output of the flip-flop. It is considered the 1-bit storage of the flip-flop.

*variable2*

The state of the inverting output.

You can name *variable1* and *variable2* anything except the name of a pin in the cell being described. Both variables are required, even if one of them is not connected to a primary output pin.

The **clocked_on** and **next_state** attributes are required in the **ff** group; all other attributes are optional.
clocked_on and clocked_on_also Attributes

The clocked_on and clocked_on_also attributes identify the active edge of the clock signals.

Single-state flip-flops use only the clocked_on attribute. When you describe flip-flops that require both a master and a slave clock, use the clocked_on attribute for the master clock and the clocked_on_also attribute for the slave clock.

Examples

A rising-edge-triggered device is:
clocked_on : "CP";

A falling-edge-triggered device is:
clocked_on : "CP’";

next_state Attribute

The next_state attribute is a logic equation written in terms of the cell’s input pins or the first state variable, variable1. For single-stage storage elements, the next_state attribute equation determines the value of variable1 at the next active transition of the clocked_on attribute.

For devices such as a master-slave flip-flop, the next_state attribute equation determines the value of the master stage’s output signals at the next active transition of the clocked_on attribute.

Example

next_state : "D";

nextstate_type Attribute

The nextstate_type attribute is a pin group attribute that defines the type of next_state attribute used in the ff or ff_bank group.

Any pin with the nextstate_type attribute must be included in the value of the next_state attribute.

Note:

Specify a nextstate_type attribute to ensure that the sync set (or sync reset) pin and the D pin of sequential cells are not swapped when instantiated.

Example

nextstate_type : data;
Example 5-5 on page 5-11 and Example 5-6 on page 5-13 show the use of the `nextstate_type` attribute.

**clear Attribute**
The `clear` attribute gives the active value for the clear input.
The example defines an active-low clear signal.

**Example**
clear : "CD’" ;

For more information about the `clear` attribute, see “Describing a Single-Stage Flip-Flop” on page 5-8.

**preset Attribute**
The `preset` attribute gives the active value for the preset input.
The example defines an active-high preset signal.

**Example**
preset : "PD’" ;

For more information about the `preset` attribute, see “Describing a Single-Stage Flip-Flop” on page 5-8.

**clear_preset_var1 Attribute**
The `clear_preset_var1` attribute gives the value that `variable1` has when `clear` and `preset` are both active at the same time.

**Example**
clear_preset_var1 : L;

For more information about the `clear_preset_var1` attribute, including its function and values, see “Describing a Single-Stage Flip-Flop” on page 5-8.

**clear_preset_var2 Attribute**
The `clear_preset_var2` attribute gives the value that `variable2` has when `clear` and `preset` are both active at the same time.

**Example**
clear_preset_var2 : L ;
For more information about the `clear_preset_var2` attribute, including its function and values, see “Describing a Single-Stage Flip-Flop” on page 5-8.

**power_down_function Attribute**

The `power_down_function` attribute specifies the Boolean condition when the cell’s output pin is switched off by the power and ground pins (when the cell is in off mode due to the external power pin states).

You specify the `power_down_function` attribute for combinational and sequential cells. For simple or complex sequential cells, the `power_down_function` attribute also determines the condition of the cell’s internal state.

**Syntax**

```plaintext
library (name) {
  cell (name) {
    ff (variable1,variable2) {
      //...flip-flop description...
      clear : "Boolean expression" ;
      clear_preset_var1 : L | H | N | T | X ;
      clear_preset_var2 : L | H | N | T | X ;
      clocked_on : "Boolean expression" ;
      clocked_on_also : "Boolean expression" ;
      next_state : "Boolean expression" ;
      preset : "Boolean expression" ;
      power_down_function : "Boolean expression" ;
    }
  }
...
}
```

**Example**

```plaintext
library ("low_power_cells") {
  cell ("retention_dff") {
    pg_pin(VDD) {
      voltage_name : VDD;
      pg_type : primary_power;
    }
    pg_pin(VSS) {
      voltage_name : VSS;
      pg_type : primary_ground;
    }
    pin ("D") {
      direction : "input";
    }
    pin ("CP") {
      direction : "input";
    }
    ff(IQ,IQN) {
```

```
next_state : "D" ;
clocked_on : "CP" ;
power_down_function : "!VDD + VSS" ;
}

pin ("Q") {
    function : " IQ ";
    direction : "output";
    power_down_function : "!VDD + VSS";
}

ff Group Examples

The following is an example of the ff group for a single-stage D flip-flop.

ff(IQ, IQN) {
    next_state : "D" ;
clocked_on : "CP" ;
}

The example defines two variables, IQ and IQN. The next_state attribute determines the value of IQ after the next active transition of the clocked_on attribute. In the example, IQ is assigned the value of the D input.

For some flip-flops, the next state depends on the current state. In this case, the first state variable, variable1 (IQ in the example), is used in the next_state statement; and the second state variable, IQN, is not used.

For the example, the ff attribute group for a JK flip-flop is,

ff(IQ, IQN) {
    next_state : "(J K IQ') + (J K') + (J' K' IQ)";
clocked_on : "CP";
}

The next_state and clocked_on attributes define the synchronous behavior of the flip-flop.
Describing a Single-Stage Flip-Flop

A single-stage flip-flop does not use the optional `clocked_on_also` attribute.

**Table 5-1  Functions of the Attributes in the ff Group for a Single-Stage Flip-Flop**

<table>
<thead>
<tr>
<th>active_edge</th>
<th>clear</th>
<th>preset</th>
<th>variable1</th>
<th>variable2</th>
</tr>
</thead>
<tbody>
<tr>
<td>clocked_on</td>
<td>inactive</td>
<td>inactive</td>
<td>next_state</td>
<td>!next_state</td>
</tr>
<tr>
<td>--</td>
<td>active</td>
<td>inactive</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>--</td>
<td>inactive</td>
<td>active</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>--</td>
<td>active</td>
<td>active</td>
<td>clear_preset_var1</td>
<td>clear_preset_var2</td>
</tr>
</tbody>
</table>

The `clear` attribute gives the active value for the clear input. The `preset` attribute gives the active value for the preset input. For example, the following statement defines an active-low clear signal.

`clear : "CD'" ;`

The `clear_preset_var1` and `clear_preset_var2` attributes specify the value for `variable1` and `variable2` when `clear` and `preset` are both active at the same time.

**Table 5-2  Valid Values for the clear_preset_var1 and clear_preset_var2 Attributes**

<table>
<thead>
<tr>
<th>Variable values</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>No change</td>
</tr>
<tr>
<td>T</td>
<td>Toggle the current value from 1 to 0, 0 to 1, or X to X</td>
</tr>
<tr>
<td>X</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

If you use both `clear` and `preset`, you must use either `clear_preset_var1`, `clear_preset_var2`, or both. Conversely, if you include `clear_preset_var1`, `clear_preset_var2`, or both, you must use both `clear` and `preset`.

The flip-flop cell is activated whenever `clear`, `preset`, `clocked_on`, or `clocked_on_also` change.
Example 5-1 is a \texttt{ff} group for a single-stage D flip-flop with a rising edge, negative clear and preset, and the output pins set to 0 when both \texttt{clear} and \texttt{preset} are active (low).

**Example 5-1 Single-Stage D Flip-Flop**

```plaintext
ff(IQ, IQN) {
    next_state : "D" ;
    clocked_on : "CP" ;
    clear : "CD'" ;
    preset : "PD'" ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}
```

Example 5-2 is a \texttt{ff} group for a single-stage, rising edge-triggered JK flip-flop with scan input, negative clear and preset, and the output pins set to 0 when \texttt{clear} and \texttt{preset} are both active.

**Example 5-2 Single-Stage JK Flip-Flop**

```plaintext
ff(IQ, IQN) {
    next_state : "(TE*TI)+(TE'*J*K')+(TE'*J'*K'*IQ)+(TE'*J*K*IQ')" ;
    clocked_on : "CP" ;
    clear : "CD'" ;
    preset : "PD'" ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}
```

Example 5-3 is a \texttt{ff} group for a D flip-flop with synchronous negative clear.

**Example 5-3 D Flip-Flop With Synchronous Negative Clear**

```plaintext
ff(IQ, IQN) {
    next_state : "D * CLR" ;
    clocked_on : "CP" ;
}
```

---

Describing a Master-Slave Flip-Flop

The specification for a master-slave flip-flop is the same as for a single-stage device, except that it includes the \texttt{clocked_on_also} attribute.

**Table 5-3 Functions of Attributes of \texttt{ff} Group for a Master-Slave Flip-Flop**

<table>
<thead>
<tr>
<th>active_edge</th>
<th>clear</th>
<th>preset</th>
<th>internal1</th>
<th>internal2</th>
<th>variable1</th>
<th>variable2</th>
</tr>
</thead>
<tbody>
<tr>
<td>clocked_on</td>
<td>inactive</td>
<td>inactive</td>
<td>next_state</td>
<td>!next_state</td>
<td>internal1</td>
<td>internal2</td>
</tr>
<tr>
<td>clocked_on_also</td>
<td>inactive</td>
<td>inactive</td>
<td></td>
<td></td>
<td>internal1</td>
<td>internal2</td>
</tr>
</tbody>
</table>
Table 5-3 Functions of Attributes of ff Group for a Master-Slave Flip-Flop (continued)

<table>
<thead>
<tr>
<th>active_edge</th>
<th>clear</th>
<th>preset</th>
<th>internal1</th>
<th>internal2</th>
<th>variable1</th>
<th>variable2</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>active</td>
<td></td>
<td>clear_preset_var1</td>
<td>clear_preset_var2</td>
<td>clear_preset_var1</td>
<td>clear_preset_var2</td>
</tr>
<tr>
<td>active</td>
<td>inactive</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>inactive</td>
<td>active</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The `internal1` and `internal2` variables represent the output values of the master stage, and `variable1` and `variable2` represent the output values of the slave stage.

The `internal1` and `internal2` variables have the same value as `variable1` and `variable2`, respectively, when the `clear` and `preset` attributes are both active at the same time.

Note:
You do not need to specify the `internal1` and `internal2` variables, which represent internal stages in the flip-flop.

Example 5-4 shows the `ff` group for a master-slave D flip-flop with a rising-edge sampling, falling-edge data transfer, negative clear and preset, and output values set to high when the `clear` and `preset` attributes are both active.

Example 5-4 Master-Slave D Flip-Flop

```plaintext
ff(IQ, IQN) {
    next_state : "D" ;
    clocked_on : "CLK" ;
    clocked_on_also : "CLKN'" ;
    clear : "CDN'" ;
    preset : "PDN'" ;
    clear_preset_var1 : H ;
    clear_preset_var2 : H ;
}
```

Using the function Attribute

Each storage device output pin needs a `function` attribute statement. Only the two state variables, `variable1` and `variable2`, can be used in the `function` attribute statement for sequentially modeled elements.
Example 5-5  Complete Functional Description of a Rising-Edge-Triggered D Flip-Flop With Active-Low Clear and Preset

cell (dff) {
    area : 1 ;
    pin (CLK) {
        direction : input ;
        capacitance : 0 ;
    }
    pin (D) {
        nextstate_type : data;
        direction : input ;
        capacitance : 0 ;
    }
    pin (CLR) {
        direction : input ;
        capacitance : 0 ;
    }
    pin (PRE) {
        direction : input ;
        capacitance : 0 ;
    }
    ff (IQ, IQN) {
        next_state : "D" ;
        clocked_on : "CLK" ;
        clear : "CLR'" ;
        preset : "PRE'" ;
        clear_preset_var1 : L ;
        clear_preset_var2 : L ;
    }
    pin (Q) {
        direction : output ;
        function : "IQ" ;
    }
    pin (QN) {
        direction : output ;
        function : "IQN" ;
    }
} /* end of cell dff */

Describing a Multibit Flip-Flop

The ff_bank group describes a cell that is a collection of parallel, single-bit sequential parts. Each part shares control signals with the other parts and performs an identical function. The ff_bank group is typically used to represent multibit registers. It can be used in cell and test_cell groups.

The syntax is similar to that of the ff group; see “Describing a Flip-Flop” on page 5-2.
Syntax
library (lib_name)
{
  cell (cell_name) {
    ...
    pin (pin_name) {
      ...
    }
    bundle (bundle_name) {
      ...
    }
  }
}

An input that is described in a pin group, such as the clk input, is fanned out to each flip-flop in the bank. Each primary output must be described in a bus or bundle group, and its function statement must include either variable1 or variable2.

Three-state output pins are allowed; you can add a three_state attribute to an output bus or bundle to specify this function.

The bits value in the ff_bank definition is the number of bits in this multibit cell.
Example 5-6 is the description of the multibit register shown in Figure 5-1.

Example 5-6 4-Bit Register Modeled Using ff_bank Group for Rising-Edge-Triggered D Flip-Flops

cell (dff4) {
    area : 1;
    pin (CLK) {
        direction : input;
        capacitance : 0;
        min_pulse_width_low : 3;
        min_pulse_width_high : 3;
    }
    bundle (D) {
        members(D1, D2, D3, D4);
    }
}
nextstate_type : data;
direction : input;
capacitance : 0;
timing() {
    related_pin   : "CLK";
timing_type   : setup_rising;
}
timing() {
    related_pin   : "CLK";
timing_type   : hold_rising;
}

pin (CLR) {
direction : input;
capacitance : 0;
timing() {
    related_pin   : "CLK";
timing_type   : recovery_rising;
}
}

pin (PRE) {
direction : input;
capacitance : 0;
timing() {
    related_pin   : "CLK";
timing_type   : recovery_rising;
}
}

ff_bank (IQ, IQN, 4) {
    next_state : "D";
clocked_on : "CLK";
clear : "CLR'";
preset : "PRE'";
clear_preset_var1 : L;
clear_preset_var2 : L;
}

bundle (Q) {
    members(Q1, Q2, Q3, Q4);
direction : output;
function : "(IQ)"

timing() {
    related_pin   : "CLK";
timing_type   : rising_edge;
}

timing() {
    related_pin   : "PRE";
timing_type   : preset;
timing_sense  : negative_unate;
}

timing() {
    related_pin   : "CLR";
timing_type   : clear;
timing_sense  : positive_unate;
}
Describing a Latch

To describe a level-sensitive storage device, you include a latch group or statetable group in the cell definition. This section describes how to define a latch by using the ff or latch format. See "Describing Sequential Cells With the Statetable Format" on page 5-23 for information about defining cells using the statetable group.

latch Group

This section describes a level-sensitive storage device found within a cell group.

Syntax

```plaintext
library (lib_name) {
  cell (cell1_name) {
    ...
    latch (variable1, variable2) {
      enable : "Boolean_expression" ;
      data_in : "Boolean_expression" ;
      clear : "Boolean_expression" ;
      preset : "Boolean_expression" ;
      clear_preset_var1 : value ;
      clear_preset_var2 : value ;
    }
  }
}
```

Chapter 5: Defining Sequential Cells
Describing a Latch
pin (pin_name) {
    ...
    data_in_type : data | preset | clear | load ;
}

variable1
    The state of the noninverting output of the latch. It is considered the 1-bit storage of the latch.

variable2
    The state of the inverting output.

You can name variable1 and variable2 anything except the name of a pin in the cell being described. Both variables are required, even if one of them is not connected to a primary output pin.

If you include both clear and preset, you must use either clear_preset_var1, clear_preset_var2, or both. Conversely, if you include clear_preset_var1, clear_preset_var2, or both, you must use both clear and preset.

enable and data_in Attributes
    The enable and data_in attributes are optional, but if you use one of them, you must include the other. The enable attribute gives the state of the enable input, and the data_in attribute gives the state of the data input.

Example
    enable : "G" ;
    data_in : "D" ;

data_in_type Attribute
    In a pin group, the data_in_type attribute specifies the type of input data defined by the data_in attribute. The valid values are data, preset, clear, or load. The default is data.

Note:
    The Boolean expression of the data_in attribute must include the pin with the data_in_type attribute.

Example
    data_in_type : data ;
clear Attribute
The clear attribute gives the active value for the clear input.

Example
This example defines an active-low clear signal.

```plaintext
clear : "CD'" ;
```

preset Attribute
The preset attribute gives the active value for the preset input.

Example
This example defines an active-low preset signal.

```plaintext
preset : "R'" ;
```

clear_preset_var1 Attribute
The clear_preset_var1 attribute gives the value that variable1 has when clear and preset are both active at the same time. Valid values are shown in Table 5-4.

Example
```
clear_preset_var1 : L;
```

Table 5-4  Valid Values for the clear_preset_var1 and clear_preset_var2 Attributes

<table>
<thead>
<tr>
<th>Variable values</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>No change</td>
</tr>
<tr>
<td>T</td>
<td>Toggle the current value from 1 to 0, 0 to 1, or X to X</td>
</tr>
<tr>
<td>X</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

clear_preset_var2 Attribute
The clear_preset_var2 attribute gives the value that variable2 has when clear and preset are both active at the same time. Valid values are shown in Table 5-4.
Example

\[
\text{clear\_preset\_var2} : L ;
\]

Table 5-5 Functions of the Attributes in the latch Group

<table>
<thead>
<tr>
<th>enable</th>
<th>clear</th>
<th>preset</th>
<th>variable1</th>
<th>variable2</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>inactive</td>
<td>inactive</td>
<td>data_in</td>
<td>!data_in</td>
</tr>
<tr>
<td>--</td>
<td>active</td>
<td>inactive</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>--</td>
<td>inactive</td>
<td>active</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>--</td>
<td>active</td>
<td>active</td>
<td>clear_preset_var1</td>
<td>clear_preset_var2</td>
</tr>
</tbody>
</table>

The latch cell is activated whenever \text{clear}, \text{preset}, \text{enable}, or \text{data\_in} changes.

Example 5-7 D Latch With Active-High enable and Negative clear

\[
latch(IQ, IQN) \{ \\
  \text{enable} : "G" ; \\
  \text{data\_in} : "D" ; \\
  \text{clear} : "CD'" ; \\
}\]

The \text{enable} and \text{data\_in} attributes are not required to model an SR latch, as shown in Example 5-8.

Example 5-8 SR Latch

\[
latch(IQ, IQN) \{ \\
  \text{clear} : "S'" ; \\
  \text{preset} : "R'" ; \\
  \text{clear\_preset\_var1} : L ; \\
  \text{clear\_preset\_var2} : L ; \\
}\]

Determining a Latch Cell’s Internal State

You can use the \text{power\_down\_function} attribute to specify the Boolean condition under which the cell’s output pin is switched off by the state of the power and ground pins (when the cell is in off mode due to the external power pin states). The attribute should be set under the \text{latch} group. The \text{power\_down\_function} attribute also determines the condition of the cell’s internal state.

For more information about \text{power\_down\_function}, see “\text{power\_down\_function Attribute}” on page 5-6.
power_down_function Syntax For Latch Cells

```plaintext
library (name) {
    cell (name) {
        latch (variable1, variable2) {
            //... latch description...  
            clear : "Boolean expression" ;
            clear_preset_var1 : L | H | N | T | X ;
            clear_preset_var2 : L | H | N | T | X ;
            data_in : "Boolean expression" ;
            enable : "Boolean expression" ;
            preset : "Boolean expression" ;
            power_down_function : "Boolean expression" ;
        }
    }
    ...
}
```

Describing a Multibit Latch

The `latch_bank` group describes a cell that is a collection of parallel, single-bit sequential parts. Each part shares control signals with the other parts and performs an identical function. The `latch_bank` group is typically used in `cell` and `test_cell` groups to represent multibit registers.

**latch_bank Group**

The syntax is similar to that of the `latch` group. See “Describing a Latch” on page 5-15.

**Syntax**

```plaintext
library (lib_name) {
    cell (cell_name) {
        ...
        pin (pin_name) {
            ...
        }
        bundle (bus_name) {
            ...
        }
        latch_bank (variable1, variable2, bits) {
            enable : "Boolean_expression" ;
            data_in : "Boolean_expression" ;
            clear : "Boolean_expression" ;
            preset : "Boolean_expression" ;
            clear_preset_var1 : value ;
            clear_preset_var2 : value ;
        }
    }
}
An input that is described in a pin group, such as the clk input, is fanned out to each latch in the bank. Each primary output must be described in a bus or bundle group, and its function statement must include either variable1 or variable2.

Three-state output pins are allowed; you can add a three_state attribute to an output bus or bundle to define this function.

The bits value in the latch_bank definition is the number of bits in the multibit cell.

**Example 5-9  4-Bit Register Modeled With latch_bank Group for Rising-Edge-Triggered D Latches**

cell (latch4) {
  area: 16;
  pin (G) { /* gate enable signal, active-high */
    direction : input;
    ...
  }  
bundle (D) { /* data input with four member pins */
    members(D1, D2, D3, D4);
    /*must be first bundle attribute*/
    direction : input;
    ...
  }
  bundle (Q) {
    members(Q1, Q2, Q3, Q4);
    direction : output;
    function : "IQ" ;
    ...
  }
  bundle (QN) {
    members (Q1N, Q2N, Q3N, Q4N);
    direction : output;
    function : "IQN";
    ...
  }
  latch_bank(IQ, IQN, 4) {
    enable : "G" ;
    data_in : "D" ;
  }
  ...
}
Example 5-10 is the cell description of the multibit register shown in Figure 5-2.

Example 5-10  4-Bit Register Modeled Using Enable-High D Latches With Clear

cell (DLT2) {

    /* note: 0 hold time */

    area : 1 ;
    pin (EN) {
        direction : input ;
        capacitance : 0 ;
    }
}
min_pulse_width_low : 3 ;
min_pulse_width_high : 3 ;
}

bundle (D) {
members(DA, DB, DC, DD);
direction : input ;
capacitance : 0 ;
timing() {
    related_pin : "EN" ;
timing_type : setup_falling ;
}

timing() {
    related_pin : "EN" ;
timing_type : hold_falling ;
}
}

bundle (CLR) {
members(CLRA, CLRB, CLRC, CLRD);
direction : input ;
capacitance : 0 ;
timing() {
    related_pin : "EN" ;
timing_type : recovery_falling ;
}
}

bundle (PRE) {
members(PREA, PREB, PREC, PRED);
direction : input ;
capacitance : 0 ;
timing() {
    related_pin : "EN" ;
timing_type : recovery_falling ;
}
}

latch_bank(IQ, IQN, 4) {
data_in : "D" ;
enable : "EN" ;
clear : "CLR ' " ;
preset : "PRE ' " ;
clear_preset_var1 : H ;
clear_preset_var2 : H ;
}

bundle (Q) {
members(QA, QB, QC, QD);
direction : output ;
function : "IQ" ;
timing() {
    related_pin : "D" ;
Describing Sequential Cells With the Statetable Format

The statetable format provides an intuitive way to describe the function of complex sequential cells. Using this format, the library developer can translate a state table in a databook to a cell description.
OUT and OUTN
Sequential output ports of the sequential cell.

IN
The set of all primary input ports in the sequential cell functionally related to OUT and OUTN.

delay*
A small time delay. An asterisk suffix indicates a time delay.

Statetable
A sequential lookup table. The state table takes a number of inputs and their delayed values and a number of internal nodes and their delayed values to form an index to new internal node values. A sequential library cell can have only one state table.

Internal Nodes
As storage elements, internal nodes store the output values of the state table. There can be any number of internal nodes.

Output Logic
A combinational lookup table. For the sequential cells supported in ff or latch format, there are at most two internal nodes and the output logic must be a buffer, an inverter, a three-state buffer, or a three-state inverter.
To capture the function of complex sequential cells, use the `statetable` group in the `cell` group to define the statetable in Figure 5-3. The `statetable` group syntax maps to the truth tables in databooks as shown in the example of Figure 5-4. For table input token values, see “statetable Group” on page 5-26.

**Figure 5-4  Mapping Databook Truth Table to Statetable**

<table>
<thead>
<tr>
<th>Databook</th>
<th>Meaning</th>
<th>Statetable input</th>
<th>Statetable output</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>Fall</td>
<td>F</td>
<td>N/A</td>
</tr>
<tr>
<td>nc</td>
<td>No event</td>
<td>N/A</td>
<td>N</td>
</tr>
<tr>
<td>r</td>
<td>Rise</td>
<td>R</td>
<td>N/A</td>
</tr>
<tr>
<td>tg</td>
<td>Toggle</td>
<td>N/A</td>
<td>(1)</td>
</tr>
<tr>
<td>u</td>
<td>Undefined</td>
<td>N/A</td>
<td>X</td>
</tr>
<tr>
<td>x</td>
<td>Don't Care</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>-</td>
<td>Not used</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

To map a databook truth table to a statetable, do the following:

1. When the databook truth table includes the name of an input port, replace that port name with the tokens for low/high (L/H).
2. When the databook truth table includes the name of an output port, use L/H for the current value of the output port and the next value of the output port.
3. When the databook truth table has the toggle flag tg (1), use L/H for the current value of the output port and H/L for the next value of the output port.

In the truth table, an output port preceded with a tilde symbol (~) is inverted. Sometimes you must map f to ~R and r to ~F.
**statetable Group**

The **statetable** group contains a table consisting of a single string.

**Syntax**

```plaintext
statetable("input node names", "internal node names")
{
    table : "input node values : current internal values \ 
        : next internal values,"
    input node values : current internal values : next internal values";
}
```

You need to follow these conventions when using a **statetable** group:

- Give nodes unique names.
- Separate node names with white space.
- Place each rule consisting of a set of input node values, current internal values, and next internal values on a separate line, followed by a comma and the line continuation character (\). To prevent syntax errors, the line continuation character must be followed immediately by the next line character.
- Separate node values and the colon delimiter (:) with white space.
- Insert comments only where a character space is allowed. For example, you cannot insert a comment within an H/L token or after the line continuation character (\).

**Figure 5-5  **statetable Group Example**

![statetable Group Example](image)
### Table 5-6  Legitimate Values for Table Inputs (Input and Current Internal Nodes)

<table>
<thead>
<tr>
<th>Input node values</th>
<th>Current internal node values</th>
<th>State represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>Low</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>High</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>Don’t care</td>
</tr>
<tr>
<td>L/H</td>
<td>L/H</td>
<td>Expands to both L and H</td>
</tr>
<tr>
<td>H/L</td>
<td>H/L</td>
<td>Expands to both H and L</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>Rising edge (from low to high)</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>Falling edge (from high to low)</td>
</tr>
<tr>
<td>~R</td>
<td></td>
<td>Not rising edge</td>
</tr>
<tr>
<td>~F</td>
<td></td>
<td>Not falling edge</td>
</tr>
</tbody>
</table>

### Table 5-7  Legitimate Values for Next Internal Node

<table>
<thead>
<tr>
<th>Next internal node values</th>
<th>State represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Low</td>
</tr>
<tr>
<td>H</td>
<td>High</td>
</tr>
<tr>
<td>-</td>
<td>Output is not specified</td>
</tr>
<tr>
<td>L/H</td>
<td>Expands to both L and H</td>
</tr>
<tr>
<td>H/L</td>
<td>Expands to both H and L</td>
</tr>
<tr>
<td>X</td>
<td>Unknown</td>
</tr>
<tr>
<td>N</td>
<td>No event from current value. Hold. Use only when all asynchronous inputs and clocks are inactive</td>
</tr>
</tbody>
</table>
Note:
It is important to use the N token value appropriately. The output is never N when any input (asynchronous or synchronous) is active. The output should be N only when all the inputs are inactive.

### Using Shortcuts

To represent a statetable explicitly, list the next internal node one time for every permutation of L and H for the current inputs, previous inputs, and current states.

**Example 5-11  Fully Expanded statetable Group for a Data Latch With Active-Low clear**

```plaintext
statetable(" G  CD  D", "IQ") {
table : " L L L : L :L,
       L L L : H :L,
       L L H : L :L,
       L L H : H :L,
       L H L : L :N,
       L H L : H :N,
       L H H : L :N,
       L H H : H :N,
       L H H : L :L,
       L H H : H :L,
       L H L : L :L,
       L H L : H :L,
       L H H : L :L,
       L H H : H :L,
       L H L : L :L,
       L H L : H :L,
       L H H : L :L,
       L H H : H :L,
       L H L : L :L,
       L H L : H :L,
       L H H : L :L,
       L H H : H :L",
}
```

You can use the following shortcuts when you represent your table.

**Don’t care symbol (-)**

For the input and current internal node, the don’t care symbol represents all permutations of L and H. For the next internal node, the don’t care symbol means the rule does not define a value for the next internal node.

For example, a master-slave flip-flop can be written as follows:

```plaintext
statetable("D  CP  CPN", "MQ  SQ") {
table : "H/L  R  N  ~F  -  -  -  : H/L  N,
       - ~R  F  H/L  -  N  H/L,\n       H/LR  F  L  -  H/L  L,\n       H/LR  F  H  -  H/L  H,\n       - ~R  ~F  -  -  -  : N  N",
}
```

Or it can be written more concisely as follows

```plaintext
statetable(" D  CP  CPN", "MQ  SQ") {
table : " H/L  R  -  -  -  : H/L  -,
```
L/H and H/L

Both L/H and H/L represent two individual lines: one with L and the other with H. For example, the following line

\[
\text{H H H/L : - : L/H,}
\]

is a concise version of the following lines.

\[
\text{H H H : - : L,}
\]
\[
\text{H H L : - : H,}
\]

R, ~R, F, and ~F (input edge-sensitive symbols)

The input edge-sensitive symbols represent permutations of L and H for the delayed input and the current input. Every edge-sensitive input, one that has at least one input edge symbol, is expanded into two level-sensitive inputs: the current input value and the delayed input value. For example, the input edge symbol R expands to an L for the delayed input value and to an H for the current input value. In the following statetable of a D flip-flop, clock C can be represented by the input pair C* and C. C* is the delayed input value, and C is the current input value.

\[
\text{statetable ("C D", "IQ") { }
\]
\[
\text{table : "R L/H : - : L/H, ~R - : - : N";}
\]

Table 5-8 Internal Representation of the Cell

<table>
<thead>
<tr>
<th>C*</th>
<th>C</th>
<th>D</th>
<th>IQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>H</td>
<td>L/H</td>
<td>L/H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>-</td>
<td>N</td>
</tr>
</tbody>
</table>

Priority ordering of outputs

The outputs follow a prioritized order. Two rules can cover the same input combinations, but the rule defined first has priority over subsequent rules.
Chapter 5: Defining Sequential Cells
Describing Sequential Cells With the Statetable Format

Note:
Use shortcuts with care. When in doubt, be explicit.

---

**Partitioning the Cell Into a Model**

You can partition a structural netlist to match the general sequential output model described in Example 5-12 by performing these tasks:

1. Represent every storage element by an internal node.
2. Separate the output logic from the internal node. The internal node must be a function of only the sequential cell data input when the sequential cell is triggered.

Note:
There are two ways to specify that an output does not change logic values. One way is to use the inactive N value as the next internal value, and the other way is to have the next internal value remain the same as the current internal value (see the italic lines in Example 5-12).

**Example 5-12  JK Flip-Flop With Active-Low, Direct-Clear, and Negative-Edge Clock**

```statetable ("J K CN CD", "IQ") {
  table : "
}
```

In Example 5-12, the value of the next internal node of the second rule is N, because both the clear CD and clock CN are inactive. The value of the next internal node of the third rule is L/H, because the clock is active.

---

**Defining an Output pin Group**

Every output pin in the cell has either an internal_node attribute, which is the name of an internal node, or a state_function attribute, which is a Boolean expression of ports and internal pins.

Every output pin can also have an optional three_state expression.

An output pin can have one of the following combinations:

- A state_function attribute and an optional three_state attribute
- An internal_node attribute, an optional input_map attribute, and an optional three_state attribute
**state_function Attribute**

The `state_function` attribute defines output logic. Ports in the `state_function` Boolean expression must be either input, inout that can be made three-state, or ports with an `internal_node` attribute.

The `state_function` attribute specifies the purely combinational function of input and internal pins. A port in the `state_function` expression refers only to the non-three-state functional behavior of that port. For example, if E is a port of the cell that enables the three-state, the `state_function` attribute cannot have a Boolean expression that uses pin E.

An inout port in the `state_function` expression is treated only as an input port.

Note:

Use the `state_function` attribute to define a cell with a state table. Use this attribute when the functional expression does not reference any state table signals, and is a function of only the input pins.

**Example**

```lisp
pin(Q)
  direction : output;
  state_function : "A"; /*combinational feedthrough*/
```

**internal_node Attribute**

The `internal_node` attribute is used to resolve node names to real port names.

The statetable is in an independent, isolated name space. The term `node` refers to the identifiers. Input node and internal node names can contain any characters except white space and comments. These node names are resolved to the real port names by each port with an `internal_node` attribute.

Each output defined by the `internal_node` attribute might have an optional `input_map` attribute.

**Example**

```lisp
internal_node :
  "IQ";
```

**input_map Attribute**

The `input_map` attribute maps real port and internal pin names to each table input and internal node specified in the state table. An input map is a listing of port names, separated by white space, that correspond to internal nodes.
Example

input_map : "Gx1 CDx1 Dx1 QN"; /*QN is internal node*/

Mapping port and internal pin names to table input and internal nodes occurs by using a combination of the `input_map` attribute and the `internal_node` attribute. If a node name is not mapped explicitly to a real port name in the input map, it automatically inherits the node name as the real port name. The internal node name specified by the `internal_node` attribute maps implicitly to the output being defined. For example, to map two input nodes, A and B, to real ports D and CP, and to map one internal node, Z, to port Q, set the `input_map` attribute as follows:

input_map : "D CP Q"

You can use a don’t care symbol in place of a name to signify that the input is not used for this output. Comments are allowed in the `input_map` attribute.

The delayed nature of outputs specified with the `input_map` attribute is implicit:

Internal node

When an output port is specified for this type of node, the internal node is forced to map to the delayed value of the output port.

Synchronous data input node

When an output port is specified for this type of node, the input is forced to map to the delayed value of the output port.

Asynchronous or clock input node

When an output port is specified for this type of node, the input is forced to map to the undelayed value of the output port.

Figure 5-6  Circuit With Latch U1 and Flip-Flop U2

For example, in Figure 5-6, internal pin n1 should map to ports “Q0 G1 n1*” and output Q0 should map to “n1* CP2 Q0**”. The subtle significance is relevant during simultaneous input transitions.
With this input_map attribute, the functional syntax for ganged cells is identical to that of nonganged cells. You can use the input map to represent the interconnection of any network of sequential cells (for example, shift registers and counters).

The input_map attribute can be completely unspecified, completely specified, or can specify only the beginning ports. The default for an incompletely defined input map is the assumption that missing trailing primary input nodes and internal nodes are equal to the node names specified in the statetable. The current state of the internal node should be equal to the output port name.

**inverted_output Attribute**

You can define output as inverted or noninverted by using an inverted_output attribute on the pin. The inverted_output attribute is a required Boolean attribute for the statetable format that you can set for any output port. Set this attribute to false for noninverting output. This is variable1 or IQ for flip-flop or latch groups. Set this attribute to true for inverting output. This is variable2 or IQN for flip-flop or latch groups.

**Example**

```plaintext
inverted_output : true;
```

In the statetable format, an output is considered inverted if it has any data input with a negative sense. This algorithm might be inconsistent with a user’s intentions. For example, whether a toggle output is considered inverted or noninverted is arbitrary.

---

**Internal Pin Type**

In the flip-flop or latch format, the pin, bus, or bundle groups can have a direction attribute with input, output, or inout values. In the statetable format, the direction attribute of a pin of a library cell (combinational or sequential) can have the internal value. A pin with the internal value to the direction attribute is treated like an output port, except that it is hidden within the library cell. It can take any of the attributes of an output pin, but it cannot have the three_state attribute.

An internal pin can have timing arcs and can have timing arcs related to it, allowing a distributed delay model for multistate sequential cells.

**Example**

```plaintext
pin (n1) {
  direction : internal;
  internal_node :"IQ"; /* use default input_map */
  ...
}
pin (QN) {
  direction : output;
```
internal_node : "IQN";
input_map : "Gx1 CDx1 Dx1 QN"; /* QN is internal node */

three_state : "EN2";

} 

pin (QNZ) {
  direction : output;
  state_function : "QN"; /* ignores QN’s three state */

  three_state : "EN";
  ...
}

pin (Y) {
  direction : output;
  state_function : "A"; /* combinational feedthrough */
  ...
}

---

Determining a Complex Sequential Cell’s Internal State

You can use the `power_down_function` string attribute to specify the Boolean condition under which the cell’s output pin is switched off by the state of the power and ground pins (when the cell is in off mode due to the external power pin states). The attribute should be set under the `statetable` group. The `power_down_function` attribute also determines the condition of the cell’s internal state.

For more information about the `power_down_function` attribute, see “`power_down_function Attribute` on page 5-6.

`power_down_function` Syntax for State Tables

The `power_down_function` attribute is specified after `table`, in the `statetable` group, as shown in the following syntax:

```
statetable( "input node names", "internal node names" ){ 
  table : " input node values : current internal values :
  next internal values,
  input node values : current internal values: 
  next internal values" ;
  power_down_function : "Boolean expression" ;
}
```
Flip-Flop and Latch Examples

Example 5-13  D Flip-Flop

/* ff/latch format */

ff (IQ,IQN) {
    next_state : "D" ;
    clocked_on : "CP" ;
}
/* statetable format */

statetable(" D CP", "IQ") {
    table : " H/L R : - : H/L,\n            - ~R : - : N";
}

Example 5-14  D Flip-Flops With Master-Slave Clock Input Pins

/* ff/latch format */

ff (IQ,IQN) {
    next_state : "D" ;
    clocked_on : "CK" ;
    clocked_on_also : "CKN'" ;
}
/* statetable format */

statetable(" D CK CKN", "MQSQ") {
}

Example 5-15  D Flip-Flop With Gated Clock

/* ff/latch format */

ff (IQ,IQN) {
    next_state : "D" ;
    clocked_on : "C1 * C2" ;
}
/* statetable format */

statetable(" D C1 C2", "IQ") {
Example 5-16  D Flip-Flop With Active-Low Direct-Clear and Active-Low Direct-Set
/* ff/latch format */
ff (IQ, IQN) {
    next_state : "D" ;
    clocked_on : "CP" ;
    clear : " CD' " ;
    preset : " SD' " ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}
/* statetable format */
statetable("D CP CD SD", "IQ IQN") {
}

Example 5-17  D Flip-Flop With Active-Low Direct-Clear, Active-Low Direct-Set, and One Output
/* ff/latch format */
ff (IQ, IQN) {
    next_state : "D" ;
    clocked_on : "CP" ;
    clear : " CD' " ;
    preset : " SD' " ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}
/* statetable format */
statetable(" D CP CD SD", "IQ") {
}

Example 5-18  JK Flip-Flop With Active-Low Direct-Clear and Negative-Edge Clock
/* ff/latch format */
ff (IQ, IQN) {
next_state : " (J' K' IQ) + (J K') + (J K IQ')" 
   ;
clocked_on : " CN'" 
clear : " CD'" 
}
/* statetable format */
statetable(" J   K   CD   CD", "IQ") {
table : " | -   -   -    L    :  -  :   L,
       : -   -   ~F   H    :  -  :   N,
L | L   L   F    H    : L/H :   L/H 
H | L   L   F    H    : -   :   H,
L | H   F   F    H    : -   :   L,
H | H   F   F    H    : L/H :   H/L";

Example 5-19  D Flip-Flop With Scan Input Pins
/* ff/latch format */
ff (IQ, IQN) {
   next_state : " (D TE') + (TI TE)" 
   ;
clocked_on : "CP" 
}
/* statetable format */
statetable(" D    TE  TI  CP", "IQ") {
table : " | H/L  L  -    R  : - :  H/L,
       : -    H  H/L  R  : - :  H/L,
       : -    -  -    ~R : - :  N";
}

Example 5-20  D Flip-Flop With Synchronous Clear
/* ff/latch format */
ff (IQ, IQN) {
   next_state : "D CR" 
   ;
clocked_on : "CP" 
}
/* statetable format */
statetable(" D    CR  CP", "IQ") {
table : " | H/L  H  R   : - :  H/L,
       : -    L  R   : - :  L,
       : -    -  ~R  : - :  N";
}

Example 5-21  D Latch
/* ff/latch format */
latch (IQ,IQN) {
  data_in : "D" ;
  enable : "G" ;
}

/* statetable format */
statetable(" D G", "IQ") {
  table : " H/L H : - : H/L,\-
           - L : - : N";
}

Example 5-22  SR Latch

/* ff/latch format */
latch (IQ,IQN) {
  clear : "R" ;
  preset : "S" ;
  clear_preset_var1 : L ;
  clear_preset_var2 : L ;
}

/* statetable format */
statetable(" R S",   "IQ IQN") {
  table : " H L : - - : L H,\
          L H : - - : H L,\
          H H : - - : L L,\
          L L : - - : N N";
}

Example 5-23  D Latch With Active-Low Direct-Clear

/* ff/latch format */
latch (IQ,IQN) {
  data_in : "D" ;
  enable : "G" ;
  clear : " CD' " ;
}

/* statetable format */
statetable(" D G CD",   "IQ") {
  table : " H/L H H : - : H/L,\
           - L H : - : N,\
           - - L : - : L";
}
Example 5-24  Multibit D Latch With Active-Low Direct-Clear

/* ff/latch format */

latch_bank(IQ, IQN, 4) {
  data_in : "D" ;
  enable  : "EN" ;
  clear   : "CLR'" ;
  preset  : "PRE'" ;
  clear_preset_var1 : H ;
  clear_preset_var2 : H ;
}

/* statetable format */

statetable(" D   EN  CL  PRE","IQ IQN") {
}

Example 5-25  D Flip-Flop With Scan Clock

statetable(" D   S  CD SC CP",  "IQ") {
  table : " H/L  -    ~R  R   : - :  H/L,
         -    H/L  R   ~R  : - :  H/L,
         -    -    ~R  ~R  : - :  N,
         -    -    R   R   : - :  X";
}

Cell Description Examples

Example 5-26  D Latches With Master-Slave Enable Input Pins

cell(ms_latch) {
  area : 16;
  pin(D) {
    direction : input;
    capacitance : 1;
  }
  pin(G1) {
    direction : input;
    capacitance : 2;
  }
  pin(G2) {
    direction : input;
    capacitance : 2;
  }
  pin(mq) {"
internal_node : "Q";
direction : internal;
input_map : "D G1"

timing() {
    related_pin : "G1"
}

pin(Q) {
    direction : output;
    function : "IQ"

    internal_node : "Q"

    input_map : "mq G2"

    timing() {
        related_pin : "G2"
    }
}

pin(QN) {
    direction : output;
    function : "IQN"

    internal_node : "QN"

    input_map : "mq G2"

    timing() {
        related_pin : "G2"
    }
}

ff(IQ, IQN) {
    clocked_on : "G1";
    clocked_on_also : "G2"

    next_state : "D"
}

statetable ( "D G", "Q QN" ) {
    table : "L/H H : - - : L/H H/L,\n    - L : - - : N N";
}

Example 5-27  FF Shift Register With Timing Removed

cell(shift_reg_ff) {
    area : 16;

    pin(D) {
        direction : input;
        capacitance : 1;
    }

    pin(CP) {
        direction : input;
        capacitance : 2;
    }

    pin(Q0) {
        direction : output;
        internal_node : "Q"

        input_map : "D CP"
    }
}
pin (Q1) {
    direction : output;
    internal_node : "Q";
    input_map : "Q0 CP";
}

pin (Q2) {
    direction : output;
    internal_node : "Q";
    input_map : "Q1 CP";
}

pin (Q3) {
    direction : output;
    internal_node : "Q";
    input_map : "Q2 CP";
}

statetable( "D CP", "Q QN" ) {
    table : "-   ~R : - - : N    N,
           H/L  R : - - : H/L  L/H";
}

Example 5-28  FF Counter With Timing Removed

cell(counter_ff) {
    area : 16;
    pin(reset) {
        direction : input;
        capacitance : 1;
    }
    pin(CP) {
        direction : input;
        capacitance : 2;
    }
    pin (Q0) {
        direction : output;
        internal_node : "Q0";
        input_map : "CP reset Q0 Q1";
    }
    pin (Q1) {
        direction : output;
        internal_node : "Q1";
        input_map : "CP reset Q0 Q1";
    }
    statetable( "CP reset", "Q0 Q1" ) {
    table : "-   L : - - : L H,
           ~R H : - - : N N,
           R H : L L : H L,
           R H : H L : L H,
           R H : L H : H H,
           R H : H H : L L";
    }

Defining Test Cells

A test cell contains information that supports a full-scan or a partial-scan methodology.

You must add test-specific details of scannable cells to your technology libraries. For example, you must identify scannable flip-flops and latches and select the types of unscannable cells they replace for a given scan methodology. To do this, you must understand the following concepts described in this chapter:

• Describing a Scan Cell
• Describing a Multibit Scan Cell
• Scan Cell Modeling Examples
Describing a Scan Cell

To specify a cell as a scan cell, add the test_cell group to the cell description.

Only the nontest mode function of a scan cell is modeled in the test_cell group. The nontest operation is described by its ff, ff_bank, latch, or latch_bank declaration and pin function attributes.

test_cell Group

The test_cell group defines only the nontest mode function of a scan cell.

Figure 6-1  test_cell Group Defined in a Scan Cell

Ensure the following when you define a test_cell group in a scan cell:

- Pin names in the scan cell and the test cell must match.
- The scan cell and the test cell must contain the same functional outputs.

Syntax

```lib_name` {cell (cell_name) {
  test_cell () {
    ... test cell
description ...
    pin ( name ) {
      ... pin description ...
    }
    pin ( name ) {
      ... name ) {
    pin description ...
  }
}`
```
You do not need to give the `test_cell` group a name, because the test cell takes the cell name of the cell being defined. The `test_cell` group can contain `ff`, `ff_bank`, `latch`, or `latch_bank` group statements and `pin` groups.

**Pins in the test_cell Group**

Both test pins and nontest pins can appear in `pin` groups within a `test_cell` group. These groups are like `pin` groups in a `cell` group but must adhere to the following rules:

- Each pin defined in the `cell` group must have a corresponding pin defined at the `test_cell` group level with the same name.
- The `pin` group can contain only `direction`, `function`, `test_output_only`, and `signal_type` attribute statements. The `pin` group cannot contain timing, capacitance, fanout, or load information.
- The `function` attributes can reflect only the nontest behavior of the cell.
- Input pins must be referenced in an `ff`, `ff_bank`, `latch`, or `latch_bank` statement or have a `signal_type` attribute assigned to them.
- An output pin must have either a `function` attribute, a `signal_type` attribute, or both.

---

**test_output_only Attribute**

This attribute indicates that an output port is set for test only (as opposed to function only, or function and test).

**Syntax**

```
test_output_only : true | false ;
```

When you use statetable format to describe the functionality of a scan cell, you must declare the output pin as set for test only by setting the `test_output_only` attribute to `true`.

**Example**

```
test_output_only : true ;
```

---

**Describing a Multibit Scan Cell**

Multibit scan cells can have parallel or serial scan chains. The scan output of a multibit scan cell can reuse the data output pins or have a dedicated scan output (SO) pin in addition to the bus, or the bundle output pins.
Multibit scan cells with the following structures are supported:

- Parallel scan chain without dedicated SO bus
- Parallel scan chain with dedicated SO bus
- Serial scan chain without a dedicated SO pin
- Serial scan chain with a dedicated SO pin

---

**Multibit Scan Cell With Parallel Scan Chain**

A multibit scan cell with parallel scan chain has parallel data and scan inputs, and parallel functional and scan outputs.

Figure 6-2 shows the schematic diagram of a generic multibit scan cell with parallel input and output buses for the normal and scan modes.

In the normal mode, the cell uses the input and output buses, D[0:n-1] and Q[0:n-1], respectively.

The balloons represent combinational logic. The combinational logic at the input of each sequential element of the cell is a multiplexer with data and scan inputs. At the input of clock, CK, to each sequential element, the combinational logic is a clock pin or a clock-gating logic.

In the scan mode, the cell functions as a parallel-in parallel-out shift register with the scan input bus, SI[0:n-1], and either reuses the bus, Q[0:n-1], or uses a dedicated bus, SO[0:n-1], to output the scan data. The scan enable signal can be a bus, SE[0:n-1], where each bit of the bus enables a corresponding sequential element of the cell. The scan enable signal can also be a single-bit pin to enable each sequential element of the cell.
Use the following syntax to model the multibit scan cell shown in Figure 6-2:

```liberty
library(library_name) {
  . . .
  cell(cell_name) {
    . . .
    bus(scan_in_pin_name) {
      /* cell scan in with signal_type "test_scan_in" from test_cell */
      . . .
    }
    bus(scan_out_pin_name) {
      /* cell scan out with signal_type "test_scan_out" from test_cell */
      . . .
    }
    bus | bundle (bus_bundle_name) {
```
direction : input | output;
}
test_cell() {
  pin(scan_in_pin_name) {
    signal_type : test_scan_in;
    ...
  }
  pin(scan_out_pin_name) {
    signal_type : test_scan_out |
    test_scan_out_inverted;
    ...
  }
  ...
}

Example

4-bit Scan Cell With Parallel Scan Chains and Dedicated Scan Out Bus

Figure 6-3 shows the schematic of a 4-bit scan cell with parallel scan chains and a dedicated scan output bus, SO[0:3]. The figure also shows an equivalent single-bit cell.
Figure 6-3  4-Bit Scan Cell With Parallel Scan Chain and Single-Bit Equivalent Cell
Each pin of the scan output bus, SO[0:3], has AND logic. The cell is defined using:

- The statetable group
- The state_function attribute on the bus, SO

In the normal mode, the cell is a parallel shift register that uses the data input bus, D[0:3], and the data output bus, Q[0:3].

In the scan mode, the cell functions as a shift register with parallel scan input, scan enable, and scan output buses, SI[0:3], SE[0:3], and SO[0:3]. To use the cell in the scan mode, set the signal_type attribute as test_scan_out on the bus, SO, in the test_cell group. Do not define the state_function attribute for this bus in the test_cell group.

Example 6-1 describes a model of the multibit scan cell shown in Figure 6-3. The example specifically models the multibit scan cell, and not the single-bit degenerate cell.

**Example 6-1 Model of 4-Bit Scan Cell With Parallel Scan Chain and Dedicated Scan Output**

cell (4-bit_Parallel_Scan_Cell) {
    ...
    statetable ("D CK SI SE", "IQ, IQN") {
        table : "- ~R - - :- -: N N, \n        H/L R L - :- -: H/L H/L, \n        - R H H/L :- -: H/L L/H";
    }

    /* functional output bus */
    bus(Q) {
        bus_type : bus4;
        direction : output;
        function : IQ;
        ...
    }

    /* dedicated scan output bus */
    bus(SO) {
        bus_type : bus4;
        direction : output;
        state_function : "Q * SE" ;
    ...
}
pin(CK) {
   direction : input;
   ...
}

/* scan enable bus */
bus(SE) {
   bus_type : bus4;
   direction : input;
   ...
};

/* scan input bus */
bus(SI) {
   bus_type : bus4;
   direction : input;
   ...
};

/* data input bus pins */
bus(D) {
   bus_type : bus4;
   direction : input;
   ...
};

...

test_cell () {
   pin(CK){
      direction : input;
   }
   bus(D) {
      bus_type : bus4;
      direction : input;
   }
   bus(SI) {
      bus_type : bus4;
      direction : input;
      signal_type : "test_scan_in";
   }
   bus(SE) {
      bus_type : bus4;
      direction : input;
      signal_type : "test_scan_enable";
   }
   ff_bank (IQ,IQN,4) {
      next_state : "D";
      clocked_on : "CK";
   }
   bus(Q) {
      bus_type : bus4 ;
      direction : output;
      function : "IQ";
   }
   bus(SO) {
      bus_type : bus4;
   }
Multibit Scan Cell With Internal Scan Chain

A multibit scan cell with an internal scan chain has a serial scan chain and a dedicated pin to output the scan signal.

*Figure 6-4* shows the schematic diagram of a generic multibit scan cell with an internal or serial scan chain and the pin, SO, for the scan chain output.

In the normal mode, the cell is a shift register that uses the parallel input and output buses, D[0:n-1] and Q[0:n-1], respectively.

The balloons represent combinational logic. The combinational logic at the input of each sequential element of the cell is a multiplexer with data and scan inputs. At the input of clock, CK, to each sequential element, the combinational logic is clock-gating logic.

In the scan mode, the cell functions as a shift register with the single-bit output pin, SO. The serial scan chain is from the scan input (SI) pin to the scan output (SO) pin. The scan chain is stitched using the data output, Q, of each sequential element of the cell.
Figure 6-4  Generic Schematic Diagram of a Multibit Scan Cell With Serial Scan Chain

- SI
- SE
- CP
- D[0,1...,n-1]
- Other Inputs
- Q[0]
- Q[1]
- Q[m]
- Q[n-1]
- SO
- Element 0
- D
- Q
- CK
- Scan Out
- Element 1
- D
- Q
- CK
- Scan Out
- Element n-1
- D
- Q
- CK
- Scan Out
Use the following syntax to model the multibit scan cell shown in Figure 6-4:

```plaintext
library(library_name) {
  ...  
  cell(cell_name) {
    pin(scan_in_pin_name) {
      /* cell scan in with signal_type "test_scan_in" from test_cell */
      ...
    }
    pin(scan_out_pin_name) {
      /* cell scan out with signal_type "test_scan_out" from test_cell */
      ...
    }
    bus | bundle (bus_bundle_name) {
      direction : inout | output;
    }
    test_cell() {
      pin(scan_in_pin_name) {
        signal_type : test_scan_in;
        ...
      }
      pin(scan_out_pin_name) {
        signal_type : test_scan_out | test_scan_out_inverted;
        ...
      }
      ...  
    }
  }  
  ...  
}
```
Examples

4-Bit Scan Cell With Internal Scan Chain and Dedicated Scan Out Pin

Figure 6-5  4-Bit Scan Cell With Internal Scan Chain and Dedicated Scan Out Pin, SO

The 4-bit cell has the output bus Q[0:3], and a single-bit output pin (SO) with combinational logic. The cell is defined using:

- The `statetable` group
- The `state_function` attribute on the pin, SO

In the normal mode, the cell is a shift register that uses the output bus Q[0:3].

In scan mode, the cell functions as a shift register with the single-bit output pin, SO. To use the cell in scan mode, set the `signal_type` attribute as `test_scan_out` on the pin, SO, in the `test_cell` group. Do not define the `function` attribute of this pin.

Example 6-2 describes a model of the multibit scan cell shown in Figure 6-5. The example specifically models the multibit scan cell, and not the single-bit degenerate cell, SB.
Example 6-2  Model of 4-Bit Multibit Scan Cell With Serial Scan Chain

cell (4-bit_Serial_Scan_Chain) {
  ...
  statetable ( " D  CK  SE  SI " ,  "Q" ) {
    table : " - ~R - - : - : N, \ 
            H/L  R  L - : - : H/L, \ 
            -  R  H  H/L : - : H/L" ;}
  bus(Q) {
    bus_type : bus4;
    direction : output;
    internal_node: Q;
    pin (Q[0]) { input_map : " D[0]  CK  SE  SI " ; }
    pin (Q[1]) { input_map : " D[1]  CK  SE  Q[0] " ; }
  } /* dedicated scan output pin */
  pin(SO) {
    direction : output;
    inverted_output : false;
    state_function : "Q[3] * SE" ;
    ...
  }
  pin(CK) {
    direction : input;
    ...
  }
  pin(SE) {
    direction : input;
    ...
  } /* scan input pin */
  pin(SI) {
    direction : input;
    ...
  } /* data input bus pins */
  bus(D) {
    direction : input;
    bus_type : bus4;
    ...
  }
  ...
  test_cell () {
    pin(CK) {
      direction : input;
    }
    bus(D) {
      bus_type : bus4;
      direction : input;
    }
pin(SI) {
   direction : input;
   signal_type : "test_scan_in";
}

pin(SE) {
   direction : input;
   signal_type : "test_scan_enable";
}

ff_bank (IQ,IQN,4) {
   next_state : "D";
   clocked_on : "CK";
}

bus(Q) {
   bus_type : bus4;
   direction : output;
   function : "IQ";
}

pin(SO) {
   direction : output;
   signal_type : "test_scan_out";
   test_output_only : "true";
}

2-Bit Scan Cell Without Dedicated SO Pin

In Figure 6-6, the scan cell has scan and enable inputs and no dedicated scan output pin. The output pin, Q1, is reused to output the scan signal.
The following example shows the relevant syntax to model the cell of Figure 6-6. In this example, the statetable group describes the behavior of a single sequential element of the cell. The input_map statements on pin Q0 and pin Q1 indicate the interconnections between the sequential elements—for example, Q of bit 0 goes to TI of bit 1.

cell (FSX2) { ...
  bundle (D) {
    members (D0, D1);
    ...
  }
  bundle (Q) {
    members (Q0, Q1);
    ...
    pin (Q0) {
      input_map : "D0 T SI SM";
      ...
    }
    pin (Q1) {
      input_map : "D1 T Q0 SM";
      ...
    }
  }
  pin (SI) {
    ...
  }
  pin (SM) {
    ...
  }
  pin (T) {
Scan Cell Modeling Examples

This section contains modeling examples for these test cells:

- Simple multiplexed D flip-flop
- Multibit cells with multiplexed D flip-flop and enable
- LSSD (level-sensitive scan design) scan cell
- Clocked-scan test cell
- Scan D flip-flop with auxiliary clock

Each example contains a complete cell description.

Simple Multiplexed D Flip-Flop

Example 6-3  Simple Multiplexed D Flip-Flop Scan Cell

cell(SDFF1) {
    area : 9;
    pin(D) {
        direction : input;
        capacitance : 1;
        timing() {...}
    }
    pin(CP) {
        direction : input;
        capacitance : 1;
        timing() {...}
    }
    pin(TI) {
        direction : input;
        capacitance : 1;
        timing() {...}
    }
}
pin(TE) {
    direction : input;
    capacitance : 2;
    timing() {...}
}
ff(IQ,IQN) {
    /* model full behavior (if possible): */
    next_state : "D TE’ + TI TE’";
    clocked_on : "CP";
}
pin(Q) {
    direction : output;
    function : "IQ";
    timing() {...}
}
pin(QN) {
    direction : output;
    function : "IQN";
    timing() {...}
}
test_cell() {
    pin(D) {
        direction : input
    }
    pin(CP) {
        direction : input
    }
    pin(TI) {
        direction : input;
        signal_type : test_scan_in;
    }
    pin(TE) {
        direction : input;
        signal_type : test_scan_enable;
    }
    ff(IQ,IQN) {
        /* just model non-test operation behavior */
        next_state : "D";
        clocked_on : "CP";
    }
    pin(Q) {
        direction : output;
        function : "IQ";
        signal_type : test_scan_out;
    }
    pin(QN) {
        direction : output;
        function : "IQN";
        signal_type : test_scan_out_inverted;
    }
}
Multibit Cells With Multiplexed D Flip-Flop and Enable

Example 6-4  Library Containing Multibit Scan Cells With Multiplexed D Flip-Flops and Enable

```liberty
library(banktest) {
  ...
  default_inout_pin_cap : 1.0;
  default_input_pin_cap  : 1.0;
  default_output_pin_cap : 0.0;
  default_fanout_load   : 1.0;
  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1uA";
  pulling_resistance_unit : "1kohm";
  capacitive_load_unit (0.1,ff);
  type (bus4) {
    base_type : array;
    data_type : bit;
    bit_width : 4;
    bit_from  : 0;
    bit_to    : 3;
  }
  cell(FDSX4) {
    area : 36;
    bus(D) {
      bus_type : bus4;
      direction : input;
      capacitance : 1;
      timing() {
        timing_type : setup_rising;
        related_pin : "CP";
      }
      timing() {
        timing_type : hold_rising;
        related_pin : "CP";
      }
    }
    pin(CP) {
      direction : input;
      capacitance : 1;
    }
    pin(TI) {
      direction : input;
      capacitance : 1;
      timing() {
        timing_type : setup_rising;
        related_pin : "CP";
      }
      timing() {
        timing_type : hold_rising;
        related_pin : "CP";
      }
    }
  }
}
```
pin(TE) {
  direction : input;
  capacitance : 2;
  timing() {
    timing_type : setup_rising;
    related_pin : "CP";
  }
  timing() {
    timing_type : hold_rising;
    related_pin : "CP";
  }
}

statetable (" D CP TI TE ", ", Q QN") {
  table :
    " - ~R - - : - - : N N, \
    - R H/L H : - - : H/L L/H, \n    H/L R - L : - - : H/L L/H" ;
}

bus(Q) {
  bus_type : bus4;
  direction : output;
  inverted_output : false;
  internal_node : "Q";
  timing() {
    timing_type : rising_edge;
    related_pin : "CP";
  }
  pin(Q[0]) {
    input_map : "D[0] CP TI TE";
  }
  pin(Q[1]) {
    input_map : "D[1] CP Q[0] TE";
  }
  pin(Q[2]) {
  }
  pin(Q[3]) {
  }
}

bus(QN) {
  bus_type : bus4;
  direction : output;
  inverted_output : true;
  internal_node : "QN";
  timing() {
    timing_type : rising_edge;
    rise_transition(scalar) {values(" 0.1458 "); }
    fall_transition(scalar) {values(" 0.0523 "); }
    related_pin : "CP";
  }
  pin(QN[0]) {
    input_map : "D[0] CP TI TE";
  }
}
pin(QN[1]) {
    input_map : "D[1] CP Q[0] TE";
}
pin(QN[2]) {
}
pin(QN[3]) {
}
test_cell() {
    bus (D) {
        bus_type : bus4;
        direction : input;
    }
    pin(CP) {
        direction : input;
    }
    pin(TI) {
        direction : input;
        signal_type : "test_scan_in";
    }
    pin(TE) {
        direction : input;
        signal_type : "test_scan_enable";
    }
    ff_bank("IQ","IQN", 4) {
        next_state : "D";
        clocked_on : "CP";
    }
    bus(Q) {
        bus_type : bus4;
        direction : output;
        function : "IQ";
        signal_type : "test_scan_out";
    }
    bus(QN) {
        bus_type : bus4;
        direction : output;
        function : "IQN";
        signal_type : "test_scan_out_inverted";
    }
}
cell(SCAN2) {
    area : 18;
    bundle(D) {
        members(D0, D1);
        direction : input;
        capacitance : 1;
        timing() {
            timing_type : setup_rising;
        }
    }
}
related_pin : "T";
}

timing() {
    timing_type : hold_rising;
    related_pin : "T";
}
}

pin(T) {
    direction : input;
    capacitance : 1;
}

pin(EN) {
    direction : input;
    capacitance : 2;
    timing() {
        timing_type : setup_rising;
        related_pin : "T";
    }
    timing() {
        timing_type : hold_rising;
        related_pin : "T";
    }
}

pin(SI) {
    direction : input;
    capacitance : 1;
    timing() {
        timing_type : setup_rising;
        related_pin : "T";
    }
    timing() {
        timing_type : hold_rising;
        related_pin : "T";
    }
}

pin(SM) {
    direction : input;
    capacitance : 2;
    timing() {
        timing_type : setup_rising;
        related_pin : "T";
    }
    timing() {
        timing_type : hold_rising;
        related_pin : "T";
    }
}

statetable ( " T  D  EN  SI  SM", " Q  QN") {
    table : " ~R  -  -  -  -  :  -  -  :  N  N , \ 
            -  -  L  -  L  :  -  -  :  N  N , \ 
            R  H/L  H  -  L  :  -  -  :  H/L  L/H, \ 
            R  -  -  H/L  H  :  -  -  :  H/L  L/H";
}
bundle(Q) {
    members(Q0, Q1);
    direction : output;
    inverted_output : false;
    internal_node : "Q";
    timing()
    {
        timing_type : rising_edge;
        related_pin : "T";
    }
    pin(Q0) {
        input_map : "T D0 EN SI SM";
    }
    pin(Q1) {
        input_map : "T D1 EN Q0 SM";
    }
}

bundle(QN) {
    members(Q0N, Q1N);
    direction : output;
    inverted_output : true;
    internal_node : "QN";
    timing()
    {
        timing_type : rising_edge;
        related_pin : "T";
    }
    pin(Q0N) {
        input_map : "T D0 EN SI SM";
    }
    pin(Q1N) {
        input_map : "T D1 EN Q0 SM";
    }
}

test_cell() {
    bundle(D) {
        members(D0, D1);
        direction : input;
    }
    pin(T) {
        direction : input;
    }
    pin(EN) {
        direction : input;
    }
    pin(SI) {
        direction : input;
        signal_type : "test_scan_in";
    }
    pin(SM) {
        direction : input;
        signal_type : "test_scan_enable";
    }
    ff_bank("IQ", "IQN", 2) {
        next_state : "D";
    }
clocked_on : "T EN";
}
bundle(Q) {
  members(Q0, Q1);
  direction : output;
  function : "IQ";
  signal_type : "test_scan_out";
}
bundle(QN) {
  members(Q0N, Q1N);
  direction : output;
  function : "IQN";
  signal_type : "test_scan_out_inverted";
}

---

**LSSD Scan Cell**

**Example 6-5  LSSD Scan Cell**

```plaintext
cell(LSSD) {
  area : 12;
  pin(D) {
    direction : input;
    capacitance : 1;
    timing() {
      timing_type : setup_falling;
      related_pin : "MCLK";
    }
    timing() {
      timing_type : hold_falling;
      related_pin : "MCLK";
    }
  }
  pin(SI) {
    direction : input;
    capacitance : 1;
    prefer_tied : "0";
    timing() {
      timing_type : setup_falling;
      related_pin : "ACLK";
    }
    timing() {
      timing_type : hold_falling;
      related_pin : "ACLK";
    }
  }
  pin(MCLK, ACLK, SCLK) {
    direction : input;
    capacitance : 1;
  }
```

Chapter 6: Defining Test Cells
Scan Cell Modeling Examples
Chapter 6: Defining Test Cells
Scan Cell Modeling Examples

pin(Q1) {
    direction : output;
    internal_node : "Q1";
    timing() {
        timing_type : rising_edge;
        related_pin : "MCLK";
    }
    timing() {
        timing_type : rising_edge;
        related_pin : "ACLK";
    }
    timing() {
        related_pin : "D";
    }
    timing() {
        related_pin : "SI";
    }
}

pin(Q1N) {
    direction : output;
    state_function : "Q1'";
    timing() {
        timing_type : rising_edge;
        related_pin : "MCLK";
    }
    timing() {
        timing_type : rising_edge;
        related_pin : "ACLK";
    }
    timing() {
        related_pin : "D";
    }
    timing() {
        related_pin : "SI";
    }
}

pin(Q2) {
    direction : output;
    internal_node : "Q2";
    timing() {
        timing_type : rising_edge;
        related_pin : "SCLK";
    }
}

pin(Q2N) {
    direction : output;
    state_function : "Q2'";
    timing() {
        timing_type : rising_edge;
        related_pin : "SCLK";
    }
}

statetable("MCLK D ACLK SCLK SI", "Q1 Q2") {
scan_cell() /* for DLATCH */
    pin(D,MCLK) {
        direction : input;
    }
    pin(SI) {
        direction : input;
        signal_type : "test_scan_in";
    }
    pin(ACLK) {
        direction : input;
        signal_type : "test_scan_clock_a";
    }
    pin(SCLK) {
        direction : input;
        signal_type : "test_scan_clock_b";
    }
    latch ("IQ","IQN") {
        data_in : "D";
        enable : "MCLK";
    }
    pin(Q1) {
        direction : output;
        function : "IQ";
    }
    pin(Q1N) {
        direction : output;
        function : "IQN";
    }
    pin(Q2) {
        direction : output;
        signal_type : "test_scan_out";
    }
    pin(Q2N) {
        direction : output;
        signal_type : "test_scan_out_inverted";
    }
}

scan_cell() /* for MSFF1 */
    pin(D,MCLK,SCLK) {
        direction : input;
    }
    pin(SI) {
        direction : input;
        signal_type : "test_scan_in";
    }
    pin(ACLK) {

```c
" L - L - - - : - - : N - , \nH L/H L - - - - : - - : L/H - , \nL - H - L/H : - - - L/H - , \nH - H - - - : - - : X - , \n- - - L - - - - : - - : - N , \n- - - H - - : L/H - - L/H";
```
direction : input;
signal_type : "test_scan_clock_a";
}
ff ("IQ","IQN") {
    next_state : "D";
clocked_on : "MCLK";
clocked_on_also : "SCLK";
}
pin(Q1,Q1N) {
    direction : output;
}
pin(Q2) {
    direction : output;
    function : "IQ";
signal_type : "test_scan_out";
}
pin(Q2N) {
    direction : output;
    function : "IQN";
signal_type : "test_scan_out_inverted";
}
}

Note:
For latch-based designs, an LSSD scan cell has two test_cell groups for use in either single-latch or double-latch mode.

---

### Scan-Enabled LSSD Cell

The scan-enabled LSSD cell is a variation of the LSSD scan cell in the double-latch mode. Unlike the double-latch LSSD scan cell, a single clock controls the enable pins of both the master and slave latches of the scan-enabled LSSD cell.

To recognize the scan-cell type, the signal_type attribute is used. If all the values of the signal_type attribute, namely, test_scan_clock_a, test_scan_clock_b, and test_scan_enable are present on the pins of a test cell, the corresponding scan cell is recognized as a scan-enabled LSSD.
Functional Model of the Scan-Enabled LSSD Cell

To model the cell shown in Figure 6-7, define the signal_type attribute on the pins of the test_cell group, such as the pins, CLK and SELECT. Example 6-6 shows the syntax to define the signal_type attribute on the CLK pin. When you set the signal_type attribute on the clock pin, CLK, to test_scan_clock_b, it indicates that the CLK input also enables the slave-latch in addition to the master-latch of the scan-enabled LSSD cell. In Figure 6-7, the clock pin, CLK, controls both enable pins, C1 and C.

Example 6-6 The signal_type attribute on the Scan-Enabled LSSD Cell CLK pin

cell(cell_name) {
  ...
  test_cell() {
    ...
    pin (pin_name) {
      direction : input;
      signal_type : "test_scan_clock_b";
      ...
    } /* End pin group */
  } /* End test_cell */
  ...
}

Example 6-7 shows the syntax to define the signal_type attribute on the select pin, SELECT. When you set the signal_type attribute on the select pin, SELECT, to test_scan_enable, the SELECT input is active and enables the scan mode of the LSSD cell. When the SELECT input is inactive, the cell is in the normal mode.
Example 6-7  The signal_type Attribute on the Scan-Enabled LSSD Cell SELECT Pin

```lisp
cell(cell_name) {
  ...
  test_cell() {
    ...
    pin (pin_name) {
      direction : input;
      signal_type : "test_scan_enable";
      ...
    } /* End pin group */
  } /* End test_cell */
  ...
}
```

Timing Model of the Scan-Enabled LSSD Cell

Figure 6-8 shows a timing arc constraint, from the clock pin, CLK, to the select pin, SELECT, of the scan-enabled LSSD cell. In the normal mode, the constraint ensures that the CLK input works correctly for the duration of the CLK pulse. Therefore, the SELECT input must be stable for the setup period before the CLK pulse (tds), when the CLK pulse is active (tdstable), and the hold period after (tdh) the CLK pulse.

Example 6-8 shows the syntax to model the timing arc, from the clock pin, CLK, to the select pin, SELECT. Timing arcs are modeled by setting the timing_type attribute to nochange values. As the CLK pin is active-low, set the timing_type attribute on the select pin, SELECT, to nochange_low_low.

Figure 6-8  A Timing-Arc Constraint of the Scan-Enabled LSSD Cell
Example 6-8  Scan-Enabled LSSD Timing Model Syntax

```plaintext
cell(cell_name) {
    ...
    pin (SELECT) {
        direction : input;
        timing() {
            timing_type: "nochange_low_low" ;
            related_pin: "CLK" ;
            fall_constraint(constraint) { /* tds */
                ...
            }
            rise_constraint(constraint) { /* tdh */
                ...
            }
        }
    }
    ...
} /* End pin group */
```

Note:
Do not use the `hold_rising (tds)` and `setup_falling (tdh)` values to model, the clock pin, CLK, to the select pin, SELECT, timing arc. These values of the `timing_type` attribute do not cover the stable region of the SELECT input (tdstable).

Scan-Enabled LSSD Cell Model Example

Example 6-9 uses the syntax in Example 6-6, Example 6-7, and Example 6-8 to model the scan-enabled LSSD cell shown in Figure 6-7 on page 6-28.

Example 6-9  Example for the Scan-Enabled LSSD Cell Syntax

```plaintext
Cell(scan_enabled_LSSD) {
    ...
    statetable ("CLK D SELECT A SI", "MQ Q") {
        table :
        L   L/H L   L - : - : H/L -,-\,
        H   - - H   L - : - : N -,-\,
        L   H  L  L - : - : N -,-\,
        L   L  H  H - : - : X -,-\,
        H   L  L/L H - : - : L/H -,-\,
        - - H  L/L : - - : L/H -,-\,
        L   - - - - : - - : - - N,-,\,
        H   - - - - : L/H - : - L/H";
    }
    pin (CLK) {
        direction : input ;
        timing() {
            timing_type: "min_pulse_width" ;
            related_pin: "CLK" ;
            ...
        }
    }
}
pin (D) {
    direction : input;
    timing() {
        timing_type: "hold_rising";
        related_pin: "CLK";
        ...
    }
    timing() {
        timing_type: "setup_rising";
        related_pin: "CLK";
        ...
    }
}

pin (SELECT) {
    direction : input;
    timing() {
        timing_type: "nochange_low_low";
        related_pin: "CLK";
        ...
    }
}

pin (SI) {
    direction : input;
    timing() {
        timing_type: "setup_falling";
        related_pin: "A";
        ...
    }
    timing() {
        timing_type: "hold_falling";
        related_pin: "A";
        ...
    }
}

pin (MQ) {
    direction : "internal";
    internal_node : "MQ";
}

pin (Q) {
    direction : output;
    internal_node : "Q";
    timing() {
        timing_type: "rising_edge";
        related_pin: "CLK";
        ...
    }
}

...

test_cell () {
    pin (CLK) {
        direction : input;
        signal_type : "test_scan_clock_b";
    }
}
pin (D) {
    direction : input;
}

pin (A) {
    direction : input;
    signal_type : "test_scan_clock_a";
}

pin ("SELECT") {
    direction : input;
    signal_type : "test_scan_enable";
}

pin (SI) {
    direction : input;
    signal_type : "test_scan_in";
}

pin (Q) {
    direction : output;
    function : "IQ";
    signal_type : "test_scan_out";
}

ff (IQ,IQN) {
    next_state : "D";
    clocked_on : "CLK";
}

...

Scan-Enabled LSSD Cell With Asynchronous Inputs

Figure 6-9 Scan-Enabled LSSD Cell With Asynchronous Inputs, PRE and CLR
Example 6-10 shows the modeling syntax for the scan-enabled LSSD cell shown in Figure 6-9.

Example 6-10  Modeling Syntax for Scan-Enabled LSSD Cell With Preset and Clear Inputs

cell(LSSD_with_clear_preset) {
  
  statetable (" CLK  D  SELECT  A  SI  CLR  PRE", "MQ  Q") {
    table : "
    L L/H L L - L L : - - : H/L -,-,
    H - - L - L L : - - : N -,-,
    L L - H L L L : - - : N -,-,
    - - - H L - L L L : H L L,
    H L - H L/H L L : H L L,
    - - - - - - H L : - - - L L,
    H - L H L/H L L : - - : L H -,-,
    - - H - H L/H L L : - - : L H -,-,
    L L - L L - H L : - - : X -,-,
    H - - L L L/H L L : - - : L/H -,-,
    - - H - H L/H L L : - - : L/H -,-,
    L L - L L : - - : N -,-,
    H - - L H L/H L L : - - : H L -,-,
    - - L - - H L : - - : H L -,-,
    L L - L L : - - : L L",
  }

  test_cell () {
    
    pin (CLK) {
      direction : input;
      signal_type : "test_scan_clock_b";
    }
    pin (D) {
      direction : input;
    }
    pin (CLR) {
      direction : input;
    }
    pin (PRE) {
      direction : input;
    }
    pin (A) {
      direction : input;
      signal_type : "test_scan_clock_a";
    }
    pin ("SELECT") {
      direction : input;
      signal_type : "test_scan_enable";
    }
    pin (SI) {
      direction : input;
      signal_type : "test_scan_in";
    }
    pin (Q) {
      direction : output;
      function : "IQ";
      signal_type : "test_scan_out";
    }
  }
}

ff (IQ, IQN){
next_state : "D" ;
clocked_on : "CLK" ;
clear : "CLR" ;
preset : "PRE" ;
clear_preset_var1 : L ;
clear_preset_var2 : L ;

...
```
test_cell() {
  bus(D) {
    bus_type : bus4;
    direction : input; }
  pin(CLK) {
    direction : input;
    signal_type : test_scan_clock_b; }
  pin(SI) {
    direction : input;
    signal_type : test_scan_in; }
  pin(A) {
    direction : input;
    signal_type : test_scan_clock_a; }
  pin(SELECT) {
    direction : input;
    signal_type : test_scan_enable; }
  ff_bank(IQ,IQN,4) {
    next_state : "D";
    clocked_on : "CLK";
  }
  bus(Q) {
    direction : output;
    bus_type : bus4;
    function : "IQ"; }
  pin(SO) {
    direction : output;
    signal_type : "test_scan_out";}
```
Clocked-Scan Test Cell

Example 6-12 shows the model of a level-sensitive latch with separate scan clocking. This example shows the scan cell used in clocked-scan implementation.

**Example 6-12 Clocked-Scan Test Cell**

cell(SC_DLATCH) {
  area : 12;
  pin(D) {
    direction : input;
    capacitance : 1;
    timing() {
      timing_type : setup_falling;
      related_pin : "G";
    }
    timing() {
      timing_type : hold_falling;
      related_pin : "G";
    }
  }
  pin(SI) {
    direction : input;
    capacitance : 1;
    prefer_tied : "0";
    timing() {
      timing_type : setup_rising;
      related_pin : "ScanClock";
    }
    timing() {
      timing_type : hold_rising;
      related_pin : "ScanClock";
    }
  }
  pin(G,ScanClock) {
    direction : input;
    capacitance : 1;
  }
  statetable( "D SI G ScanClock", " Q QN" ) {
    table : "L/H H L : - - : L/H H/L,\ 
            L/H L R : - - : L/H H/L,\ 
            - L L ~R : - - : N N";
  }
  pin(Q) {
    direction : output;
    internal_node : "Q";
    timing() {
  }
timing_type : rising_edge;
        related_pin : "G";
    
    timing() {
        related_pin : "D";
    }

timing() {
    timing_type : rising_edge;
    related_pin : "ScanClock";
}

pin(QN) {
    direction : output;
    internal_node : "QN";
    timing() {
        timing_type : rising_edge;
        related_pin : "G";
    }

timing() {
    related_pin : "D";
    }

timing() {
    timing_type : rising_edge;
    related_pin : "ScanClock";
    timing() {
        timing_type : rising_edge;
        related_pin : "ScanClock";
    }
}

test_cell() {
    pin(D,G) {
        direction : input;
    }

    pin(SI) {
        direction : input;
        signal_type : "test_scan_in";
    }

    pin(ScanClock) {
        direction : input;
        signal_type : "test_scan_clock";
    }

    pin(SE) {
        direction : input;
        signal_type : "test_scan_enable";
    }

    latch ("IQ","IQN") {
        data_in : "D";
        enable : "G";
    }

    pin(Q) {
        direction : output;
        function : "IQ";
        signal_type : "test_scan_out";
Scan D Flip-Flop With Auxiliary Clock

Example 6-13  Scan D Flip-Flop With an Input and an Auxiliary Clock

cell(AUX_DFF1) {
    area : 12;
    pin(D) {
        direction : input;
        capacitance : 1;
        timing() {
            timing_type : setup_rising;
            related_pin : "CK";
        }
        timing() {
            timing_type : hold_rising;
            related_pin : "CK";
        }
        timing() {
            timing_type : setup_rising;
            related_pin : "IH";
        }
        timing() {
            timing_type : hold_rising;
            related_pin : "IH";
        }
    }
    pin(CK,IH,A,B) {
        direction : input;
        capacitance : 1;
    }
    pin(SI) {
        direction : input;
        capacitance : 1;
        prefer_tied : "0";
        timing() {
            timing_type : setup_falling;
            related_pin : "A";
        }
        timing() {
            timing_type : hold_falling;
            related_pin : "A";
        }
    }
}

pin(Q) {
    direction : output;
    timing() {
        timing_type : rising_edge;
        related_pin : "CK IH";
    }
    timing() {
        timing_type : rising_edge;
        related_pin : "B";
    }
}

pin(QN) {
    direction : output;
    timing() {
        timing_type : rising_edge;
        related_pin : "CK IH";
    }
    timing() {
        timing_type : rising_edge;
        related_pin : "B";
    }
}

statetable( "C TC D A B SI", "IQ1 IQ2" ) {
    table :
    /* C TC D A B   SI           IQ1 IQ2   */
    H   -  -   L  -   -  : -   - : L/H  -   /* D in mast */
    -  H  -   L  -   -  : -   - : L/H  -   /* both active */
    H   -  -   L  -   -  : -   - : L/H  -   /* slave loads */
    -  H  -   L  -   -  : -   - : L/H  -   /* slave loads */
    L  L  L/H L  -   -  : -   - : L/H  -   /* D in mast */
    L  L  -   H  -   -  : -   - : X  -   /* both active */
    H   -  -   L  -   -  : L/H  -   - : L/H  -   /* slave loads */
    -  H  -   L  -   -  : L/H  -   - : L/H  -   /* slave loads */
    L  L  -   -  -   -  : -   - : -   N  /* slave loads */
    -  -  -   H  -   -  : -   - : -   N"; /* slave loads */
}

test_cell(){
    pin(D,CK){
        direction : input
    }
    pin(IH){
        direction : input;
        signal_type : "test_clock";
    }
    pin(SI){
        direction : input;
        signal_type : "test_scan_in";
    }
    pin(A){
        direction : input;
        signal_type : "test_scan_clock_a";
    }
    pin(B){
direction : input;
signal_type : "test_scan_clock_b";
}
ff ("IQ","IQN"){
  next_state : "D";
clocked_on : "CK";
}
pin(Q){
direction : output;
function : "IQ";
signal_type : "test_scan_out";
}
pin(QB){
direction : output;
function : "IQN";
signal_type : "test_scan_out_inverted";
}
Timing Arcs

Timing arcs are divided into two major areas: timing delays (the actual circuit timing) and timing constraints (at the boundaries). This chapter explains the timing concepts and describes the timing group attributes for setting constraints and defining delay.

The following sections describe how to specify timing delays:

- Understanding Timing Arcs
- Modeling Method Alternatives
- Describing Three-State Timing Arcs
- Describing Edge-Sensitive Timing Arcs
- Describing Preset and Clear Timing Arcs
- Describing Clock Insertion Delay
- Describing Intrinsic Delay
- Describing Transition Delay
- Modeling Load Dependency
- Describing Slope Sensitivity
- Describing State-Dependent Delays

The following sections describe how to use timing constraints:
• Setting Setup and Hold Constraints
• Setting Nonsequential Timing Constraints
• Setting Recovery and Removal Timing Constraints
• Setting No-Change Timing Constraints
• Setting Skew Constraints
• Setting Conditional Timing Constraints

For additional information, see these sections:
• Impossible Transitions
• Examples of NLDM Libraries
• Describing a Transparent Latch Clock Model
• Checking Timing and Nonlinear Delay Data
• Driver Waveform Support
• Sensitization Support
• Phase-Locked Loop Support
Understanding Timing Arcs

Timing arcs, along with netlist interconnect information, are the paths followed by the path tracer during path analysis. Each timing arc has a startpoint and an endpoint. The startpoint can be an input, output, or I/O pin. The endpoint is always an output pin or an I/O pin. The only exception is a constraint timing arc, such as a setup or hold constraint between two input pins. All delay information in a library refers to an input-to-output pin pair or an output-to-output pin pair.

Figure 7-1  Timing Arcs (AC and BC) for AND Gate

Combinational Timing Arcs

A combinational timing arc describes the timing characteristics of a combinational element. The timing arc is attached to an output pin, and the related pin is either an input or an output.

A combinational timing arc is of one of the following types:

- combinational
- combinational_rise
- combinational_fall
- three_state_disable
- three_state_disable_rise
- three_state_disable_fall
- three_state_enable
- three_state_enable_rise
- three_state_enable_fall

For information about describing combinational timing types, see “timing Group Attributes” on page 7-23.
Sequential Timing Arcs

Sequential timing arcs describe the timing characteristics of sequential elements. In descriptions of the relationship between a clock transition and data output (input to output), the timing arc is considered a delay arc. In descriptions of the relationship between a clock transition and data input (input to input), the timing arc is considered a constraint arc.

A sequential timing arc is of one of the following types:

- Edge-sensitive (rising_edge or falling_edge)
- Setup or clear
- Setup or hold (setup_rising, setup_falling, hold_rising, or hold_falling)
- Nonsequential setup or hold (non_seq_setup_rising, non_seq_setup_falling, non_seq_hold_rising, non_seq_hold_falling)
- Recovery or removal (recovery_rising, recovery_falling, removal_rising, or removal_falling)
- No change (nochange_high_high, nochange_high_low, nochange_low_high, nochange_low_low)

For information about describing sequential timing types, see “timing Group Attributes” on page 7-23.

Modeling Method Alternatives

Timing information for combinational cells such as the one in Figure 7-2 can be modeled in one of two ways, as Figure 7-3 shows.

*Figure 7-2  Two-Inverter Cell*

[Diagram of a two-inverter cell with inputs A, Y, and Z]

In Figure 7-3, Model A defines two timing arcs. The first timing arc starts at primary input pin A and ends at primary output pin Y. The second timing arc starts at primary input pin A and ends at primary output pin Z. This is the simple case.
Figure 7-3  Two Modeling Techniques for Two-Inverter Cell

Model B for this cell also has two arcs but is more accurate than Model A. The first arc starts at pin A and ends at pin Y. This arc is modeled like the AY arc in Model A. The second arc is different; it starts with primary output Y and ends with primary output Z, modeling the effect the load on Y has on the delay on Z. Output-to-output timing arcs can be used in either combinational or sequential cells.

Defining the timing Group

The timing group contains information to model timing arcs and trace paths. The timing group defines the timing arcs through a cell and the relationships between clock and data input signals.

The timing group describes

• Timing relationships between an input pin and an output pin
• Timing relationships between two output pins
• Timing arcs through a noncombinational element
• Setup and hold times on flip-flop or latch input
• Optionally, the names of the timing arcs

The timing group describes setup and hold information when the constraint information refers to an input-to-input pin pair.

The timing group is defined in the pin group. This is the syntax:

```plaintext
library (lib_name) {
  cell (cell_name) {
    pin (pin_name) {
      timing () { ...
     .timing description ...
    }
  }
}
```
Define the timing group in the pin group of the endpoint of the timing arc, as illustrated by pin C in Figure 7-1 on page 7-3.

---

**Naming Timing Arcs Using the timing Group**

Within the timing group, you can identify the name or names of different timing arcs.

A single timing arc can occur between an identified pin and a single related pin identified with the related_pin attribute.

Multiple timing arcs can occur in many ways. The following list shows six possible multiple timing arcs. The following descriptive sections explain how you configure other possible multiple timing arcs:

- Between a single related pin and the identified multiple members of a bundle.
- Between multiple related pins and the identified multiple members of a bundle.
- Between a single related pin and the identified multiple bits on a bus.
- Between multiple related pins and the identified multiple bits of a bus.
- Between the identified multiple bits of a bus and the multiple pins of related bus pins (of a designated width).
- Between the internal pin and all the bits of the endpoint bus group.

The following sections provide descriptions and examples for various timing arcs.

**Timing Arc Between a Single Pin and a Single Related Pin**

Identify the timing arc that occurs between a single pin and a single related pin by entering a name in the timing group, as shown in the following example:

**Example**

```lisp
(cell (my_inverter) {
  ...
  pin (A) {
    direction : input;
    capacitance : 1;
  }
  pin (B) {
    direction : output
    function : "A'";
    timing (A_B) {

)```
The timing arc is as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A_B</td>
</tr>
</tbody>
</table>

**Timing Arcs Between a Pin and Multiple Related Pins**

This section describes how to identify the timing arcs when a timing group is within a pin group and the timing arc has more than a single related pin.

**Example**

You identify the multiple timing arcs on a name list entered with the timing group as shown in the following example.

```plaintext
cell (my_and) {
    ...
    pin (A) {
        direction : input;
        capacitance : 1;
    }
    pin (B) {
        direction : input;
        capacitance : 2;
    }
    pin (C) {
        direction : output
        function : "A B"
        timing (A_C, B_C) {
            related_pin : "A B";
            ...
        }
    }
    /* end timing() */
    /* end pin B */
    /* end cell */
```
The timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>A_C</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>B_C</td>
</tr>
</tbody>
</table>

**Timing Arcs Between a Bundle and a Single Related Pin**

When the timing group is within a bundle group that has several members with a single related pin, enter the names of the resulting multiple timing arcs in a name list in the timing group.

**Example**

```plaintext
...  
bundle (Q){  
    members (Q0, Q1, Q2, Q3);  
    direction : output;  
    function : "IQ";  
    timing (G_Q0, G_Q1, G_Q2, G_Q3){  
        timing_type : rising_edge;  
        related_pin : "G";  
    }  
}
```
If G is a pin, as opposed to another bundle group, the timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Q0</td>
<td>G_Q0</td>
</tr>
<tr>
<td>G</td>
<td>Q1</td>
<td>G_Q1</td>
</tr>
<tr>
<td>G</td>
<td>Q2</td>
<td>G_Q2</td>
</tr>
<tr>
<td>G</td>
<td>Q3</td>
<td>G_Q3</td>
</tr>
</tbody>
</table>

If G is another bundle of member size 4 and G0, G1, G2, and G3 are members of bundle G, the timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0</td>
<td>Q0</td>
<td>G_Q0</td>
</tr>
<tr>
<td>G1</td>
<td>Q1</td>
<td>G_Q1</td>
</tr>
<tr>
<td>G2</td>
<td>Q2</td>
<td>G_Q2</td>
</tr>
<tr>
<td>G3</td>
<td>Q3</td>
<td>G_Q3</td>
</tr>
</tbody>
</table>

### Timing Arcs Between a Bundle and Multiple Related Pins

When the timing group is within a bundle group that has several members, each having a corresponding related pin, enter the names of the resulting multiple timing arcs as a name list in the timing group.

**Example**

```plaintext
bundle (Q){
  members (Q0, Q1, Q2, Q3);
  direction : output;
  function : "IQ";
  timing (G_Q0, H_Q0, G_Q1, H_Q1, G_Q2, H_Q2, G_Q3, H_Q3){
    timing_type : rising_edge;
    related_pin : "G H";
  }
}
```

If G is a pin, as opposed to another bundle group, the timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
</table>
If G was another bundle of member size 4 and G0, G1, G2, and G3 are members of bundle G, the timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0</td>
<td>Q0</td>
<td>G_Q0</td>
</tr>
<tr>
<td>H</td>
<td>Q0</td>
<td>H_Q0</td>
</tr>
<tr>
<td>G1</td>
<td>Q1</td>
<td>G_Q1</td>
</tr>
<tr>
<td>H</td>
<td>Q1</td>
<td>H_Q1</td>
</tr>
<tr>
<td>G2</td>
<td>Q2</td>
<td>G_Q2</td>
</tr>
<tr>
<td>H</td>
<td>Q2</td>
<td>H_Q2</td>
</tr>
<tr>
<td>G3</td>
<td>Q3</td>
<td>G_Q3</td>
</tr>
<tr>
<td>H</td>
<td>Q3</td>
<td>H_Q3</td>
</tr>
</tbody>
</table>

The same rule applies if H is a size-4 bundle.

**Timing Arcs Between a Bus and a Single Related Pin**

This section describes how to identify the timing arcs created when a timing group is within a bus group that has several bits with the same single related pin. You identify the resulting multiple timing arcs by entering a name list with the timing group.
Example

... 

bus (X){
    /*assuming MSB is X[0] */
    bus_type : bus4;
    direction : output;
    capacitance : 1;
    pin (X[0:3]){ 
        function : "B'";
        timing (B_X0, B_X1, B_X2, B_X3){
            related_pin : "B";
        }
    }
}

If B is a pin, as opposed to another 4-bit bus, the timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>X[0]</td>
<td>B_X0</td>
</tr>
<tr>
<td>B</td>
<td>X[1]</td>
<td>B_X1</td>
</tr>
<tr>
<td>B</td>
<td>X[2]</td>
<td>B_X2</td>
</tr>
<tr>
<td>B</td>
<td>X[3]</td>
<td>B_X3</td>
</tr>
</tbody>
</table>

If B is another 4-bit bus and B[0] is the MSB for bus B, the timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[0]</td>
<td>X[0]</td>
<td>B_X0</td>
</tr>
</tbody>
</table>

Timing Arcs Between a Bus and Multiple Related Pins

This section describes the timing arcs created when a timing group is within a bus group that has several bits, where each bit has its own related pin. You identify the resulting multiple timing arcs by entering a name list with the timing group.

Example

bus (X){
    /*assuming MSB is X[0] */
bus_type : bus4;
direction : output;
capacitance : 1;

pin (X[0:3]){
  function : "B''";
  timing (B_X0, C_X0, B_X1, C_X1, B_X2, C_X2, B_X3, C_X3 ){
    related_pin : "B C";
  }
}
}
If B and C are pins, as opposed to another 4-bit bus, the timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>X[0]</td>
<td>B_X0</td>
</tr>
<tr>
<td>C</td>
<td>X[0]</td>
<td>C_X0</td>
</tr>
<tr>
<td>B</td>
<td>X[1]</td>
<td>B_X1</td>
</tr>
<tr>
<td>C</td>
<td>X[1]</td>
<td>C_X1</td>
</tr>
<tr>
<td>B</td>
<td>X[2]</td>
<td>B_X2</td>
</tr>
<tr>
<td>C</td>
<td>X[2]</td>
<td>C_X2</td>
</tr>
<tr>
<td>B</td>
<td>X[3]</td>
<td>B_X3</td>
</tr>
<tr>
<td>C</td>
<td>X[3]</td>
<td>C_X3</td>
</tr>
</tbody>
</table>

If B is another 4-bit bus and B[0] is the MSB for bus B, the timing arcs are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[0]</td>
<td>X[0]</td>
<td>B_X0</td>
</tr>
<tr>
<td>C</td>
<td>X[0]</td>
<td>C_X0</td>
</tr>
<tr>
<td>C</td>
<td>X[1]</td>
<td>C_X1</td>
</tr>
<tr>
<td>C</td>
<td>X[2]</td>
<td>C_X2</td>
</tr>
<tr>
<td>C</td>
<td>X[3]</td>
<td>C_X3</td>
</tr>
</tbody>
</table>

The same rule applies if C is a 4-bit bus.

**Timing Arcs Between a Bus and a Related Bus Equivalent**

You can generate an arc from each element in a starting bus to each element of an ending bus, such as when you create arcs from each related bus pin defined with the `related_bus_pins` attribute to each endpoint.
Instead of using this approach, you can use the `related_bus_equivalent` attribute to generate a single timing arc for all paths from points in a group through an internal pin (I) to given endpoints. Figure 7-4 compares the setup created with the `related_bus_pins` attribute with a setup created with the `related_bus_equivalent` attribute.

**Figure 7-4  Comparing related_bus_pins Setup With related_bus_equivalent Setup**

This section describes the timing arcs created from all the bits of a related bus equivalent group, which you define with the `related_bus_equivalent` attribute, to an internal pin (I) and all the timing arcs created from the same internal pin to all the bits of the endpoint bus group.

You identify the resulting multiple timing arcs by entering a name list, using the `timing` group.

It is assumed that the first name in the name list is the arc going from the first bit (A[0]) of the related `bus` group to the internal pin (I), the second name in the name list is the arc going from the second bit (A[1]) to the internal pin (I), and so on in order until all the related `bus` group bits are used.

The next name on the name list is of the timing arc going from the internal pin (I) to the first bit (X[0]) in the endpoint `bus` group, the following name in the name list is of the arc going from the internal join pin (I) to the second bit (X[1]) of the `bus` group, and so on in order until all the bits in the `bus` group are used. See the following example.

**Note:**
The widths of bus A and bus X do not need to be identical.

**Example**

```plaintext
bus (X) {
  bus_type : bus4;
```

Chapter 7: Timing Arcs
Defining the timing Group
direction : output;
capacitance : 1;
timing (A0_I, A1_I, A2_I, I_X0, I_X1, I_X2, I_X3, ...) {...
    related_bus_equivalent : A;
}...

The following is a list of the timing arcs and their labels:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0]</td>
<td>I</td>
<td>A0_I</td>
</tr>
<tr>
<td>I</td>
<td>X[0]</td>
<td>I_X0</td>
</tr>
<tr>
<td>I</td>
<td>X[1]</td>
<td>I_X1</td>
</tr>
<tr>
<td>I</td>
<td>X[2]</td>
<td>I_X2</td>
</tr>
<tr>
<td>I</td>
<td>X[3]</td>
<td>I_X3</td>
</tr>
</tbody>
</table>

**Delay Model**

The timing groups are defined by the timing group attributes. The delay model determines the set of delay calculation attributes you specify in a timing group.

The NLDM is characterized by tables that define the timing arcs. To describe delay or constraint arcs with this model,

- Use the library-level `lu_table_template` group to define templates of common information to use in lookup tables.
- Use the templates and the timing groups described in this chapter to create lookup tables.

Lookup tables and their corresponding templates can be one-dimensional, two-dimensional, or three-dimensional. Delay arcs allow a maximum of two dimensions. Device degradation constraint tables allow only one dimension. Load-dependent constraint modeling requires three dimensions.
delay_model Attribute

To specify the delay model, use the `delay_model` attribute in the `library` group.

The `delay_model` attribute must be the first attribute in the library if a `technology` attribute is not present. Otherwise, it follows the `technology` attribute.

Example

```plaintext
library (demo) {
    delay_model : table_lookup ;
}
```

Defining the NLDM Template

Table templates store common table information that multiple lookup tables can use. A table template specifies the table parameters and the breakpoints for each axis. Assign each template a name so that lookup tables can refer to it.

lu_table_template Group

Define your lookup table templates in the library group.

Syntax

```plaintext
lu_table_template(name) {
    variable_1 : value;
    variable_2 : value;
    variable_3 : value;
    index_1 ("float, ..., float");
    index_2 ("float, ..., float");
    index_3 ("float, ..., float");
}
```

Template Variables for Timing Delays

The table template specifying timing delays can have up to three variables (`variable_1`, `variable_2`, and `variable_3`). The variables indicate the parameters used to index into the lookup table along the first, second, and third table axes. The parameters are the input transition time of a constrained pin, the output net length and capacitance, and the output loading of a related pin.

The following is a list of the valid values (divided into sets) that you can assign to a table:

- **Set 1:**
  - `input_net_transition`

- **Set 2:**
  - `total_output_net_capacitance`
  - `output_net_length`
output_net_wire_cap
output_net_pin_cap

- Set 3:
  
  related_out_total_output_net_capacitance
  related_out_output_net_length
  related_out_output_net_wire_cap
  related_out_output_net_pin_cap

The values you can assign to the variables of a table specifying timing delay depend on whether the table is one-, two-, or three-dimensional.

For every table, the value you assign to a variable must be from a set different from the set from which you assign a value to the other variables. For example, if you want a two-dimensional table and you assign variable_1 with the input_net_transition value from set 1, then you must assign variable_2 with one of the values from set 2. Table 7-1 lists the combinations of values you can assign to the different variables for the varying dimensional tables specifying timing delays.

<table>
<thead>
<tr>
<th>Template dimension</th>
<th>Variable_1</th>
<th>Variable_2</th>
<th>Variable_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>set1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>set2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>set1</td>
<td>set2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>set2</td>
<td>set1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>set1</td>
<td>set2</td>
<td>set3</td>
</tr>
<tr>
<td>3</td>
<td>set1</td>
<td>set3</td>
<td>set2</td>
</tr>
<tr>
<td>3</td>
<td>set2</td>
<td>set1</td>
<td>set3</td>
</tr>
<tr>
<td>3</td>
<td>set2</td>
<td>set3</td>
<td>set1</td>
</tr>
<tr>
<td>3</td>
<td>set3</td>
<td>set1</td>
<td>set2</td>
</tr>
<tr>
<td>3</td>
<td>set3</td>
<td>set2</td>
<td>set1</td>
</tr>
</tbody>
</table>
Template Variables for Load-Dependent Constraints

The table template specifying load-dependent constraints can have up to three variables (variable_1, variable_2, and variable_3). The variables indicate the parameters used to index into the lookup table along the first, second, and third table axes. The parameters are the input transition time of a constrained pin, the transition time of a related pin, and the output loading of a related pin.

The following is a list of the valid values (divided into sets) that you can assign to a table.

- **Set 1:**
  - constrained_pin_transition
- **Set 2:**
  - related_pin_transition
- **Set 3:**
  - related_out_total_output_net_capacitance
  - related_out_output_net_length
  - related_out_output_net_wire_cap
  - related_out_output_net_pin_cap

The values you can assign to the variables of a table specifying load-dependent constraints depend on whether the table is one-, two-, or three-dimensional.

For every table, the value you assign to a variable must be from a set different from the set from which you assign a value to the other variables. For example, if you want a two-dimensional table and you assign variable_1 with the constrained_pin_transition value from set 1, then you must assign variable_2 with one of the values from set 2.

Table 7-2 lists the combination of values you can assign to the different variables for the varying dimensional tables specifying load-dependent constraints.

<table>
<thead>
<tr>
<th>Template dimension</th>
<th>Variable_1</th>
<th>Variable_2</th>
<th>Variable_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>set1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>set2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>set1</td>
<td>set2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>set2</td>
<td>set1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>set1</td>
<td>set2</td>
<td>set3</td>
</tr>
</tbody>
</table>
Template Breakpoints

The index statements define the breakpoints for an axis. The breakpoints defined by index_1 correspond to the parameter values indicated by variable_1. The breakpoints defined by index_2 correspond to the parameter values indicated by variable_2. The breakpoints defined by index_3 correspond to the parameter values indicated by variable_3.

The index values are lists of floating-point numbers greater than or equal to 0.0. The values in the list must be in increasing order. The size of each dimension is determined by the number of floating-point numbers in the indexes.

You must define at least one index_1 in the lu_table_template group. For a one-dimensional table, use only variable_1.

Creating Lookup Tables

The rules for specifying lookup tables apply to delay arcs as well as to constraints. "Defining Delay Arcs With Lookup Tables" on page 7-38 shows the groups to use as delay lookup tables. See the sections on the various constraints for the groups to use as constraint lookup tables.

This is the syntax for lookup table groups:

```plaintext
lu_table(name) {  
    index_1 ("float, ..., float");  
    index_2 ("float, ..., float");  
    index_3 ("float, ..., float");  
    values("float, ..., float", ..., "float, ..., float");  
}
```
These rules apply to lookup table groups:

- Each lookup table has an associated name for the lu_table_template it uses. The name of the template must be identical to the name defined in a library lu_table_template group.

- You can overwrite any or all of the indexes in a lookup table template, but the overwrite must occur before the actual definition of values.

- The delay value of the table is stored in a values attribute.
  - Transition table delay values must be 0.0 or greater. Propagation tables and cell tables can contain negative delay values.
  - In a one-dimensional table, represent the delay value as a list of \( n_{index_1} \) floating-point numbers.
  - In a two-dimensional table, represent the delay value as \( n_{index_1} \times n_{index_2} \) floating-point numbers.
  - If a table contains only one value, you can use the predefined scalar table template as the template for that timing arc.

- Each group of floating-point values enclosed in quotation marks represents a row in the table.
  - In a one-dimensional table, the number of floating-point values in the group must equal \( n_{index_1} \).
  - In a two-dimensional table, the number of floating-point values in a group must equal \( n_{index_2} \) and the number of groups must equal \( n_{index_1} \).
  - In a three-dimensional table, the total number of groups is \( n_{index_1} \times n_{index_2} \) and each group contains as many floating-point numbers as \( n_{index_3} \). In a three-dimensional table, the first group represents the value indexed by the \( (1, 1, 1) \) to the \( (1, 1, n_{index_3}) \) points in the index. The first \( n_{index_2} \) groups represent the value indexed by the \( (1, 1, 1) \) to the \( (1, n_{index_2}, n_{index_3}) \) points in the index. The rest of the groups are grouped in the same order.
Defining the Scalable Polynomial Delay Model Template

Polynomial templates store common format information that equations can use.

**poly_template Group**

You use a `poly_template` group to specify the equation variables, the variable ranges, the voltage mapping, and the piecewise data. Assign each `poly_template` group a unique name, so that equations in the `timing` group can refer to it.

**Syntax**

```plaintext
poly_template(name_id) {
  variables(variable_i_enum, ..., variable_n_enum)
  variable_i_range(float, float)
  ...
  variable_n_range(float, float)
  mapping(Voltage_enum, power_rail_id)
  domain(domain_name_id) {
    calc_mode : name_id;
    variables(variable_i_enum,..., variable_n_enum)
    variable_i_range(float, float)
    ...
    variable_n_range(float, float)
    mapping(Voltage_enum, power_rail_id)
  }
}
```

**poly_template Variables**

The `poly_template` group that defines timing delays can have up to \( n \) variables (\( \text{variable}_i, ..., \text{variable}_n \)), which you specify in the `variables` complex attribute. The variables you specify represent the following in the equation:

- The input transition time of a constrained pin
- The output net length and capacitance
- The output loading of a related pin
- The default power supply voltage
- The frequency
- The voltage for multivoltage cells
- The temperature
- User parameters (parameter1...parameter5)
The following list shows the valid values, divided into four sets, that you can assign to variables in an equation:

- **Set 1:**
  - `input_net_transition`
  - `constrained_pin_transition`

- **Set 2:**
  - `total_output_net_capacitance`
  - `output_net_length`
  - `output_net_wire_cap`
  - `output_net_pin_cap`
  - `related_pin_transition`

- **Set 3:**
  - `related_out_total_output_net_capacitance`
  - `related_out_output_net_length`
  - `related_out_output_net_wire_cap`
  - `related_out_output_net_pin_cap`

- **Set 4:**
  - `frequency`
  - `temperature`
  - `voltage`
  - `i`
  - `i`

**delay_model Simple Attribute**

Use the `delay_model` attribute in the `library` group to specify the scalable polynomial delay model.

```plaintext
library (demo) {
  delay_model : polynomial ;
}
```
timing Group Attributes

Table 7-3  Attributes in timing Group for NLDM

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Attribute or group</th>
</tr>
</thead>
<tbody>
<tr>
<td>To specify a default timing arc</td>
<td>default_timing</td>
</tr>
<tr>
<td>To identify a timing arc startpoint</td>
<td>related_pin</td>
</tr>
<tr>
<td></td>
<td>related_bus_pins</td>
</tr>
<tr>
<td>To describe a logical effect of input pin on output pin</td>
<td>timing_sense</td>
</tr>
<tr>
<td>To identify an arc as combinational or sequential</td>
<td>timing_type</td>
</tr>
<tr>
<td>To specify a propagation delay in total cell delay, (Used with transition delay)</td>
<td>rise_propagation</td>
</tr>
<tr>
<td></td>
<td>fall_propagation</td>
</tr>
<tr>
<td>To specify a cell delay independent of transition delay</td>
<td>cell_rise</td>
</tr>
<tr>
<td></td>
<td>cell_fall</td>
</tr>
<tr>
<td>To specify a retain delay within the delay arc</td>
<td>retaining_rise</td>
</tr>
<tr>
<td></td>
<td>retaining_fall</td>
</tr>
<tr>
<td>To specify an output or I/O pin for load-dependency model</td>
<td>related_output_pin</td>
</tr>
<tr>
<td>To specify when a timing arc is active</td>
<td>mode</td>
</tr>
</tbody>
</table>

related_pin Simple Attribute

The related_pin attribute defines the pin or pins that are the startpoint of the timing arc. The primary use of related_pin is for multiple signals in ganged-logic timing. This attribute is a required component of all timing groups.
Example

pin (B) {
    direction : output;
    function : "A’";
    timing () {
        related_pin : "A";
        ...
    }
}

You can use the related_pin attribute statement as a shortcut for defining two identical timing arcs for a cell. For example, for a 2-input NAND gate with identical delays from both input pins to the output pin, you need to define only one timing arc with two related pins, as shown in the following example.

pin (Z) {
    direction : output;
    function : "(A * B)’";
    timing () {
        related_pin : "A B";
        ... timing information ...
    }
}

When you use a bus name in a related_pin attribute statement, the bus members or the range of members is distributed across all members of the parent bus. In the following example, the timing arcs described are from each element in bus A to each element in bus B: A(1) to B(1) and A(2) to B(2).

The width of the bus or range must be the same as the width of the parent bus.

bus (A) {
    bus_type : bus2;
    ...
}

bus (B) {
    bus_type : bus2;
    direction : output;
    function : "A’";
    timing () {
        related_pin : "A";
        ... timing information ...
    }
}

related_bus_pins Simple Attribute

The related_bus_pins attribute defines the pin or pins that are the startpoint of the timing arc. The primary use of related_bus_pins is for module generators.
Example

In this example, the timing arcs described are from each element in bus A to each element in bus B: A(1) to B(1), A(1) to B(2), A(1) to B(3), and so on. The widths of bus A and bus B do not need to be identical.

```markdown
bus (A){
  bus_type : bus2;
  ...
}
bus (B){
  bus_type : bus4;
  direction : output;
  function : "A";
  timing () {
    related_bus_pins : "A";
    ...
    timing information ...
  }
}
```

timing_sense Simple Attribute

The `timing_sense` attribute describes the way an input pin logically affects an output pin.

Example

```markdown
timing () {
  timing_sense : positive_unate;
}
```

A function is `unate` if a rising (or falling) change on a positive unate input variable causes the output function variable to rise (or fall) or not change. A rising (or falling) change on a negative unate input variable causes the output function variable to fall (or rise) or not change. For a nonunate variable, further state information is required to determine the effects of a particular state transition.

It is possible that one path is positive unate while another is negative unate. In this case, the first timing arc gets a `positive_unate` designation and the second arc gets a `negative_unate` designation.

Note:

When `timing_sense` describes the transition edge used to calculate delay for the `three_state_enable` or `three_state_disable` pin, it has a meaning different from its traditional one. If a 1 value on the control pin of a three-state cell causes a Z value on the output pin, `timing_sense` is `positive_unate` for the `three_state_disable` timing arc and `negative_unate` for the `three_state_enable` timing arc. If a 0 value on the control pin of a three-state cell causes a Z value on the output pin, `timing_sense` is `negative_unate` for the `three_state_disable` timing arc and `positive_unate` for the `three_state_enable` timing arc.
If a related_pin is an output pin, you must define timing_sense for that pin.

**timing_type Simple Attribute**

The timing_type attribute distinguishes between combinational and sequential cells, by defining the type of timing arc. If this attribute is not assigned, the cell is considered combinational.

The following sections show the timing_type attribute values for the following types of timing arcs:

- Combinational
- Sequential
- Nonsequential
- No-change

**Values for Combinational Timing Arcs**

The timing type and timing sense define the signal propagation pattern. The default timing type is combinational.

<table>
<thead>
<tr>
<th>Timing type</th>
<th>positive_unate</th>
<th>negative_unate</th>
<th>non_unate</th>
</tr>
</thead>
<tbody>
<tr>
<td>combinational</td>
<td>R-&gt;R,F-&gt;F</td>
<td>R-&gt;F,F-&gt;R</td>
<td>{R,F}-&gt;R</td>
</tr>
<tr>
<td>combinational_rise</td>
<td>R-&gt;R</td>
<td>F-&gt;R</td>
<td>{R,F}-&gt;R</td>
</tr>
<tr>
<td>combinational_fall</td>
<td>F-&gt;F</td>
<td>R-&gt;F</td>
<td>{R,F}-&gt;F</td>
</tr>
<tr>
<td>three_state_disable</td>
<td>R-&gt;(0Z,1Z)</td>
<td>F-&gt;(0Z,1Z)</td>
<td>{R,F}-&gt;(0Z,1Z)</td>
</tr>
<tr>
<td>three_state_enable</td>
<td>R-&gt;(Z0,Z1)</td>
<td>F-&gt;(Z0,Z1)</td>
<td>{R,F}-&gt;(Z0,Z1)</td>
</tr>
<tr>
<td>three_state_disable_rise</td>
<td>R-&gt;0Z</td>
<td>F-&gt;0Z</td>
<td>{R,F}-&gt;0Z</td>
</tr>
<tr>
<td>three_state_disable_fall</td>
<td>R-&gt;1Z</td>
<td>F-&gt;1Z</td>
<td>{R,F}-&gt;1Z</td>
</tr>
<tr>
<td>three_state_enable_rise</td>
<td>R-&gt;Z1</td>
<td>F-&gt;Z1</td>
<td>{R,F}-&gt;Z1</td>
</tr>
<tr>
<td>three_state_enable_fall</td>
<td>R-&gt;Z0</td>
<td>F-&gt;Z0</td>
<td>{R,F}-&gt;Z0</td>
</tr>
</tbody>
</table>
Values for Sequential Timing Arcs

You use sequential timing arcs to model the timing requirements for sequential cells.

**rising_edge**
Identifies a timing arc whose output pin is sensitive to a rising signal at the input pin.

**falling_edge**
Identifies a timing arc whose output pin is sensitive to a falling signal at the input pin.

**preset**
Preset arcs affect only the rise arrival time of the arc's endpoint pin. A preset arc implies that you are asserting a logic 1 on the output pin when the designated related pin is asserted.

**clear**
Clear arcs affect only the fall arrival time of the arc's endpoint pin. A clear arc implies that you are asserting a logic 0 on the output pin when the designated related pin is asserted.

**hold_rising**
Designates the rising edge of the related pin for the hold check.

**hold_falling**
Designates the falling edge of the related pin for the hold check.

**setup_rising**
Designates the rising edge of the related pin for the setup check on clocked elements.

**setup_falling**
Designates the falling edge of the related pin for the setup check on clocked elements.

**recovery_rising**
Uses the rising edge of the related pin for the recovery time check. The clock is rising-edge-triggered.

**recovery_falling**
Uses the falling edge of the related pin for the recovery time check. The clock is falling-edge-triggered.

**skew_rising**
The timing constraint interval is measured from the rising edge of the reference pin (specified in related_pin) to a transition edge of the parent pin in the timing group. The rise_constraint value is the maximum skew time between the reference pin
rising and the parent pin rising. The `fall_constraint` value is the maximum skew time between the reference pin falling and the parent pin falling.

`skew_falling`

The timing constraint interval is measured from the falling edge of the reference pin (specified in `related_pin`) to a transition edge of the parent pin in the `timing` group. The `rise_constraint` value is the maximum skew time between the reference pin falling and the parent pin rising. The `fall_constraint` value is the maximum skew time between the reference pin falling and the parent pin falling.

`removal_rising`

Used when the cell is a low-enable latch or a rising-edge-triggered flip-flop. For active-low asynchronous control signals, define the removal time with the `rise_constraint` group. For active-high asynchronous control signals, define the removal time with the `fall_constraint` group.

`removal_falling`

Used when the cell is a high-enable latch or a falling-edge-triggered flip-flop. For active-low asynchronous control signals, define the removal time with the `rise_constraint` group. For active-high asynchronous control signals, define the removal time with the `fall_constraint` group.

`min_pulse_width`

This value, together with the `minimum_period` value, lets you specify the minimum pulse width for a clock pin. The timing check is performed on the pin itself, so the related pin should be the same. You can also include rise and fall constraints, as with other timing checks.

Besides scalar values, table-based minimum pulse width is supported. For an example, see “A Library With timing_type Statements” (Example 7-11 on page 7-73).

`minimum_period`

This value, together with the `min_pulse_width` value, lets you specify the minimum pulse width for a clock pin. The timing check is performed on the pin itself, so the related pin should be the same. You can also include rise and fall constraints as with other timing checks.

`max_clock_tree_path`

Used in `timing` groups under a clock pin. Defines the maximum clock tree path constraint.

`min_clock_tree_path`

Used in `timing` groups under a clock pin. Defines the minimum clock tree path constraint.
Values for Nonsequential Timing Arcs

In some nonsequential cells, the setup and hold timing constraints are specified on the data pin with a nonclock pin as the related pin. The signal of a pin must be stable for a specified period of time before and after another pin of the same cell range state for the cell to function as expected.

**non_seq_setup_rising**

Defines (with **non_seq_setup_falling**) the timing arcs used for setup checks between pins with nonsequential behavior. The related pin in a timing arc is used for the timing check.

**non_seq_setup_falling**

Defines (with **non_seq_setup_rising**) the timing arcs used for setup checks between pins with nonsequential behavior. The related pin in a timing arc is used for the timing check.

**non_seq_hold_rising**

Defines (with **non_seq_hold_falling**) the timing arcs used for hold checks between pins with nonsequential behavior. The related pin in a timing arc is used for the timing check.

**non_seq_hold_falling**

Defines (with **non_seq_hold_rising**) the timing arcs used for hold checks between pins with nonsequential behavior. The related pin in a timing arc is used for the timing check.

Values for No-Change Timing Arcs

You use no-change timing arcs to model the timing requirement for latch devices with latch-enable signals. The four no-change timing types define the pulse waveforms of both the constrained signal and the related signal in NLDM. The information is used in static timing verification during synthesis.

**nochange_high_high**

Indicates a positive pulse on the constrained pin and a positive pulse on the related pin.

**nochange_high_low**

Indicates a positive pulse on the constrained pin and a negative pulse on the related pin.

**nochange_low_high**

Indicates a negative pulse on the constrained pin and a positive pulse on the related pin.

**nochange_low_low**

Indicates a negative pulse on the constrained pin and a negative pulse on the related pin.
mode Complex Attribute

You define the `mode` attribute within a `timing` group. A `mode` attribute pertains to an individual timing arc. The timing arc is active when `mode` is instantiated with a name and a value. You can specify multiple instances of the mode attribute, but only one instance for each timing arc.

Syntax

```
mode (mode_name, mode_value);
```

Example

```
timing() {
  mode(rw, read);
}
```

Example 7-1  A mode Instance Description

```
pin(my_outpin) {
  direction : output;
  timing() {
    related_pin : b;
    timing_sense : non_unate;
    mode(rw, read);
    cell_rise(delay3x3) {
      values("1.1, 1.2, 1.3", "2.0, 3.0, 4.0", "2.5, 3.5, 4.5");
    }
    rise_transition(delay3x3) {
      values("1.0, 1.1, 1.2", "1.5, 1.8, 2.0", "2.5, 3.0, 3.5");
    }
    cell_fall(delay3x3) {
      values("1.1, 1.2, 1.3", "2.0, 3.0, 4.0", "2.5, 3.5, 4.5");
    }
    fall_transition(delay3x3) {
      values("1.0, 1.1, 1.2", "1.5, 1.8, 2.0", "2.5, 3.0, 3.5");
    }
  }
}
```

Example 7-2  Multiple mode Descriptions

```
library (MODE EXAMPLE) {
  delay_model : "table_lookup";
  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1mA";
  pulling_resistance_unit : "1kohm";
  leakage_power_unit : "1nW";
  capacitive_load_unit : (1, pf);
  nom_process : 1.0;
  nom_voltage : 1.0;
  nom_temperature : 125.0;
  slew_lower_threshold_pct_rise : 10;
  slew_upper_threshold_pct_rise : 90;
  input_threshold_pct_fall : 50;
}
output_threshold_pct_fall : 50 ;
input_threshold_pct_rise : 50 ;
output_threshold_pct_rise : 50 ;
slew_lower_threshold_pct_fall : 10 ;
slew_upper_threshold_pct_fall : 90 ;
slew_derate_from_library : 1.0 ;
cell (mode_example) {
  mode_definition(RAM_MODE) {
    mode_value(MODE_1) {
    }
    mode_value(MODE_2) {
    }
    mode_value(MODE_3) {
    }
    mode_value(MODE_4) {
    }
  }
  interface_timing : true;
dont_use : true;
dont_touch : true;
pin(Q) {
  direction : output;
  max_capacitance : 2.0;
  three_state : "!OE";
  timing() {
    related_pin : "CK";
    timing_sense : non_unate
    timing_type : rising_edge
    mode(RAM_MODE,"MODE_1 MODE_2");
    cell_rise(scalar) {
      values( " 0.0 ");
    }
    cell_fall(scalar) {
      values( " 0.0 ");
    }
    rise_transition(scalar) {
      values( " 0.0 ");
    }
    fall_transition(scalar) {
      values( " 0.0 ");
    }
  }
  timing() {
    related_pin : "OE";
    timing_sense : positive_unate
    timing_type : three_state_enable
    mode(RAM_MODE, " MODE_2 MODE_3");
    cell_rise(scalar) {
      values( " 0.0 ");
    }
    cell_fall(scalar) {
      values( " 0.0 ");
    }
  }
}
rise_transition(scalar) {
    values( "0.0");
}
fall_transition(scalar) {
    values( "0.0");
}

timing() {
    related_pin     : "OE";
    timing_sense    : negative_unate
    timing_type     : three_state_disable
    mode(RAM_MODE, MODE_3);
    cell_rise(scalar) {
        values( "0.0");
    }
    cell_fall(scalar) {
        values( "0.0");
    }
    rise_transition(scalar) {
        values( "0.0");
    }
    fall_transition(scalar) {
        values( "0.0");
    }
}
}

pin(A) {
    direction            : input;
capacitance           : 1.0;
max_transition        : 2.0;
timing() {
    timing_type       : setup_rising;
    related_pin       : "CK";
    mode(RAM_MODE, MODE_2);
    rise_constraint(scalar) {
        values( "0.0");
    }
    fall_constraint(scalar) {
        values( "0.0");
    }
}

timing() {
    timing_type       : hold_rising;
    related_pin       : "CK";
    mode(RAM_MODE, MODE_2);
    rise_constraint(scalar) {
        values( "0.0");
    }
    fall_constraint(scalar) {
        values( "0.0");
    }
}
pin(OE) {
    direction : input;
capacitance : 1.0;
max_transition : 2.0;
}

pin(CS) {
direction : input;
capacitance : 1.0;
max_transition : 2.0;
timing() {
timing_type : setup_rising;
related_pin : "CK";
mode(RAM_MODE, MODE_1);
rise_constraint(scalar) {
    values(" 0.0 ");
}
fall_constraint(scalar) {
    values(" 0.0 ");
}
}
timing() {
timing_type : hold_rising;
related_pin : "CK";
mode(RAM_MODE, MODE_1);
rise_constraint(scalar) {
    values(" 0.0 ");
}
fall_constraint(scalar) {
    values(" 0.0 ");
}
}

pin(CK) {
timing() {
timing_type : "min_pulse_width";
related_pin : "CK";
mode(RAM_MODE, MODE_4);
fall_constraint(scalar) {
    values(" 0.0 ");
}
rise_constraint(scalar) {
    values(" 0.0 ");
}
}
timing() {
timing_type : "minimum_period"
related_pin : "CK";
mode(RAM_MODE, MODE_4);
rise_constraint(scalar) {
    values(" 0.0 ");
}
fall_constraint(scalar) {
    values(" 0.0 ");
}
Describing Three-State Timing Arcs

Three-state arcs describe a three-state output pin in a cell.

Describing Three-State-Disable Timing Arcs

To designate a three-state-disable timing arc when defining a three-state pin,

1. Assign related_pin to the enable pin of the three-state function.
2. Define the 0-to-Z propagation time with the cell_rise or rise_transition statement.
3. Define the 1-to-Z propagation time with the cell_fall or fall_transition statement.
4. Include the timing_type:three_state_disable statement.

Example

timing () {
  related_pin : "OE" ;
  timing_type : three_state_disable ;
  /* 0 to Z modeling */
  cell_rise(scalar) {
    values( " 0.0 ");
  }
  rise_transition(scalar) {
    values( " 0.0 ");
  }
  /* 1 to Z modeling */
  cell_fall(scalar) {
    values( " 0.0 ");
  }
  fall_transition(scalar) {
    values( " 0.0 ");
  }
}
Note:
The `timing_sense` attribute, which describes the transition edge used to calculate delay for a timing arc, has a nontraditional meaning when it is included in a `timing` group that also contains a `three_state_disable` attribute. See “`timing_sense` Simple Attribute” on page 7-25 for more information.

---

**Describing Three-State-Enable Timing Arcs**

To designate a three-state-enable timing arc when defining a three-state pin,

1. Assign `related_pin` to the enable pin of the three-state function.

2. Define the Z-to-1 propagation time with the `cell_rise` or `rise_transition` group.

3. Define the Z-to-0 propagation time with the `cell_fall` or `fall_transition` group.

4. Include the `timing_type : three_state_enable` statement.

**Example**

```plaintext
timing () {
    related_pin : "OE" ;
    timing_type : three_state_enable ;

    /* 0 to Z modeling */
    cell_rise(scalar) { 
        values( " 0.0 ");
    } 
    rise_transition(scalar) { 
        values( " 0.0 ");
    } 
    /* 1 to Z modeling */
    cell_fall(scalar) { 
        values( " 0.0 ");
    } 
    fall_transition(scalar) { 
        values( " 0.0 ");
    }
}
```

**Note:**
The `timing_sense` attribute that describes the transition edge used to calculate delay for a timing arc has a nontraditional meaning when it is included in a `timing` group that also contains a `three_state_enable` attribute. See “`timing_sense` Simple Attribute” on page 7-25 for more information.
Describing Edge-Sensitive Timing Arcs

Edge-sensitive timing arcs, such as the arc from the clock on a flip-flop, are identified by the following values of the `timing_type` attribute in the `timing` group.

- **rising_edge**
  Identifies a timing arc whose output pin is sensitive to a rising signal at the input pin.

- **falling_edge**
  Identifies a timing arc whose output pin is sensitive to a falling signal at the input pin.

These arcs are path-traced; the path tracer propagates only the active edge (rise or fall) path values along the timing arc.

See "timing_type Simple Attribute" on page 7-26 for information about the `timing_type` attribute.

The following example shows the timing arc for the QN pin of a JK flip-flop.

**Example**

```plaintext
pin(QN) {
  direction : output ;
  function : "IQN" ;
  timing() {
    related_pin : "CP" ;
    timing_type : rising_edge ;
  }
}
```

The QN pin makes a transition after the clock signal rises.

---

Describing Clock Insertion Delay

Arrival timing paths are the timing paths from an input clock pin to the clock pins that are internal to a cell. The arrival timing paths describe the minimum and the maximum timing constraint for a pin driving an internal clock tree for each input transition, as shown in Figure 7-5.
The max_clock_tree_path and min_clock_tree_path attributes let you define the maximum and minimum clock tree path constraints.

The clock tree path for any one clock can have up to eight values depending on the unateness of the pins and the fastest and slowest paths.

You can use lookup tables to model the cell delays. Lookup tables are indexed only by the input transition time of the clock.

Polynomials can include only the following variables in piecewise domains: input_net_transition, voltage, voltagei, and temperature.

For timing groups whose timing_sense attribute is set to non_unate and whose only variable is input_net_transition, use pairs of lookup tables to model both positive unate and negative unate.

**Figure 7-5 Minimum and Maximum Clock Tree Paths**

**Describing Intrinsic Delay**

The intrinsic delay of an element is the zero-load (fixed) component of the total delay equation. Intrinsic delay attributes have different meanings, depending on whether they are for an input or an output pin.

When describing an output pin, the intrinsic delay attributes define the fixed delay from input to output pin. These values are used to calculate the intrinsic delay of the total delay equation.
When describing an input pin, such as in a setup or hold timing arc, intrinsic attributes define the timing requirements for that pin. Timing constraints are not used in the delay equation. The description of intrinsic delay is inherent in the lookup tables you create.

---

**In the CMOS Piecewise Linear Delay Model**

You describe the intrinsic delay in the CMOS piecewise linear delay model the same way you describe it in the CMOS generic delay model.

---

**In the Scalable Polynomial Delay Model**

The description of intrinsic delay is inherent in the polynomials you create for this delay model. See “Defining the Scalable Polynomial Delay Model Template” on page 7-21 to learn how to create and use templates for scalable polynomials in a library, using the CMOS scalable polynomial nonlinear delay model.

---

**Describing Transition Delay**

The transition delay of an element is the time it takes the driving pin to change state. Transition delay attributes represent the resistance encountered in making logic transitions.

The components of the total delay calculation depend on the timing delay model used. Include the transition delay attributes that apply to the delay model you are using.

Transition time is the time it takes for an output signal to make a transition between the high and low logic states. It is computed by table lookup and interpolation. Transition delay is a function of capacitance at the output pin and input transition time.

---

**Defining Delay Arcs With Lookup Tables**

These timing group attributes provide valid lookup tables for delay arcs:

- `cell_rise`
- `cell_fall`
- `rise_propagation`
- `fall_propagation`
- `retaining_rise`
- `retaining_fall`
• retain_rise_slew
• retain_fall_slew

Note:
For timing groups with timing type clear, only fall groups are valid. For timing groups with timing type preset, only rise groups are valid.

There are two methods for defining delay arcs. Choose the method that best fits your library data characterization.

Method 1
To specify cell delay independently of transition delay, use one of these timing group attributes as your lookup table:
• cell_rise
• cell_fall

Method 2
To specify transition delay as a term in the total cell delay, use one of these timing group attributes as your lookup table:
• rise_propagation
• fall_propagation

retaining_rise and retaining_fall Groups
The retaining delay is the time during which an output port retains its current logical value after a voltage rise or fall at a related input port.

The retaining delay is part of the arc delay (I/O path delay); therefore, its time cannot exceed the arc delay time. Because retaining delay is part of the arc delay, the retaining delay tables are placed within the timing arc.

The value you enter for the retaining_rise attribute determines how long the output pin retains its current value, 0, after the value at the related input port has changed.

The value you enter for the retaining_fall attribute determines how long the output retains its current value, 1, after the value at the related input port has changed.

Figure 7-6 shows retaining delay in regard to changes in a related input port.
Figure 7-6 Retaining Time Delay

Example 7-3 shows how to use the `retaining_rise` and `retaining_fall` attributes.

Example 7-3 Retaining Time Delay

```liberty
library(foo) {
  ...
  lu_table_template (retaining_table_template){
    ...
    variable_1: total_output_net_capacitance;
    variable_2: input_net_transition;
    index_1 ("0.0, 1.5");
    index_2 ("1.0, 2.1");
  }
  ...
  cell (cell_name){
    ...
    pin (A) {
      direction : output;
      ...
      timing(){
        related_pin : "B"
        ...
        retaining_rise (retaining_table_template){
          values ("0.00, 0.23", "0.11, 0.28");
        }
        retaining_fall (retaining_table_template){
          values ("0.01, 0.30", 0.12, 0.18");
        }
      }/*end of pin() */
    ...
  }/*end of cell() */
```
See “Specifying Delay Scaling Attributes” on page 3-10 for information about calculating delay factors.

**retain_rise_slew and retain_fall_slew Groups**

These groups let you specify a slew table for the retain arc that is separate from the table of the parent delay arc. This retain arc represents the time it takes until an output pin starts losing its current logical value after a related input pin is changed. This decaying of the output logic value occurs at a different time than the propagation of the final logical value, and at a different rate.

The retain delay is part of the arc delay (I/O path delay), and therefore its time cannot exceed the arc delay time. Because the retain delay is part of the arc delay, the retain delay tables are placed within the timing arc.

The value you enter for the *retain_rise_slew* attribute determines how long the output pin retains its current value, 0, after the value at the related input port has changed.

The value you enter for the *retain_fall_slew* attribute determines how long the output retains its current value, 1, after the value at the related input port has changed.
Figure 7-7  Timing Diagram of Synchronous RAM

**CL**K

**ADDR**

**DOUT**

**CL**K

**ADDR**

**DOUT**

**CL**K

**ADDR**

**DOUT**

**CL**K

**ADDR**

**DOUT**

**CL**K

**ADDR**

**DOUT**

**CL**K

**ADDR**

**DOUT**
Example

library(library_name) {
  ...
  lu_table_template (retaining_table_template){
    ...
    variable_1: total_output_net_capacitance;
    variable_2: input_net_transition;
    index_1 ("0.0, 1.5");
    index_2 ("1.0, 2.1");
  }
  ...
  cell (cell_name){
    ...
    pin (A) {
      direction : output;
      ...
      timing()
      related_pin : "B"
      ...
      retaining_rise (retaining_table_template){
        values ("0.00, 0.23", "0.11, 0.28");
      }
      retaining_fall (retaining_table_template){
        values ("0.01, 0.30", 0.12, 0.18");
      }
      retain_rise_slew(retaining_time_template){
        values("0.01,0.02");
      }
      retain_fall_slew(retaining_time_template){
        values("0.01,0.02");
      }
    }/*end of pin() */
    ...
  }/*end of cell() */
  ...
}/*end of library() */

Modeling Transition Time Degradation

Current nonlinear delay models are based on the assumption that the transition time at the load pin is the same as the transition time created at the driver pin. In reality, the net acts as a low-pass filter and the transition flattens out as it propagates from the driver of the net to each load, as shown in Figure 7-8. The higher the interconnect load, the greater the flattening effect and the greater the transition delay.
To model the degradation of the transition time as it propagates from an output pin over the net to the next input pin, include these library-level groups in your library:

- `rise_transition_degradation`
- `fall_transition_degradation`

These groups contain the values describing the degradation functions for rise and fall transitions in the form of lookup tables. The lookup tables are indexed by

- Transition time at the net driver
- Connect delay between the driver and a load

These are the supported values for transition degradation (`variable_1` and `variable_2`):

- `output_pin_transition`
- `connect_delay`

You can assign either `connect_delay` or `output_pin_transition` to `variable_1` or `variable_2`, if the index and table values are consistent with the assignment.

The values you use in the table compute the degradation transition according to the following formula:

\[
\text{degraded_transition} = \\
\text{table_lookup}(f(\text{output_pin_transition}, \text{connect_delay}))
\]
The k-factors for process, voltage, and temperature are not supplied for the new tables. The `output_pin_transition` value and the `connect_delay` value are computed at the current, rather than nominal, operating conditions.

In Example 7-4, `trans_deg` is the name of the template for the transition degradation.

**Example 7-4 Using Degradation Tables**

```liberty
library (simple_tlu) {
  delay_model : table_lookup;

  /* define the table templates */
  lu_table_template(prop) {
    variable_1 : input_net_transition ;
    variable_2 : total_output_net_capacitance ;
    index_1("0, 1, 2");
    index_2("0, 1, 2");
  }

  lu_table_template(tran) {
    variable_1 : total_output_net_capacitance ;
    variable_2 : input_net_transition ;
    index_1("0, 1, 2");
    index_2("0, 1, 2");
  }

  lu_table_template(constraint) {
    variable_1 : constrained_pin_transition ;
    index_1("0, 1, 2");
    variable_2 : related_pin_transition ;
    index_2("0, 1, 2");
  }

  lu_table_template(trans_deg) {
    variable_1 : output_pin_transition ;
    index_1("0, 1");
    variable_2 : connect_delay ;
    index_2("0, 1");
  }

  /* the new degradation tables */
  rise_transition_degradation(trans_deg) {
    values("0.0, 0.6", "1.0, 1.6");
  }

  fall_transition_degradation(trans_deg) {
    values("0.0, 0.8", "1.0, 1.8");
  }

  /* other library level defaults */
  default_inout_pin_cap : 1.0;
```

Modeling Load Dependency

“Describing Transition Delay” on page 7-38 describes how to model the transition time dependency of a constrained pin and its related pin on timing constraints. You can further model the effect of unbuffered output on timing constraints by modeling load dependency. Constraints can also be load-dependent.

This is the procedure for modeling load dependency.

1. In the timing group of the output pin, set the timing_type attribute.

2. Use the related_output_pin attribute in the timing group to specify which output pin to use to calculate the load dependency.

3. Create a three-dimensional table template that uses two variables and indexes to model transition time and the third variable and index to model load. The variable values for representing output loading on the related_output_pin are

   related_out_total_output_net_capacitance
   related_out_output_net_length
   related_out_output_net_wire_cap
   related_out_output_net_pin_cap

   See “Defining the NLDM Template” on page 7-16.

4. Create a three-dimensional lookup table, using the table template and the index_3 attribute in the lookup table group. (See “Creating Lookup Tables” on page 7-19.) The following groups are valid lookup tables for output load modeling:

   • rise_constraint
• fall_constraint

See “Setting Setup and Hold Constraints” on page 7-54 for information about these groups.

Example 7-5  Load-Dependent Model in a Library

library(load_dependent) {
  delay_model : table_lookup;
  ...
  lu_table_template(constraint) {
    variable_1 : constrained_pin_transition;
    variable_2 : related_pin_transition;
    variable_3 : related_out_total_output_net_capacitance;
    index_1 ("1", 5, 10");
    index_2 ("1", 5, 10");
    index_3 ("1", 5, 10");
  }
  cell(selector) {
    ...
    pin(d) {
      direction : input;
      capacitance : 4;
      timing() {
        related_pin : "sel";
        related_output_pin : "so";
        timing_type : non_seq_hold_rising;
        rise_constraint(constraint) {
          values("1.5, 2.5, 3.5", "1.6, 2.6, 3.6", "1.7, 2.7, 3.7", \
          "1.8, 2.8, 3.8", "1.9, 2.9, 3.9", "2.0, 3.0, 4.0", \
          "2.1, 3.1, 4.1", "2.2, 3.2, 4.2", "2.3, 3.3, 4.3");
        }
        fall_constraint(constraint) {
          values("1.5, 2.5, 3.5", "1.6, 2.6, 3.6", "1.7, 2.7, 3.7", \
          "1.8, 2.8, 3.8", "1.9, 2.9, 3.9", "2.0, 3.0, 4.0", \
          "2.1, 3.1, 4.1", "2.2, 3.2, 4.2", "2.3, 3.3, 4.3");
        }
    }
  }
  ...
}

In the CMOS Scalable Polynomial Delay Model

This is the procedure for modeling load dependency.

1. In the timing group of the output pin, set the timing_type attribute value.

2. Specify the output pin used to figure the load dependency with the related_output_pin attribute described later.
3. Create a three-dimensional table template that uses two variables to model transition time and a third variable, poly_template, to model load. The variable values for representing output loading on the related_output_pin are

related_out_total_output_net_capacitance
related_out_output_net_length
related_out_output_net_wire_cap
related_out_output_net_pin_cap

See “Defining the Scalable Polynomial Delay Model Template” on page 7-21.

4. Create a three-dimensional lookup table, using the table template and the index_3 attribute in the lookup table group.

Express the delay equation in terms of scalable polynomial delay coefficients, using the variable_3 variable and the variable_3_range attribute in the poly_template group.

- rise_constraint
- fall_constraint

Example 7-6 is an example of a library that includes a load-dependent model.

**Example 7-6 Load-Dependent Model**

library(load_dependent) {
  ...
  technology (cmos);
  delay_model : polynomial;
  ...
  poly_template ( const ) {
    variables (constrained_pin_transition, related_pin_transition, \
      related_out_total_output_net_capacitance);
    variable_1_range (0.0000, 4.0000);
    variable_2_range (0.0000, 4.0000);
    variable_3_range (0.0000, 4.0000);
  }
  ...
  cell(example) {
    ...
    pin(D) {
      direction : input;
      capacitance : 1.00;
      timing() {
        timing_type : setup_rising;
        fall_constraint(const) {
          orders ("2, 1, 1")
          coeffs ("1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0");
        }
        rise_constraint(const){
          orders ("2, 1, 1")
          coeffs ("1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0");
        }
        related_pin : "CP";
        related_output_pin : "Q";
      }
    }
  }
}
Describing Slope Sensitivity

The slope delay of an element is the incremental time delay due to slowing changing input signals. Slope delay is calculated with the transition delay at the previous output pin with a slope sensitivity factor.

A slope sensitivity factor accounts for the time during which the input voltage begins to rise but has not reached the threshold level at which channel conduction begins.

Describing State-Dependent Delays

These timing attributes describe the delay values for specified conditions.

In the `timing` group of a technology library, you need to specify state-dependent delays that correspond to entries in Open Verilog International Standard Delay Format (OVI SDF 2.1) syntax.

To define a state-dependent timing arc, you must specify both the following attributes in each `timing` group:

- `when`
- `sdf_cond`

Note:

The PrimeTime tool uses the `when` attribute values during delay calculation and for SDF generation. The verification simulator tool, VCS, uses the `sdf_cond` attribute values for SDF back annotation. Therefore, you must define the `sdf_cond` and the `when` attributes in the same `timing` group and their values must match. You must define mutually exclusive
conditions for state-dependent timing arcs. *Mutually exclusive* means that no more than one condition (defined in the *when* attribute) can be met at any time. Use the *default_timing* attribute to specify a default timing arc in the case of multiple timing arcs with *when* attributes.

### when Simple Attribute

The *when* attribute is a Boolean expression in the *timing* group that specifies the condition on which a timing arc depends to activate a path. Conditional timing lets you control the output pin of a cell with respect to the various *states* of the input pins.

#### Table 7-4 Valid Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'</code></td>
<td>Invert previous expression</td>
</tr>
<tr>
<td><code>!</code></td>
<td>Invert following expression</td>
</tr>
<tr>
<td><code>^</code></td>
<td>Logical XOR</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Logical AND</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>Logical AND</td>
</tr>
<tr>
<td>space</td>
<td>Logical AND</td>
</tr>
<tr>
<td><code>+</code></td>
<td>Logical OR</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>1</code></td>
<td>Signal tied to logic 1</td>
</tr>
<tr>
<td><code>0</code></td>
<td>Signal tied to logic 0</td>
</tr>
</tbody>
</table>

The order of precedence of the operators is left to right, with inversion performed first, then XOR, then AND, then OR.

#### Example

```
when : "B";
```
Example 7-7 shows how to use the when attribute for an XOR gate. In the description of the XOR cell, pin A sets conditional timing for the output pin "out" when you define the timing arc for related_pin B. In this example, when you set conditional timing for related_pin A with the when : “B” statement, the output pin gets the negative_unate value of A when the condition is B.

There are limitations on the pin types that can be used with different types of cells with the when attribute.

For a combinational cell, these pins are valid with the when attribute:

- Pins in the function attribute for regular combinational timing arcs
- Pins in the three_state attribute for the endpoint of the timing arc

For a sequential cell, valid pins or variables to use with the when attribute are determined by the timing type of the arc.

- For timing types rising_edge and falling_edge: Pins in these attributes are allowed in the when attribute:
  - next_state
  - clocked_on
  - clocked_on_also
  - enable
  - data_in

- For timing type clear
  - If the pin’s function is the first state variable in the flip-flop or latch group, the pin that defines the clear condition in the flip-flop or latch group is allowed in the when construct.
  - If the pin’s function is the second state variable in the flip-flop or latch group, the pin that defines the preset condition in the flip-flop or latch group is allowed in the when construct.
• For timing type `preset`
  ❍ If the pin’s function is the first state variable in the flip-flop or latch group, the pin that defines the preset condition in the flip-flop or latch group is allowed in the `when` construct.
  ❍ If the pin’s function is the second state variable in the flip-flop or latch group, the pin that defines the clear condition in the flip-flop or latch group is allowed in the `when` construct.

See “`timing_type` Simple Attribute” on page 7-26 for more information.

All input pins in a black box cell (a cell without a `function` attribute) are allowed in the `when` attribute.

---

**sdf_cond Simple Attribute**

Defined in the state-dependent timing group, the `sdf_cond` attribute supports Standard Delay Format (SDF) file generation and condition matching during back-annotation.

**Example**

```
sdf_cond : "SE ==1’B1";
```

The `sdf_cond` attribute must be logically equivalent to the `when` attribute for the same timing arc. If the two Boolean expressions are not equivalent, back-annotation is not performed properly.

The `sdf_cond` expressions must be syntax-compliant with SDF 2.1. If the expressions do not meet this standard, errors are generated later in the flow during the generation and reuse of the SDF files.

For simple delay paths, such as IOPATH, you can use the Boolean operators, such as `&&` and `||`, with the `sdf_cond` attribute. However, Verilog timing check statements, including setup, hold, recovery, and removal do not support Boolean operators.

In Example 7-7, the delay between pin A and pin OUT is 1.3 for rising and 1.5 for falling when pin B = 1. There is an additional timing arc between the same two pins that has a rise delay of 1.4 and a fall delay of 1.6 when pin B = 0.

**Example 7-7  2-Input XOR Cell With State-Dependent Timing**

```
cell(XOR) {
  pin(A) {
    direction : input;
    ...
  }
  pin(B) {
    direction : input;
    sdf_cond : "SE ==1’B1";
  }
```
... pin(out) {
  direction : output;
  function : "A ^ B";
  timing() {
    related_pin : "A";
    timing_sense : negative_unate;
    when : "B";
    sdf_cond : " B == 1'B1 ";
    cell_rise(scalar) {
      values( " 1.3 ");
    }
    cell_fall(scalar) {
      values( " 1.5 ");
    }
  }
  timing() {
    related_pin : "A";
    timing_sense : positive_unate;
    when : "!B";
    sdf_cond : " B == 1'B0 ";
    cell_rise(scalar) {
      values( " 1.4 ");
    }
    cell_fall(scalar) {
      values( " 1.6 ");
    }
  }
  timing() { /* default timing arc */
    related_pin : "A";
    timing_sense : non_unate;
    cell_rise(scalar) {
      values( " 1.4 ");
    }
    cell_fall(scalar) {
      values( " 1.6 ");
    }
  }
  timing() {
    related_pin : "B";
    timing_sense : negative_unate;
    when : "A";
    sdf_cond : " A == 1'B1 ";
    cell_rise(scalar) {
      values( " 1.3 ");
    }
    cell_fall(scalar) {
      values( " 1.5 ");
    }
  }
  timing() {
    related_pin : "B";
    timing_sense : positive_unate;
  }
}
Setting Setup and Hold Constraints

Signals arriving at an input pin have ramp times. Therefore, you must ensure that the data signal has stabilized before latching its value by defining setup and hold arcs as timing requirements.

- **Setup constraints** describe the minimum time allowed between the arrival of the data and the transition of the clock signal. During this time, the data signal must remain constant. If the data signal makes a transition during the setup time, an incorrect value might be latched.

- **Hold constraints** describe the minimum time allowed between the transition of the clock signal and the latching of the data. During this time, the data signal must remain constant. If the data signal makes a transition during the hold time, an incorrect value might be latched.

By combining a setup time and a hold time, you can ensure the stability of data that is latched.

The timing checks for flip-flops use the activating edge of the clock, which is the rising edge for the case shown in Figure 7-10.
The timing checks for latches generally use the deactivating edge of the enable signal, which is the falling edge for the case shown in Figure 7-11. However, the method used depends on the vendor.

**Figure 7-10 Setup and Hold Constraints for Rising-Edge-Triggered Flip-Flop**

![Graph showing setup and hold constraints for a rising-edge-triggered flip-flop]

The timing checks for latches generally use the deactivating edge of the enable signal, which is the falling edge for the case shown in Figure 7-11. However, the method used depends on the vendor.

**Figure 7-11 Setup and Hold Constraints for High-Enable Latch**

![Graph showing setup and hold constraints for a high-enable latch]

NLDM supports timing constraints sensitive to clock or data-input transition times. Each constraint is defined by a timing group with two lookup tables:

- `rise_constraint group`
- `fall_constraint group`

**rise_constraint and fall_constraint Groups**

These are constraint tables. The format of the lookup table template and the format of the lookup table are the same as described previously in “Defining the NLDM Template” on page 7-16 and “Creating Lookup Tables” on page 7-19.
These are valid variable values for the timing constraint template:

- `constrained_pin_transition`
  Value for the transition time of the pin that owns the timing group.

- `related_pin_transition`
  Value for the transition time of the `related_pin` defined in the timing group.

For each timing group containing one of the following `timing_type` attribute values, at least one lookup table is required:

- `setup_rising`
- `setup_falling`
- `hold_rising`
- `hold_falling`
- `skew_rising`
- `skew_falling`
- `non_seq_setup_rising`
- `non_seq_setup_falling`
- `non_seq_hold_rising`
- `non_seq_hold_falling`
- `nochange_high_high`
- `nochange_high_low`
- `nochange_low_high`
- `nochange_low_low`

For each timing group with one of the following `timing_type` attribute values, only one lookup table is required:

- `recovery_rising`
- `recovery_falling`
- `removal_rising`
- `removal_falling`

**Example 7-8 Using Lookup Tables to Specify Setup Constraints for Flip-Flop**

``` liberty
library( vendor_b ) {
```
/* 1. Use delay lookup table */
delay_model : table_lookup;

/* 2. Define template of size 3 x 3*/
lu_table_template(constraint_template) {
  variable_1 : constrained_pin_transition;
  variable_2 : related_pin_transition;
  index_1 ("0.0, 0.5, 1.5");
  index_2 ("0.0, 2.0, 4.0");
}

cell(dff) {
  pin(d) {
    direction: input;
    timing() {
      related_pin : "clk";
      timing_type : setup_rising;
      /* Inherit the constraint_template template */
      rise_constraint(constraint_template) {
        /* Specify all the values */
        values ("0.0, 0.13, 0.19", \
            "0.21, 0.23, 0.41", \
            "0.33, 0.37, 0.50");
      }
      fall_constraint(constraint_template) {
        values ("0.0, 0.14, 0.20", \
            "0.22, 0.24, 0.42", \
            "0.34, 0.38, 0.51");
      }
    }
    ...
  }
}

In the Scalable Polynomial Delay Model

Example 7-9 shows how to specify constraint in a scalable polynomial delay model.

Example 7-9   CMOS Scalable Polynomial Delay Model Using Constraint

library(vendor_b) {
  /* Use polynomial delay model */
  delay_model : polynomial;
  /* Define template */
  poly_template ( constraint_template_poly ) {
    variables (constrained_pin_transition, related_pin_transition);
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
  }
  .......
  cell(dff) {
    pin(D) {
      direction : input ;
      timing() {
related_pin : "CP" ;
timing_type : setup_rising ;
    rise_constraint ( constraint_template_poly ) {
        orders("1, 1");
        /* (a0 + a1x1 )(b0 + b1x2) = A00 + A10x1 + A01x2 + A11x1x2 */
        coefs ("0.2487 +0.0520 -0.0268 -0.0053 ");
        variable_1_range (0.01, 3.00);
        variable_2_range (0.01, 3.00);
    }
    fall_constraint ( constraint_template_poly ) {
        orders("1, 2");
        /* (a0 + a1x1 )(b0 + b1x2 + b2x2^2) = */
        /* A00 + A10x1 + A01x2 + A11x1x2 + A02x2^2 + A12x1x2^2 */
        coefs ("0.2732 +0.0668 +0.0216 -0.0002 -0.0003 +0.0000 ");
        variable_1_range (0.01, 3.00);
        variable_2_range (0.01, 3.00);
    }
}

Identifying Interdependent Setup and Hold Constraints

To reduce slack violation, use pairs of interdependence_id attributes to identify interdependent pairs of setup and hold constraint tables. Interdependence data is supported in conditional constraint checking. The interdependence_id increases independently for each condition. Interdependence data can be specified in pin or bus and bundle groups.

Setting Nonsequential Timing Constraints

You can set constraints requiring that the data signal on an input pin remain stable for a specified amount of time before or after another pin in the same cell changes state. These cells are termed nonsequential cells, because the related pin is not a clock signal.

Scaling of nonsequential setup and hold constraints based on the environment use k-factors for sequential setup and hold constraints.

The values you can assign to a timing_type attribute to model nonsequential setup and hold constraints are

non_seq_setup_rising

Designates the rising edge of the related pin for the setup check.

non_seq_setup_falling

Designates the falling edge of the related pin for the setup check.
non_seq_hold_rising

Designates the rising edge of the related pin for the hold check.

non_seq_hold_falling

Designates the falling edge of the related pin for the hold check.

To model nonsequential setup and hold constraints for a cell,

1. Assign a value to the timing_type attribute in a timing group of an input or I/O pin.

2. Specify a related pin with the related_pin attribute in the timing group. The related pin in a timing arc is the pin used for the timing check.

   Use any pin in the same cell, except for output pins, and the constrained pin itself as the related pin.

   You can use both rising and falling edges as the active edge of the related pin for one cell.

**Example**

```plaintext
pin(T) {
    timing() {
        timing_type : non_seq_setup_falling;
        rise_constraint(scalar) {
            values (" 1.5 ");
        }
        fall_constraint(scalar) {
            values (" 1.5 ");
        }
        related_pin : "S";
    }
}
```

**Figure 7-12** shows the waveforms for the nonsequential timing arc described in the preceding example. In this timing arc, the constrained pin is T and its related pin is S. The intrinsic rise value describes setup time C or hold time B. The intrinsic fall value describes setup time A or hold time D.
Setting Recovery and Removal Timing Constraints

Use the recovery and removal timing arcs for asynchronous control pins such as clear and preset.

Recovery Constraints

The recovery timing arc describes the minimum allowable time between the control pin transition to the inactive state and the active edge of the synchronous clock signal (time between the control signal going inactive and the clock edge that latches data in).

The asynchronous control signal must remain constant during this time, or else an incorrect value might appear at the outputs.

Figure 7-13 Recovery Timing Constraint for a Rising-Edge-Triggered Flip-Flop With Active-Low Clear

A = clear to clock recovery time
The values you can assign to a \texttt{timing\_type} attribute to define a recovery time constraint are

\texttt{recovery\_rising}

Uses the rising edge of the related pin for the recovery time check; the clock is rising-edge-triggered.

\texttt{recovery\_falling}

Uses the falling edge of the related pin for the recovery time check; the clock is falling-edge-triggered.

To define a recovery time constraint for an asynchronous control pin,

1. Assign a value to the \texttt{timing\_type} attribute.

   Use \texttt{recovery\_rising} for rising-edge-triggered flip-flops and low-enable latches; use \texttt{recovery\_falling} for negative-edge-triggered flip-flops and high-enable latches.

2. Identify the synchronous clock pin as the \texttt{related\_pin}.

   For active-low control signals, define the recovery time with the \texttt{rise\_constraint} statement.

   For active-high control signals, define the recovery time with the \texttt{fall\_constraint} statement.

\textbf{Example}

This example shows a recovery timing arc for the active-low clear signal in a rising-edge-triggered flip-flop. The \texttt{rise\_constraint} value represents clock recovery time \( A \) in Figure 7-13.

\begin{verbatim}
pin (Clear) {
  direction : input ;
  capacitance : 1 ;
  timing() {
    related\_pin : "Clock" ;
  }
}
\end{verbatim}
The following example shows a recovery timing arc for the active-high preset signal in a low-enable latch. The fall_constraint value represents clock recovery time $A$ in Figure 7-14.

```
pin (Preset) {
  direction : input;
  capacitance : 1;
  timing() {
    related_pin : "Enable";
    timing_type : recovery_rising;
    fall_constraint(scalar) {
      values (" 1.0 ");
    }
  }
}
```

### Removal Constraint

This constraint is also known as the asynchronous control signal hold time. The removal constraint describes the minimum allowable time between the active edge of the clock pin while the asynchronous pin is active and the inactive edge of the same asynchronous control pin (see Figure 7-15).

**Figure 7-15 Timing Diagram for Removal Constraint**

The values you can assign to a `timing_type` attribute to define a removal constraint are `removal_rising`

Use when the cell is a low-enable latch or a rising-edge-triggered flip-flop.
removal_falling

Use when the cell is a high-enable latch or a falling-edge-triggered flip-flop.

To define a removal constraint,
1. Assign a value to the timing_type attribute.
2. Identify the synchronous clock pin as the related_pin.
3. For active-low asynchronous control signals, define the removal time with the rise_constraint attribute.
   For active-high asynchronous control signals, define the removal time with the fall_constraint attribute.

Example
pin ( SET ) {
  ....
  timing() {
    timing_type : removal_rising;
    related_pin : " CK1 ";
    rise_constraint(scalar) {
      values (" 1.0 ");
    }
  }
}

---

Setting No-Change Timing Constraints

A no-change timing check checks a constrained signal against a level-sensitive related signal. The constrained signal must remain stable during an established setup period, for the width of the related pulse, and during an established hold period.

For example, you can use the no-change timing check to model the timing requirements of latch devices with latch enable signals. To ensure correct latch sampling, the latch enable signal must remain stable during the clock pulse and the setup and hold time around the clock pulse.

You can also use the no-change timing check to model the timing requirements of memory devices. To guarantee correct read/write operations, the address or data must remain stable during a read/write enable pulse and the setup and hold margins around the pulse.
The values you can assign to a `timing_type` attribute to define a no-change timing constraint are:

- `nochange_high_high`  
  Specifies a positive pulse on the constrained pin and a positive pulse on the related pin.

- `nochange_high_low`  
  Specifies a positive pulse on the constrained pin and a negative pulse on the related pin.

- `nochange_low_high`  
  Specifies a negative pulse on the constrained pin and a positive pulse on the related pin.

- `nochange_low_low`  
  Specifies a negative pulse on the constrained pin and a negative pulse on the related pin.

*Figure 7-17* shows the waveforms for these constraints.
To model no-change timing constraints,

1. Assign a value to the `timing_type` attribute.

2. Specify a related pin with the `related_pin` attribute in the `timing` group.

   The related pin in the timing arc is the pin used for the timing check.

3. Specify delay attribute values according to the delay model you use, as summarized in the following table.

<table>
<thead>
<tr>
<th>No-change constraint</th>
<th>Setup attribute for nonlinear delay models</th>
<th>Hold attribute for nonlinear delay models</th>
</tr>
</thead>
<tbody>
<tr>
<td>nochange_high_high</td>
<td>rise_constraint</td>
<td>fall_constraint</td>
</tr>
<tr>
<td>nochange_high_low</td>
<td>rise_constraint</td>
<td>fall_constraint</td>
</tr>
<tr>
<td>nochange_low_high</td>
<td>fall_constraint</td>
<td>rise_constraint</td>
</tr>
<tr>
<td>nochange_low_low</td>
<td>fall_constraint</td>
<td>rise_constraint</td>
</tr>
</tbody>
</table>

Note:
With no-change timing constraints, conditional timing constraints have different interpretations than they do with other constraints. See “Setting Conditional Timing Constraints” on page 7-69 for more information.
Specify setup time with the `rise_constraint` attribute and the hold time with the `fall_constraint` attribute.

This is the syntax for the no-change timing check:

```plaintext
timing () {
    timing_type : nochange_high_high | nochange_high_low | \ 
                 nochange_low_high | nochange_low_low ;
    related_pin : related_pinname;
    rise_constraint (template_nameid) { /* constrained signal rising */
        values (float, ... , float) ;
    }
    fall_constraint (template_nameid) { /* constrained signal falling */
        values (float, ... , float) ;
    }
}
```

**Example**

This is an example of a no-change timing check in a technology library:

```plaintext
library (the_lib) {
    delay_model : table_lookup;
    k_process_nochange_rise : 1.0; /* no-change scaling factors */
    k_process_nochange_fall : 1.0;
    k_volt_nochange_rise : 0.0;
    k_volt_nochange_fall : 0.0;
    k_temp_nochange_rise : 0.0;
    k_temp_nochange_fall : 0.0;
    cell (the_cell) {
        pin(EN {
            timing () {
                timing_type : nochange_high_low;
                related_pin : CLK;
                rise_constraint (temp) { /* setup time */
                    values (2.98);
                }
                fall_constraint (temp) { /* hold time */
                    values (0.98);
                }
            }
        }
    }
    ...
    ...
    ...
    ...
}
```
In the CMOS Scalable Polynomial Delay Model

In the CMOS scalable polynomial delay model, specify setup time with \texttt{rise\_constraint} and hold time with \texttt{fall\_constraint}.

This is the syntax for the no-change timing check in the CMOS scalable polynomial delay model:

```plaintext
timing () {
    timing_type : nochange\_high\_high | nochange\_high\_low | \nochange\_low\_high | nochange\_low\_low;
    related_pin : related\_pinname;
    rise\_constraint (poly\_template\_nameid) {
        /* constrained signal rising */
        orders(integer, integer) ;
        coefs(float, ..., float) ;
        variable\_n\_range(float, float) ;
    }
    fall\_constraint (poly\_template\_nameid) {
        /* constrained signal falling */
        orders(integer, integer) ;
        coefs(float, ..., float) ;
        variable\_n\_range(float, float) ;
    }
}
```

Example

This is an example of a technology library no-change timing check using the CMOS scalable polynomial delay model:

```plaintext
library (the\_lib) {
    delay\_model : table\_lookup;
    k\_process\_nochange\_rise : 1.0; /* no-change scaling factors */
    k\_process\_nochange\_fall : 1.0;
    k\_volt\_nochange\_rise : 0.0;
    k\_volt\_nochange\_fall : 0.0;
    k\_temp\_nochange\_rise : 0.0;
    k\_temp\_nochange\_fall : 0.0;
    ...
    cell (the\_cell) {
        pin(EN {
            timing () {
                timing_type : nochange\_high\_low;
                related_pin : CLK;
                rise\_constraint (poly\_template) { /* setup time */
                    orders ("1, 1");
                    coefs ("0.2487 +0.0520 -0.0268 -0.0053 ");
                    variable\_1\_range (0.01, 3.00);
                    variable\_2\_range (0.01, 3.00);
                }
                fall\_constraint (poly\_template) { /* hold time */
                    orders ("1, 1");
                    coefs ("0.2487 +0.0520 -0.0268 -0.0053 ");
                }
            }
        }
    }
}
```
Setting Skew Constraints

The skew constraint defines the maximum separation time allowed between two clock signals.

Figure 7-18  Timing Diagram for Skew Constraint

The values you can assign to a `timing_type` attribute to define a skew constraint are:

- `skew_rising`
  - The timing constraint interval is measured from the rising edge of the reference pin (specified in `related_pin`) to a transition edge of the parent pin of the `timing` group.

- `skew_falling`
  - The timing constraint interval is measured from the falling edge of the reference pin (specified in `related_pin`) to a transition edge of the parent pin of the `timing` group.

To set skew constraint,
1. Assign a value to the `timing_type` attribute.
2. Define only one of these attributes in the `timing` group:
   - `rise_constraint`
   - `fall_constraint`
3. Use the related_pin attribute in the timing group to specify a reference clock pin. Only the following attributes in a skew timing group are used (all others are ignored):

- timing_type
- related_pin
- rise_constraint
- fall_constraint

Example

This example shows how to model constraint CK1 rise to CK2 rise skew:

```plaintext
pin (CK2) {
    ....
    timing() {
        timing_type : skew_rising;
        related_pin : " CK1 ";
        rise_constraint(scalar) { /* constrained signal rising */
            values (" float ");
        }
    }
}
```

Setting Conditional Timing Constraints

A conditional timing constraint describes a check when a specified condition is met. You can specify conditional timing checks in pin, bus, and bundle groups.

Use the following attributes and groups to specify conditional timing checks.

Attributes:

- when
- sdf_cond
- when_start
- sdf_cond_start
- when_end
- sdf_cond_end
- sdf_edges
Groups:

- min_pulse_width
- minimum_period

---

**when and sdf_cond Simple Attributes**

The `when` attribute defines enabling conditions for timing checks such as setup, hold, and recovery.

If you define `when`, you must define `sdf_cond`.

Using the `when` and `sdf_cond` pair is a short way of specifying `when_start`, `sdf_cond_start`, `when_end`, and `sdf_cond_end` when the start condition is identical to the end condition.

“Describing State-Dependent Delays” on page 7-49 describes the `sdf_cond` and `when` attributes in defining state-dependent timing arcs.

---

**when_start Simple Attribute**

In a timing group, `when_start` defines a timing check condition specific to a start event. The expression in this attribute contains any pin, including input, output, I/O, and internal pins. You must use real pin names. Bus and bundle names are not allowed.

**Example**

```plaintext
when_start : "SIG_A"; /*SIG_A must be a declared pin */
```

The `when_start` attribute requires an `sdf_cond_start` attribute in the same timing group. The end condition is considered always true if a timing group contains `when_start` but no `when_end`.

---

**sdf_cond_start Simple Attribute**

In a timing group, `sdf_cond_start` defines a timing check condition specific to a start event in OVI SDF 2.1 syntax.

**Example**

```plaintext
sdf_cond_start : "SIG_A";
```

The `sdf_cond_start` attribute requires a `when_start` attribute in the same timing group.
The end condition is considered always true if a timing group contains sdf_cond_start but no sdf_cond_end.

---

**when_end Simple Attribute**

In a timing group, when_end defines a timing check condition specific to an end event. The expression in this attribute contains any pin, including input, output, I/O, and internal pins. Pins must use real pin names. Bus and bundle names are not allowed.

**Example**

```
when_end : "CD * SD";
```

The when_end attribute requires an sdf_cond_end attribute in the same timing group.

The start condition is considered always true if a timing group contains when_end but no when_start.

---

**sdf_cond_end Simple Attribute**

In a timing group, sdf_cond_end defines a timing check condition specific to an end in OVI SDF 2.1 syntax.

**Example**

```
sdf_cond_end : "SIG_0 == 1'b1";
```

The sdf_cond_end attribute requires a when_end attribute in the same timing group.

The start condition is considered always true if a timing group contains sdf_cond_end but no sdf_cond_start.

---

**sdf_edges Simple Attribute**

The sdf_edges attribute defines edge-specific information for both the start pins and the end pins. Edge types can be noedge, start_edge, end_edge, or both_edges. The default is noedge.

**Example**

```
sdf_edges : both_edges;
```
**min_pulse_width Group**

In a pin, bus, or bundle group, the `min_pulse_width` group models the enabling conditional minimum pulse width check. In the case of a pin, the timing check is performed on the pin itself, so the related pin must be the same.

**min_pulse_width Example**

The following example shows the `min_pulse_width` group with the `constraint_high` and `constraint_low` attributes specified.

```liberty
Example 7-10  min_pulse_width Example
min_pulse_width() {
  constraint_high : 3.0; /* min_pulse_width_high */
  constraint_low : 3.5; /* min_pulse_width_low */
  when : "SE";
  sdf_cond : "SE == 1'B1";
}
```

**constraint_high and constraint_low Simple Attributes**

At least one of these attributes must be defined in the `min_pulse_width` group. The `constraint_high` attribute defines the minimum length of time the pin must remain at logic 1. The `constraint_low` attribute defines the minimum length of time the pin must remain at logic 0.

**when and sdf_cond Simple Attributes**

These attributes define the enabling condition for the timing check. Both attributes are required in the `min_pulse_width` group.

---

**minimum_period Group**

In a pin, bus, or bundle group, the `minimum_period` group models the enabling conditional minimum period check. In the case of a pin, the check is performed on the pin itself, so the related pin must be the same. The attributes in this group are `constraint`, `when`, and `sdf_cond`.

**minimum_period Example**

```liberty
minimum_period() {
  constraint : 9.5; /* min_period */
  when : "SE";
  sdf_cond : "SE == 1’B1";
}
```
constraint Simple Attribute
This required attribute defines the minimum clock period for the pin.

when and sdf_cond Simple Attributes
These attributes define the enabling condition for the timing check. Both attributes are required in the \texttt{minimum\_period} group.

\textbf{min\_pulse\_width and minimum\_period Example}

Example 7-11 shows how to specify a lookup table with the \texttt{timing\_type} attribute and \texttt{min\_pulse\_width} and \texttt{minimum\_period} values. The \texttt{rise\_constraint} group defines the rising pulse width constraint for \texttt{min\_pulse\_width}, and the \texttt{fall\_constraint} group defines the falling pulse width constraint. For \texttt{minimum\_period}, the \texttt{rise\_constraint} group is used to model the period when the pulse is rising and the \texttt{fall\_constraint} group is used to model the period when the pulse is falling. You can specify the \texttt{rise\_constraint} group, the \texttt{fall\_constraint} group, or both groups.

Example 7-11  Library With timing\_type Statements

\begin{verbatim}
library(example) {
    technology (cmos) ;
    delay_model : table_lookup ;

    /* 2-D table template */
    lu_table_template ( mpw ) {
        variable_1 : constrained_pin_transition;
        /* You can replace the constrained_pin_transition value with related_pin_transition, but you cannot specify both values. */
        variable_2 : related_out_total_output_net_capacitance;
        index_1("1, 2, 3");
        index_2("1, 2, 3");
    }

    /* 1-D table template */
    lu_table_template( f_ocap ) {
        variable_1 : total_output_net_capacitance;
        index_1 (" 0.0000, 1.0000");
    }

    cell( test ) {
        area : 200.000000 ;
        dont_use : true ;
        dont_touch : true ;

        pin ( CK ) {
            direction : input;
        }
    }
}
\end{verbatim}
rise_capacitance : 0.00146468;
fall_capacitance : 0.00145175;
capacitance : 0.00146468;
clock : true;

timing (mpw_constraint) {
  related_pin : "CK";
timing_type : min_pulse_width;
  related_output_pin : "Z";

  fall_constraint (mpw) {
    index_1("0.1, 0.2, 0.3");
    index_2("0.1, 0.2");
    values( "0.10 0.11", \
      "0.12 0.13" \
      "0.14 0.15");
  }

  rise_constraint (mpw) {
    index_1("0.1, 0.2, 0.3");
    index_2("0.1, 0.2");
    values( "0.10 0.11", \n      "0.12 0.13" \n      "0.14 0.15");
  }
}

timing (mpw_constraint) {
  related_pin : "CK";
timing_type : minimum_period;
  related_output_pin : "Z";

  fall_constraint (mpw) {
    index_1("0.2, 0.4, 0.6");
    index_2("0.2, 0.4");
    values( "0.20 0.22", \n      "0.24 0.26" \n      "0.28 0.30");
  }

  rise_constraint (mpw) {
    index_1("0.2, 0.4, 0.6");
    index_2("0.2, 0.4");
    values( "0.20 0.22", \n      "0.24 0.26" \n      "0.28 0.30");
  }
}

} /* end of arc */
} /* end of cell */
} /* end of library */
Using Conditional Attributes With No-Change Constraints

As shown in Table 7-5, conditional timing check attributes have different interpretations when you use them with no-change timing constraints. See “Setting No-Change Timing Constraints” on page 7-63 for a description of no-change timing constraint values.

Table 7-5 Conditional Timing Attributes With No-Change Constraints

<table>
<thead>
<tr>
<th>Conditional attributes</th>
<th>With no-change constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>when sdf_cond</td>
<td>Defines both the constrained and related signal-enabling conditions</td>
</tr>
<tr>
<td>when_start sdf_cond_start</td>
<td>Defines the constrained signal-enabling condition</td>
</tr>
<tr>
<td>when_end sdf_cond_end</td>
<td>Defines the related signal-enabling condition</td>
</tr>
<tr>
<td>sdf_edges: start_edge</td>
<td>Adds edge specification to the constrained signal</td>
</tr>
<tr>
<td>sdf_edges: end_edge</td>
<td>Adds edge specification to the related signal</td>
</tr>
<tr>
<td>sdf_edges: both_edges</td>
<td>Specifies edges to both signals</td>
</tr>
<tr>
<td>sdf_edges: noedges</td>
<td>Specifies edges to neither signal</td>
</tr>
</tbody>
</table>

Figure 7-19 Interpretation of Conditional Timing Constraints With No-Change Constraints
Impossible Transitions

This information applies to the table lookup delay model and only to combinational and three-state timing arcs. Certain output transitions cannot result from a single input change when function, three_state, and x_function share input.

In the following table, Y is the function of A, B, and C.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

No isolated signal change on C can cause the 0-to-1 or 1-to-0 transitions on Y. Therefore, there is no combinational arc from C to Y, although the two are functionally related. Further, no isolated change on A causes the 1-to-Z or Z-to-1 transitions on Y.

three_state_enable has no rising value (Z to 1), and three_state_disable has no falling value (1 to Z).

Examples of NLDM Libraries

This section contains examples of libraries using the CMOS nonlinear delay model.

CMOS Piecewise Linear Delay Model

In the CMOS piecewise linear D flip-flop description in Example 7-12, pin D contains setup and hold constraints; pins Q and QN contain preset, clear, and edge-sensitive arcs.
Example 7-12  D Flip-Flop Description (CMOS Piecewise Linear Delay Model)

library (example){
date : "January 14, 2002";
revision : 2000.01;
delay_model : piecewise_cmos;
technology (cmos);
.piece_define("0,15,40");
cell ( DFLOP_CLR_PRE ) {
    area : 11;
    ff ( IQ , IQN ) {
        clocked_on : " CLK ";
        next_state : " D ";
        clear : " CLR' ";
        preset : " PRE' ";
        clear_preset_var1 : L ;
        clear_preset_var2 : L ;
    }
    pin ( D ){
        direction : input;
        capacitance : 1 ;
        timing () {
            related_pin : "CLK" ;
            timing_type : hold_rising;
            intrinsic_rise : 0.12;
            intrinsic_fall : 0.12 ;
        }
        timing (){
            related_pin : "CLK" ;
            timing_type : setup_rising ;
            intrinsic_rise : 2.77 ;
            intrinsic_fall : 2.77 ;
        }
    }
    pin ( CLK ){
        direction : input;
        capacitance : 1 ;
    }
    pin ( PRE ) {
        direction : input ;
        capacitance : 2 ;
    }
    pin ( CLR ){
        direction : input ;
        capacitance : 2 ;
    }
    pin ( Q ) {
        direction : output;
        function : "IQ" ;
        timing () {
            related_pin : "PRE" ;
            timing_type : preset ;
            timing_sense : positive_unate ;
        }
    }
}
intrinsic_rise : 0.65;
rise_delay_intercept (0,0.054); /* piece 0 */
rise_delay_intercept (1,0.0); /* piece 1 */
rise_delay_intercept (2,-0.062); /* piece 2 */
rise_pin_resistance (0,0.25); /* piece 0 */
rise_pin_resistance(1,0.50); /* piece 1 */
rise_pin_resistance (2,1.00); /* piece 2 */
}
timing (){  
  related_pin : "CLR" ;  
  timing_type : clear ;  
  timing_sense : negative_unate ;  
  intrinsic_fall : 1.45 ;  
  fall_delay_intercept (0,1.0); /* piece 0 */  
  fall_delay_intercept (1,0.0); /* piece 1 */  
  fall_delay_intercept (2,-1.0); /* piece 2 */  
  fall_pin_resistance (0,0.25); /* piece 0 */  
  fall_pin_resistance (1,0.50); /* piece 1 */  
  fall_pin_resistance (2,1.00); /* piece 2 */
}
timing (){  
  related_pin : "CLK" ;  
  timing_type : rising_edge;  
  intrinsic_rise : 1.40 ;
  intrinsic_fall : 1.91 ;
  rise_pin_resistance (0,0.25); /* piece 0 */
  rise_pin_resistance (1,0.50); /* piece 1 */
  rise_pin_resistance (2,1.00); /* piece 2 */
  fall_pin_resistance (0,0.15); /* piece 0 */
  fall_pin_resistance (1,0.40); /* piece 1 */
  fall_pin_resistance (2,0.90); /* piece 2 */
}
}

pin ( QN ){
  direction : output ;
  function : "IQN" ;
timing (){  
  related_pin : "PRE" ;
  timing_type : clear ;
  timing_sense : negative_unate ;
  intrinsic_fall : 1.87 ;
  fall_delay_intercept (0,1.0); /* piece 0 */
  fall_delay_intercept (1,0.0); /* piece 1 */
  fall_delay_intercept (2,-1.0); /* piece 2 */
  fall_pin_resistance (0,0.25); /* piece 0 */
  fall_pin_resistance (1,0.50); /* piece 1 */
  fall_pin_resistance (2,1.00); /* piece 2 */
}
timing (){  
  related_pin : "CLR" ;
  timing_type : preset ;
  timing_sense : positive_unate ;
  intrinsic_rise : 0.68 ;
Library With Timing Constraints

In the nonlinear library description in Example 7-13, pin D contains setup and hold constraints; pins Q and QN contain preset, clear, and edge-sensitive arcs.

Example 7-13  D Flip-Flop Description

```Liberty
library {NLDM}{
date : "January 14, 2015";
revision : 2015.01;
delay_model : table_lookup;
technology (cmos);
/* Define template of size 2 x 2*/
lu_table_template(cell_template) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1 ("0.0, 1.5");
  index_2 ("0.0, 4.0");
}
/* Define one-dimensional lu_table of size 4 */
lu_table_template(tran_template) {
  variable_1 : total_output_net_capacitance;
  index_1 ("0.0, 0.5, 1.5, 2.0");
}
cell (DFLOP_CLR_PRE) {
  area : 11;
  ff (IQ , IQN ) {
    clocked_on : "CLK";
    next_state : "D";
}
```
clear : " CLR’ " ;
preset : " PRE’ " ;
clear_preset_var1 : L ;
clear_preset_var2 : L ;
}

pin ( D ){
direction : input;
capacitance : 1 ;
timing () {
    related_pin : "CLK" ;
timing_type : hold_rising;
    rise_constraint(scalar) {
        values("0.12") ;
    }
    fall_constraint(scalar) {
        values("0.29") ;
    }
}
timing (){ 
    related_pin : "CLK" ;
timing_type : setup_rising ;
    rise_constraint(scalar) {
        values("2.93") ;
    }
    fall_constraint(scalar) {
        values("2.14") ;
    }
}
}

pin ( CLK ){
direction : input;
capacitance : 1 ;
}

pin ( PRE ) {
direction : input ;
capacitance : 2 ;
}

pin ( CLR ){
direction : input ;
capacitance : 2 ;
}

pin ( Q ) {
direction : output;
function : "IQ" ;
timing () {
    related_pin : "PRE" ;
timing_type : preset ;
timing_sense : negative_unate ;
cell_rise(cell_template) {
        values("0.00, 0.23", "0.11, 0.28") ;
    }
    rise_transition(tran_template) {

values("0.01, 0.12, 0.15, 0.40");
}

timing (){  
    related_pin : "CLR" ;  
    timing_type : clear ;  
    timing_sense : positive_unate ;  
    cell_fall(cell_template) {  
        values("0.00, 0.24", "0.15, 0.26");  
    }
    fall_transition(tran_template) {  
        values("0.03, 0.15, 0.18, 0.38");  
    }
}

timing (){  
    related_pin : "CLK" ;  
    timing_type : rising_edge;  
    cell_rise(cell_template) {  
        values("0.00, 0.25", "0.11, 0.28");  
    }
    rise_transition(tran_template) {  
        values("0.01, 0.08, 0.15, 0.40");  
    }
    cell_fall(cell_template) {  
        values("0.00, 0.33", "0.11, 0.38");  
    }
    fall_transition(tran_template) {  
        values("0.01, 0.11, 0.18, 0.40");  
    }
}

pin ( QN ){
    direction : output ;
    function : "IQN" ;
    timing (){  
        related_pin : "PRE" ;  
        timing_type : clear ;  
        timing_sense : positive_unate ;  
        cell_fall(cell_template) {  
            values("0.00, 0.23", "0.11, 0.28");  
        }
        fall_transition(tran_template) {  
            values("0.01, 0.12, 0.15, 0.40");  
        }
}

timing (){  
    related_pin : "CLR" ;  
    timing_type : preset ;  
    timing_sense : negative_unate ;  
    cell_rise(cell_template) {  
        values("0.00, 0.23", "0.11, 0.28");  
    }
    rise_transition(tran_template) {
values("0.01, 0.12, 0.15, 0.40");
}
}
timing (){
    related_pin : "CLK";
    timing_type : rising_edge;
    cell_rise(cell_template) {
        values("0.00, 0.25", "0.11, 0.28");
    }
    rise_transition(tran_template) {
        values("0.01, 0.08, 0.15, 0.40");
    }
    cell_fall(cell_template) {
        values("0.00, 0.33", "0.11, 0.38");
    }
    fall_transition(tran_template) {
        values("0.01, 0.11, 0.18, 0.40");
    }
}
}

---

**CMOS Scalable Polynomial Delay Model**

In the scalable polynomial delay model in Example 7-14, pin D contains setup and hold constraints; pins Q and QN contain preset, clear, and edge-sensitive arcs.

**Example 7-14  D Flip-Flop Description (CMOS Scalable Polynomial Delay Model)**

library (SPDM) {
    technology (cmos);
    date : "September 19, 2002" ;
    revision : 2002.01 ;
    delay_model : polynomial ;
    /* Define template of 2D polynomial */
    poly_template(cell_template) {
        variables(input_net_transition, total_output_net_capacitance) ;
        variable_1_range(0.0, 1.5) ;
        variable_2_range(0.0, 4.0) ;
    }
    /* Define template of 1D polynomial */
    poly_template(tran_template) {
        variables(total_output_net_capacitance);
        variable_1_range(0.0, 2.0) ;
    }
    cell(DFLOP_CLR_PRE) {
        area : 11;
        ff(IQ, IQN) {
            clocked_on : "CLK" ;
            next_state : "D" ;
            clear : "CLR" ;
            preset : "PRE" ;
            clear_preset_var1 : L ;
        }
    }
clear_preset_var2 : L ;
}

pin(D) {
  direction : input;
  capacitance : 1;
  timing () {
    related_pin : "CLK";
    timing_type : hold_rising;
    rise_constraint(scalar) {
      values("0.12") ;
    }
    fall_constraint(scalar) {
      values("0.29") ;
    }
  } /* end timing */
  timing () {
    related_pin : "CLK";
    timing_type : setup_rising;
    rise_constraint(scalar) {
      values("2.93") ;
    }
    fall_constraint(scalar) {
      values("2.14") ;
    }
  } /* end timing */
} /* end pin D */

pin(CLK) {
  direction : input;
  capacitance : 1;
}

pin(PRE) {
  direction : input;
  capacitance : 2;
}

pin(CLR) {
  direction : input;
  capacitance : 2;
}

pin(Q) {
  direction : output;
  function : "IQ";
  timing () {
    related_pin : "PRE";
    timing_type : preset;
    timing_sense : negative_unate;
    cell_rise(cell_template) {
      orders("1, 1") ;
      coefs("0.1632, 3.0688, 0.0013, 0.0320") ;
      variable_1_range (0.01, 3.00);
      variable_2_range (0.01, 3.00);
    }
    rise_transition(tran_template) {
      orders("1") ;
      coefs("0.2191, 1.7580") ;
      variable_1_range (0.01, 3.00);
      variable_2_range (0.01, 3.00);
    }
  } /* end timing */
timing () {
    related_pin : "CLR"
    timing_type : clear;
    timing_sense : positive_unate;
    cell_fall(cell_template) {
        orders("1, 1")
        coefs("0.0542, 6.3294, 0.0214, -0.0310")
        variable_1_range (0.01, 3.00);
        variable_2_range (0.01, 3.00);
    }
    fall_transition(tran_template) {
        orders("1")
        coefs("0.0652, 2.9232")
        variable_1_range (0.01, 3.00);
        variable_2_range (0.01, 3.00);
    }
} /* end timing */

timing () {
    related_pin : "CLR"
    timing_type : rising_edge;
    cell_rise(cell_template) {
        orders("1, 1")
        coefs("0.1687, 3.0627, 0.0194, 0.0155")
        variable_1_range (0.01, 3.00);
        variable_2_range (0.01, 3.00);
    }
    rise_transition(tran_template) {
        orders("1")
        coefs("0.2130, 1.7576")
    }
}

pin(Q) {
    direction : output;
    function : "IQN"
    timing () {
        related_pin : "PRE"
        timing_type : clear;
        timing_sense : positive_unate;
        cell_fall(cell_template) {
            orders("1, 1")
            coefs("0.0539, 6.3360, 0.0194, -0.0289")
            variable_1_range (0.01, 3.00);
            variable_2_range (0.01, 3.00);
        }
        fall_transition(tran_template) {
            orders("1")
            coefs("0.0647, 2.9220")
            variable_1_range (0.01, 3.00);
            variable_2_range (0.01, 3.00);
        }
    } /* end timing */
} /* end pin Q */

pin(QN) {
    direction : output;
    function : "IQN"
    timing () {
        related_pin : "PRE"
        timing_type : clear;
        timing_sense : positive_unate;
        cell_fall(cell_template) {
            orders("1, 1")
            coefs("0.1605, 3.0639, 0.0325, 0.0104")
            variable_1_range (0.01, 3.00);
            variable_2_range (0.01, 3.00);
        }
        fall_transition(tran_template) {
orders ("1");
  coefs("0.1955, 1.7535");
  variable_1_range (0.01, 3.00);
  variable_2_range (0.01, 3.00);
}
} /* end timing */
timing () {
  related_pin : "CLR" ;
  timing_type : preset ;
  timing_sense : negative_unate ;
  cell_rise(cell_template) {
    orders ("1, 1");
    coefs("0.0540, 6.3849, 0.0211, -0.0720" );
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
  }
  rise_transition(tran_template) {
    orders ("1");
    coefs("0.0612, 2.9541" );
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
  }
} /* end timing */
timing () {
  related_pin : "CLK" ;
  timing_type : rising_edge ;
  cell_rise(cell_template) {
    orders ("1, 1");
    coefs ("0.2407, 3.1568, 0.0129, 0.0143"");
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
  }
  rise_transition(tran_template) {
    orders ("1");
    coefs ("0.3355, 1.7578" );
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
  }
  cell_fall(cell_template) {
    orders("1, 1");
    coefs("0.0742, 6.3452, 0.0260, -0.0938" );
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
  }
  fall_transition(tran_template) {
    orders ("1");
    coefs("0.0597, 2.9997" );
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
  }
} /* end timing */
} /* end pin QN */
} /* end library */
Library With Clock Insertion Delay

library( vendor_a ) {
    delay_model :table_lookup;
    /* 1. Define library-level one-dimensional lu_table of size 4 */
    lu_table_template(lu_template) {
        variable_1 :input_net_transition;
        index_1 ("0.0, 0.5, 1.5, 2.0");
    }
    /* 2. Define a cell and pins within it which has clock tree path*/
    cell (general) {
        ...
        pin(clk) {
            direction:input;
            timing() {
                timing_type :max_clock_tree_path;
                timing_sense :positive_unate;
                cell_rise(lu_template) {
                    values ("0.1, 0.15, 0.20, 0.29");
                }
                cell_fall(lu_template) {
                    values ("0.2, 0.25, 0.30, 0.39");
                }
            }
            timing() {
                timing_type :min_clock_tree_path;
                timing_sense :positive_unate;
                cell_rise(lu_template) {
                    values ("0.2, 0.35, 0.40, 0.59");
                }
                cell_fall(lu_template) {
                    values ("0.3, 0.45, 0.50, 0.69");
                }
            }
        }
        ...
    }
}

Describing a Transparent Latch Clock Model

The tlatch group lets you specify a functional latch when the latch arcs are absent. You use the tlatch group at the data pin level to specify the relationship between the data pin and the enable pin on a latch.
Syntax

```plaintext
library (name_string) {  
  cell (name_string) {  
    ...  
    timing_model_type : value_enum ;  
    ...  
    pin (data_pin_name_string) {  
      tlatch (enable_pin_name_string) {  
        edge_type : value_enum ;  
        tdisable : value_Boolean ;  
      }  
    }  
  }  
}
```

The `tlatch` name specifies the enable pin that defines the latch clock pin. You define the `tlatch` group in a `pin` group, but it is only effective if you also define the `timing_model_type` attribute in the cell that the pin belongs to. The `timing_model_type` attribute can have the following values: abstracted, extracted, and quick timing model.

A `tlatch` group is optional. You can define one or more `tlatch` groups for a pin, but you must not define more than two `tlatch` groups between the same pair of data and enable pins, one rising and one falling. Also, the data pin and the enable pin must be different.

Pins in the `tlatch` group can be input or inout pins or internal pins. When a `tlatch` group is not present, a timing analysis tool infers the latch clock pin based on the presence of:

- A related pin statement in a timing group with either a rising_edge or falling_edge timing_type value within a latch output pin
- A related pin statement in a timing group with a setup, hold_rising, or falling timing_type value within a latch input pin
The `edge_type` attribute defines whether the latch is positive (high) transparent or negative (low) transparent. The rising and falling `edge_type` values specify the opening edge, and therefore the transparent window of the latch, and completely define the latch to be level-high transparent or level-low transparent.

When a `tlatch` group is not present, a timing analysis tool infers transparency on an output pin based on the timing arc attribute and the presence of a latch functional construct on that pin.

The rising and falling `edge_type` attribute values explicitly define the transparency windows

- When the `rising_edge` and `falling_edge` `timing_type` values are missing
- When the `rising_edge` and `falling_edge` `timing_type` values are different from the latch transparency

The `tdisable` attribute disables transparency in a latch. During path propagation, timing analysis tools disable and ignore all data pin output pin arcs that reference a `tlatch` group whose `tdisable` attribute is set to true on an edge triggered flip-flop.

Example

```plaintext
pin (D) {
  tlatch (CP) {
    edge_type : rising ;
    tdisable : true ;
  }
}

pin (Q) {
  direction : output ;
  timing () {
    timing_)sense : positive_unate ;
    related_pin : D ;
    cell_rise ... 
  }
  timing () {
    /* optional arc that can differ from edge_type */
    timing_type : falling_edge ;
    related_pin : CP ;
    cell_fall ... 
  }
}
```

**Driver Waveform Support**

In cell characterization, the shape of the waveform driving the characterized circuit can have a significant impact on the final results. Typically, the waveform is generated by a simple piecewise linear (PWL) waveform or an active-driver cell (a buffer or inverter).
Liberty supports driver waveform syntax, which specifies the type of waveform that is applied to library cells during characterization. The driver waveform syntax helps facilitate the characterization process for existing libraries and correlation checking. The driver waveform requirements can vary.

Possible usage models include:

- Using a common driver waveform for all cells.
- Using a different driver waveform for different categories of cells.
- Using a pin-specific driver waveform for complex cell pins.
- Using a different driver waveform for rise and fall timing arcs.

**Syntax**

The driver waveform syntax is as follows:

```plaintext
library(library_name) {
  ...
  lu_table_template (waveform_template_name) {
    variable_1: input_net_transition;
    variable_2: normalized_voltage;
    index_1 ("float..., float");
    index_2 ("float..., float");
  }
  normalized_driver_waveform(waveform_template_name) {
    driver_waveform_name : string; /* Specifies the name of the driver waveform table */
    index_1 ("float..., float"); /* Specifies input net transition */
    index_2 ("float..., float"); /* Specifies normalized voltage */
    values ( "float..., float", /* Specifies the time in library units */
      ..., "float..., float");
  }
  ...
  cell (cell_name) {
    ...
    driver_waveform : string;
    driver_waveform_rise : string;
    driver_waveform_fall : string;
    pin (pin_name) {
      driver_waveform : string;
      driver_waveform_rise : string;
      driver_waveform_fall : string;
      ...
    }
  }
}/* end of library*/
```

In the driver waveform syntax, the first index value in the table specifies the input slew and the second index value specifies the voltage normalized to VDD. The values in the table specify the time in library units (not scaled) when the waveform crosses the corresponding
voltages. The `driver_waveform_name` attribute specified for the driver waveform table differentiates the tables when multiple driver waveform tables are defined.

The cell-level `driver_waveform`, `driver_waveform_rise` and `driver_waveform_fall` attributes meet cell-specific and rise- and fall-specific requirements. The attributes refer to the driver waveform table name predefined at the library level.

Similar to the cell-level driver waveform attributes, the pin group includes the `driver_waveform`, `driver_waveform_rise` and `driver_waveform_fall` attributes to meet pin-specific predriver requirements for complex cell pins (such as macro cells). These attributes also refer to the predefined driver waveform table name.

---

### Library-Level Tables, Attributes, and Variables

This section describes driver waveform tables, attributes, and variables that are specified at the library level.

**normalized_voltage Variable**

The `normalized_voltage` variable is specified under the `lu_table_template` table to describe a collection of waveforms with various input slew values. For a given input slew in `index_1` (for example, `index_1[0] = 1.0 ns`), the `index_2` values are a set of points that represent how the voltage rises from 0 to VDD in a rise arc, or from VDD to 0 in a fall arc.

**Rise Arc Example**

```plaintext
normalized_driver_waveform (waveform_template) {
    index_1 ("1.0"); /* Specifies the input net transition*/
    index_2 ("0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0"); /* Specifies the voltage normalized to VDD */
    values ("0, 0.2, 0.4, 0.6, 0.8, 0.9, 1.1"); /* Specifies the time when the voltage reaches the index_2 values*/
}
```

The `lu_table_template` table represents an input slew of 1.0 ns, when the voltage is 0%, 10%, 30%, 50%, 70%, 90% or 100% of VDD, and the time values are 0, 0.2, 0.4, 0.6, 0.8, 0.9, 1.1 (ns). The time value can go beyond the corresponding input slew because a long tail might exist in the waveform before it reaches the final status.

**normalized_driver_waveform Group**

The library-level `normalized_driver_waveform` group represents a collection of driver waveforms under various input slew values. The `index_1` specifies the input slew and `index_2` specifies the normalized voltage.
The slew index in the `normalized_driver_waveform` table is based on the slew derate and slew trip points of the library (global values). When applied on a pin or cell with different slew or slew derate, the new slew should be interpreted from the waveform.

**driver_waveform_name Attribute**

The `driver_waveform_name` string attribute differentiates the driver waveform table from other driver waveform tables when multiple tables are defined. Cell-specific, rise-specific, and fall-specific driver waveform usage modeling depend on this attribute.

The `driver_waveform_name` attribute is optional. You can define a driver waveform table without the attribute, but there can be only one table in a library, and that table is regarded as the default driver waveform table for all cells in the library. If more than one table is defined without the attribute, the last table is used. The other tables are ignored and not stored in the library database.

---

**Cell-Level Attributes**

This section describes driver waveform attributes defined at the cell level.

**driver_waveform Attribute**

The `driver_waveform` attribute is an optional string attribute that allows you to define a cell-specific driver waveform. The value must be the `driver_waveform_name` predefined in the `normalized_driver_waveform` table.

When the attribute is defined, the cell uses the specified driver waveform during characterization. When it is not specified, the common driver waveform (the `normalized_driver_waveform` table without the `driver_waveform_name` attribute) is used for the cell.

**driver_waveform_rise and driver_waveform_fall Attributes**

The `driver_waveform_rise` and `driver_waveform_fall` string attributes are similar to the `driver_waveform` attribute. These two attributes allow you to define rise-specific and fall-specific driver waveforms. The `driver_waveform` attribute can coexist with the `driver_waveform_rise` and `driver_waveform_fall` attributes, though the `driver_waveform` attribute becomes redundant.

You should specify a driver waveform for a cell by using the following priority:

1. Use the `driver_waveform_rise` for a rise arc and the `driver_waveform_fall` for a fall arc during characterization. If they are not defined, specify the second and third priority driver waveforms.
2. Use the cell-specific driver waveform (defined by the `driver_waveform` attribute).

3. Use the library-level default driver waveform (defined by the `normalized_driver_waveform` table without the `driver_waveform_name` attribute).

The `driver_waveform_rise` attribute can refer to a `normalized_driver_waveform` that is either rising or falling. You can invert the waveform automatically during runtime if necessary.

---

**Pin-Level Attributes**

This section describes driver waveform attributes defined at the pin level.

**driver_waveform Attribute**

The `driver_waveform` attribute is the same as the `driver_waveform` attribute specified at the cell level. For more information, see “driver_waveform Attribute” on page 7-91.

**driver_waveform_rise and driver_waveform_fall Attributes**

The `driver_waveform_rise` and `driver_waveform_fall` attributes are the same as the `driver_waveform_rise` and `driver_waveform_fall` attributes specified at the cell level. For more information, see “driver_waveform_rise and driver_waveform_fall Attributes” on page 7-91.

---

**Driver Waveform Example**

```verbatim
library(test_library) {

  lu_table_template(waveform_template) {
    variable_1 : input_net_transition;
    variable_2 : normalized_voltage;
    index_1 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7");
    index_2 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }

  /* Specifies the default library-level driver waveform table (the default driver waveform without the driver_waveform attribute) */
  normalized_driver_waveform (waveform_template) { 
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", 
    "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", 
    ...
    "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }

  /* Specifies the driver waveform for the clock pin */
  normalized_driver_waveform (waveform_template) {
```
driver_waveform_name : clock_driver;
    index_1 ("0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75");
    index_2 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
    values ("0.012, 0.03, 0.045, 0.06, 0.075, 0.090, 0.105, 0.13, 0.145", \\
            "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
/* Specifies the driver waveform for the bus */
normalized_driver_waveform (waveform_template) {
    driver_waveform_name : bus_driver;
    index_1 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7");
    index_2 ("0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95");
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", \\
            "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", \\
            "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
/* Specifies the driver waveform for the rise */
normalized_driver_waveform (waveform_template) {
    driver_waveform_name : rise_driver;
    index_1 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7");
    index_2 ("0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95");
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", \\
            "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", \\
            "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
/* Specifies the driver waveform for the fall */
normalized_driver_waveform (waveform_template) {
    driver_waveform_name : fall_driver;
    index_1 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7");
    index_2 ("0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95");
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", \\
            "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", \\
            "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
...
Timing information specified in libraries results from circuit simulator (library characterization) tools. The models themselves often represent only a partial record of how a particular arc was sensitized during characterization. To fully reproduce the conditions that allowed the model to be generated, the states of all the input pins on the cell must be known.

In composite current source (CCS) models, the accuracy requirements are very high (the expectation is as high as 2 percent compared to SPICE). To achieve this level of accuracy, correlation with SPICE requires that the cell conditions be represented exactly as during characterization. For more information about CCS models, see Chapter 10, “Composite Current Source Modeling.”

The following sections describe the pin sensitization condition information details used to characterize the timing data. The syntax predeclares state vectors as reusable sensitization patterns in the library. The patterns are referenced and instantiated as stimuli waveforms specific to timing arcs. The same sensitization pattern can be referenced by multiple cells or multiple timing arcs. One cell can also reference multiple sensitization patterns, which saves storage resources. The Liberty attributes and groups highlighted in the following sections help to specify the sensitization information of timing arcs during simulation and characterization.

### Sensitization Group

The **sensitization** group defined at the library level describes the complete state patterns for a specific list of pins (defined by the `pin_names` attribute) that is referenced and instantiated as stimuli in the timing arc.

Vector attributes in the group define all possible pin states used as stimuli. Actual stimulus waveforms can be described by a combination of these vectors. Multiple `sensitization` groups are allowed in a library. Each `sensitization` group can be referenced by multiple cells, and each cell can make reference to multiple `sensitization` groups.

The following attributes are library-level attributes under the `sensitization` group.
**pin_names Attribute**

The `pin_names` complex attribute defines a list of pin names. All vectors in this `sensitization` group are the exhaustive list of all possible transitions of the input pins and their subsequent output response.

The `pin_names` attribute is required, and it must be declared in the `sensitization` group before all vector declarations.

**vector Attribute**

Similar to the `pin_names` attribute, the `vector` attribute describes a transition pattern for the specified pins. The stimulus is described by an ordered list of vectors.

The two arguments for the `vector` attribute are as follows:

- **vector id**
  - The `vector id` argument is an identifier to the vector string. The `vector id` value must be an integer greater than or equal to zero and unique among all vectors in the current `sensitization` group.

- **vector string**
  - The `vector string` argument represents a pin transition state. The string consists of the following transition status values: 0, 1, X, and Z where each character is separated by a space. The number of elements in the vector string must equal the number of arguments in `pin_names`.

The `vector` attribute can also be declared as:

```plaintext
vector (positive_integer, "[0|1|X|Z] [0|1|X|Z]..."ammers);
```

**Example**

```plaintext
sensitization(sensitization_nand2) {
  pin_names ( IN1, IN2, OUT1);
  vector ( 1, "0 0 1" );
  vector ( 2, "0 1 1" );
  vector ( 3, "1 0 1" );
  vector ( 4, "1 1 0" );
}
```

---

**Cell-Level Attributes**

Generally, one cell references one `sensitization` group for cells. A cell-level attribute that can link the cell with a specific `sensitization` group helps to simplify sensitization usage. The following cell-level attributes ensure that `sensitization` groups can be referenced by cells with similar functionality but can have different pin names.
sensitization_master Attribute

The sensitization_master attribute defines the sensitization group referenced by the cell to generate stimuli for characterization. The attribute is required if the cell contains sensitization information. Its string value should be any sensitization group name predefined in the current library.

pin_name_map Attribute

The pin_name_map attribute defines the pin names that are used to generate stimuli from the sensitization group for all timing arcs in the cell. The pin_name_map attribute is optional when the pin names in the cell are the same as the pin names in the sensitization master, but it is required when they are different.

If the pin_name_map attribute is set, the number of pins must be the same as that in the sensitization master, and all pin names should be legal pin names in the cell.

Timing Group Attributes

This section describes the sensitization_master and pin_name_map timing-arc attributes. These attributes enable a complex cell (a macro in most cases) to refer to multiple sensitization groups. You can also specify a sampling vector and user-defined time intervals between vectors. The wave_rise and wave_fall attributes, which describe characterization stimuli (instantiated in pin timing arcs), are also discussed in this section.

sensitization_master Attribute

The sensitization_master attribute defines the sensitization group specific to the current timing group to generate stimulus for characterization. The attribute is optional when the sensitization master used for the timing arc is the same as that defined in the current cell, and it is required when they are different. Any sensitization group name predefined in the current library is a valid attribute value.

pin_name_map Attribute

Similar to the pin_name_map attribute defined in the cell level, the timing-arc pin_name_map attribute defines pin names used to generate stimulus for the current timing arc. The attribute is optional when pin_name_map pin names are the same as the following (listed in order of priority):

1. Pin names in the sensitization_master of the current timing arc.
2. Pin names in the pin_name_map attribute of the current cell group.
3. Pin names in the sensitization_master of the current cell group.
The `pin_name_map` attribute is required when `pin_name_map` pin names are different from all of the pin names in the previous list.

**wave_rise and wave_fall Attributes**

The `wave_rise` and `wave_fall` attributes represent the two stimuli used in characterization. The value for both attributes is a list of integer values, and each value is a vector ID predefined in the library sensitization group. The following example describes the `wave_rise` and `wave_fall` attributes:

```
wave_rise (vector_id[m]..., vector_id[n]);
wave_fall (vector_id[j]..., vector_id[k]);
```

**Example**

```liberty
library(my_library) {
  ...
  sensitization(sensi_2in_1out) {
    pin_names (IN1, IN2, OUT);
    vector (0, "0 0 0");
    vector (1, "0 0 1");
    vector (2, "0 1 0");
    vector (3, "0 1 1");
    vector (4, "1 0 0");
    vector (5, "1 0 1");
    vector (6, "1 1 0");
    vector (7, "1 1 1");
  }
  cell (my_nand2) {
    sensitization_master : sensi_2in_1out;
    pin_name_map (A, B, Z); /* these are pin names for the sensitization in this cell. */
    ...
    pin(A) {
      ...
    }
    Pin(B) {
      ...
    }
    pin(Z) {
      ...
      timing() {
        related_pin : "A";
        wave_rise (6, 3);
        /* 6, 3 - vector id in sensi_2in_1out sensitization group. Waveform interpretation of wave_rise is (for "A, B, Z" pins): 10 1 01 */
        wave_fall (3, 6);
        ...
      }
      timing() {
        related_pin : "B";
        wave_rise (7, 4); /* 7, 4 - vector id in sensi_2in_1out... */
      }
      ...
    }
    ...
  }
```

sensitization group. */

    wave_fall (4, 7);
    ...
} /* end pin(Z)*/
} /* end cell(my_nand2) */
...
} /* end library */

wave_rise_sampling_index and wave_fall_sampling_index Attributes

The wave_rise_sampling_index and wave_fall_sampling_index simple attributes override the default behavior of the wave_rise and wave_fall attributes. (The wave_rise and wave_fall attributes select the first and the last vectors to define the sensitization patterns of the input to the output pin transition that are predefined inside the sensitization template specified at the library level).

Example

wave_rise (2, 5, 7, 6); /* wave_rise ( wave_rise[0],
wave_rise[1], wave_rise[2], wave_rise[3] ); */

In the previous example, the wave rise vector delay is measured from the last transition (vector 7 changing to vector 6) to the output transition. The default wave_rise_sampling_index value is the last entry in the vector, which is 3 in this case (because the numbering begins at 0).

To override this default, set the wave_rise_sampling_index attribute, as shown:

wave_rise_sampling_index : 2;

When you specify this attribute, the delay is measured from the second last transition of the sensitization vector to the final output transition, in other words from the transition of vector 5 to vector 7.

Note:
You cannot specify a value of 0 for the wave_rise_sampling_index attribute.

wave_rise_time_interval and wave_fall_time_interval Attributes

The wave_rise_time_interval and wave_fall_time_interval attributes control the time interval between transitions. By default, the stimuli (specified in wave_rise and wave_fall) are widely spaced apart during characterization (for example, 10 ns from one vector to the next) to allow all output transitions to stabilize. The attributes allow you to specify a short duration between one vector to the next to characterize special purpose cells and pessimistic timing characterization.
The \texttt{wave\_rise\_time\_interval} and \texttt{wave\_fall\_time\_interval} attributes are optional when the default time interval is used for all transitions, and they are required when you need to define special time intervals between transitions. Usually, the special time interval is smaller than the default time interval.

The \texttt{wave\_rise\_time\_interval} and \texttt{wave\_fall\_time\_interval} attributes can have an argument count from 1 to \(n-1\), where \(n\) is the number of arguments in corresponding \texttt{wave\_rise} or \texttt{wave\_fall}. Use 0 to imply the default time interval used between vectors.

\textbf{Example}

\begin{verbatim}
wave_rise (2, 5, 7, 6); /* wave_rise (wave_rise[0], wave_rise[1], wave_rise[2], wave_rise[3]);*/
wave_rise_time_interval (0.0, 0.3);
\end{verbatim}

The previous example suggests the following:

\begin{itemize}
  \item Use the default time interval between \texttt{wave\_rise[0]} and \texttt{wave\_rise[1]} (in other words, vector 2 and vector 5).
  \item Use 0.3 between \texttt{wave\_rise[1]} and \texttt{wave\_rise[2]} (in other words, vector 5 and vector 7).
  \item Use the default time interval between \texttt{wave\_rise[2]} and \texttt{wave\_rise[3]} (in other words, vector 7 and vector 6).
\end{itemize}

\textbf{timing Group Syntax}

\begin{verbatim}
library(library_name) {
  ...
  sensitization (sensitization_group_name) {
    pin_names (string..., string);
    vector (integer, string);
    ...
    vector (integer, string);
  }
  ...
  cell(cell_name) {
    sensitization_master : sensitization_group_name;
    pin_name_map (string..., string);
    ...
    pin(pin_name) {
      ...
      timing() {
        related_pin : string;
        sensitization_master : sensitization_group_name;
        pin_name_map (string,..., string);
        wave_rise (integer,..., integer);
      }
    }
  }
\end{verbatim}
NAND Cell Example

The following is an example of a NAND cell with sensitization information.

```plaintext
library(cell1) {
  sensitization(sensitization_nand2) {
    pin_names (IN1, IN2, OUT1);
    vector (1, "0 0 1");
    vector (2, "0 1 1");
    vector (3, "1 0 1");
    vector (4, "1 1 0");
  }

  cell (nand2) {
    sensitization_master : sensitization_nand2;
    pin_name_map (A, B, Y);
    pin (A) {
      direction : input;
      ...
    }
    pin (B) {
      direction : input;
      ...
    }
    pin (Y) {
      direction : output;
      ...
      timing() {
        related_pin : "A";
        timing_sense : negative_unate;
        wave_rise (4, 2); /* 10 1 01 */
        wave_fall (2, 4); /* 01 1 10 */
        ...
      }
      timing() {
        related_pin : "B";
        timing_sense : negative_unate;
        wave_rise (4, 3);
        wave_fall (3, 4);
      }
  }
}
```
Complex Macro Cell Example

The following is an example of a complex macro cell highlighting the usage of the sensitization_master group inside the timing group.

```
library(cell1) {
    sensitization(sensitization_2in_1out) {
        pin_names ( IN1, IN2, OUT );
        vector ( 1, "0 0 1" );
        vector ( 2, "0 1 1" );
        vector ( 3, "1 0 1" );
        vector ( 4, "1 1 0" );
    }
    sensitization(sensitization_3in_1out) {
        pin_names ( IN1, IN2, IN3, OUT );
        vector ( 0, "0 0 0 0" );
        vector ( 1, "0 0 0 1" );
        vector ( 2, "0 0 1 0" );
        vector ( 3, "0 0 1 1" );
        vector ( 4, "0 1 0 0" );
        vector ( 5, "0 1 0 1" );
        vector ( 6, "0 1 1 0" );
        vector ( 7, "0 1 1 1" );
        vector ( 8, "1 0 0 0" );
        vector ( 9, "1 0 0 1" );
        vector (10, "1 0 1 0" );
        vector (11, "1 0 1 1" );
        vector (12, "1 1 0 0" );
        vector (13, "1 1 0 1" );
        vector (14, "1 1 1 0" );
        vector (15, "1 1 1 1" );
    }
    cell (nand2) {
        sensitization_master : sensitization_2in_1out;
        pin_name_map (A, B, Y);
        pin (A) {
            direction : input ;
            ...
        }
        pin (B) {
            direction : input ;
            ...
        }
        pin (CIN0) {
            direction : input ;
        }
    }
}
```
pin (CIN1) {
  direction : input;
  ...
}

pin (CK) {
  direction : input;
  ...
}

pin (Y) {
  direction : output;
  ...

timing() {
  related_pin : "A";
  /* inherit sensitization_master & pin_name_map from cell level */
  timing_sense : negative_unate;
  wave_rise ( 4, 2);
  wave_fall ( 2, 4);
  ...
}

timing() {
  related_pin : "B";
  timing_sense : negative_unate;
  wave_rise ( 4, 3);
  wave_fall ( 3, 4);
  ...
}

} /* end pin(Y) */

pin (Z) {
  direction : output;
  ...

timing () {
  related_pin : "CK";
  sensitization_master : sensitization_3in_1out; /* timing arc specific sensitization master, overwrite the cell level attribute. */
  pin_name_map (CIN0, CIN1, CK, Z); /* timing arc specific pin_name_map, overwrite the cell level attribute. */
  wave_rise (14, 4, 0, 3, 10, 5); /* the waveform describe here has no real meaning, just select random vector id in sensitization sensitization_3in_1out group. */
  wave_fall (15, 9, 3, 1, 6, 7);
  wave_rise_sampling_index : 3; /* sampling index, specific for this timing arc. */
  wave_fall_sampling_index : 3;
  wave_rise_timing_interval(0, 0.3, 0.3); /* special timing interval, specific for this timing arc. */
  wave_fall_timing_interval(0, 0.3, 0.3);
Phase-Locked Loop Support

A phase-locked loop (PLL) is a feedback control system that automatically adjusts the phase of a locally-generated signal to match the phase of an input signal. Phase-locked loops contain the following pins:

- The reference clock pin where the reference clock is connected
- The phase-locked loop output clock pin where the phase-locked loop generates a phase-shifted version of the reference clock
- The feedback pin where the feedback path from the output of the clock ends

A phase-locked loop reduces the large skew between the clocks arriving at the launch point and at the flip-flops. The large skew results in an extremely tight delay constraint for the combinational logic.

**Figure 7-21  Phase-Locked Loop Circuit**

The phase-locked loop generates a phase-shifted version of the reference clock at the output pin so that the phase of the feedback clock matches the phase of the reference clock.

In Figure 7-21, if the clock arrives at the reference pin of the phase-locked loop at the time, $D_{\text{clock}}$, and the delay of the feedback path is $D_{\text{feedback}}$, the source latency of the clock generated at the output of the phase-locked loop is $D_{\text{clock}} - D_{\text{feedback}}$. The clock arrives at the feedback pin at time, $D_{\text{clock}}$, which is the same as the time the clock arrives at the reference pin. Therefore, the phase-locked loop eliminates the latency on the launch path and relaxes the delay constraint on the combinational logic.
The phase-locked loop information that a library must contain are pins and timing arcs inside the phase-locked loop. In Figure 7-22, the forward arcs from REF_CLK to PLL_OUT_CLK_1 in the figure simulate the phase shift behavior of the phase-locked loop. There are two half-unate arcs from the reference clock to each output of the phase-locked loop.

**Figure 7-22 Simple Phase-Locked Loop Model**

![Simple Phase-Locked Loop Model](image)

---

**Phase-Locked Loop Syntax**

cell (cell_name) {
    is_pll_cell : true;
    pin (ref_pin_name) {
        is_pll_reference_pin : true;
        direction : output;
        ...
    }
    pin (feedback_pin_name) {
        is_pll_feedback_pin : true;
        direction : output;
        ...
    }
    pin (output_pin_name) {
        is_pll_output_pin : true;
        direction : output;
        ...
    }
}

---

**Cell-Level Attribute**

This section describes a cell-level attribute.

**is_pll_cell Attribute**

The `is_pll_cell` Boolean attribute identifies a phase-locked loop cell.

---

**Pin-Level Attributes**

This section describes pin-level attributes.
**is_pll_reference_pin Attribute**

The `is_pll_reference_pin` Boolean attribute tags a pin as a reference pin on the phase-locked loop. In a phase-locked loop cell group, the `is_pll_reference_pin` attribute should be set to true in only one input pin group.

**is_pll_feedback_pin Attribute**

The `is_pll_feedback_pin` Boolean attribute tags a pin as a feedback pin on a phase-locked loop. In a phase-locked loop cell group, the `is_pll_feedback_pin` attribute should be set to true in only one input pin group.

**is_pll_output_pin Attribute**

The `is_pll_output_pin` Boolean attribute tags a pin as an output pin on a phase-locked loop. In a phase-locked loop cell group, the `is_pll_output_pin` attribute should be set to true in one or more output pin groups.

---

**Phase-Locked Loop Example**

cell(my_pll) {
    is_pll_cell : true;
    
    pin( REFCLK ) {
        direction : input;
        is_pll_reference_pin : true;
    }
    
    pin( FBKCLK ) {
        direction : input;
        is_pll_feedback_pin : true;
    }
    
    pin (OUTCLK_1x) {
        direction : output;
        is_pll_output_pin : true;
        
        timing() {/* Timing Arc */
            related_pin: "REFCLK";
            timing_type: combinational_rise;
            timing_sense: positive_unate;
        . . .
        }
        
        timing() {/* Timing Arc */
            related_pin: "REFCLK";
            timing_type: combinational_fall;
            timing_sense: positive_unate;
        . . .
    }
pin (OUTCLK_2x) {
  direction : output;
  is_pll_output_pin : true;
  timing() {/* Timing Arc */
    related_pin: "REFCLK";
    timing_type: combinational_rise;
    timing_sense: positive_unate;
    . . .
  }
  timing() {/* Timing Arc */
    related_pin: "REFCLK";
    timing_type: combinational_fall;
    timing_sense: positive_unate;
    . . .
  }
} /* End pin group */

/* End cell group */
Modeling Power and Electromigration

This chapter provides an overview of modeling static and dynamic power for CMOS technology.

To model CMOS static and dynamic power, you must understand the topics covered in the following sections:

- Modeling Power Terminology
- Switching Activity
- Modeling for Leakage Power
- Representing Leakage Power Information
- Threshold Voltage Modeling
- Modeling for Internal and Switching Power
- Representing Internal Power Information
- Defining Internal Power Groups
- Multiple Power Supply Library Requirements
- Modeling Libraries With Integrated Clock-Gating Cells
- Modeling Electromigration
Modeling Power Terminology

The power a circuit dissipates falls into two broad categories:

- Static power
- Dynamic power

Static Power

Static power is the power dissipated by a gate when it is not switching—that is, when it is inactive or static.

Static power is dissipated in several ways. The largest percentage of static power results from source-to-drain subthreshold leakage. This leakage is caused by reduced threshold voltages that prevent the gate from turning off completely. Static power also results when current leaks between the diffusion layers and substrate. For this reason, static power is often called leakage power.

Dynamic Power

Dynamic power is the power dissipated when a circuit is active. A circuit is active anytime the voltage on a net changes due to some stimulus applied to the circuit. Because voltage on a net can change without necessarily resulting in a logic transition, dynamic power can result even when a net does not change its logic state.

The dynamic power of a circuit is composed of

- Internal power
- Switching power

Internal Power

During switching, a circuit dissipates internal power by the charging or discharging of any existing capacitances internal to the cell. The definition of internal power includes power dissipated by a momentary short circuit between the P and N transistors of a gate, called short-circuit power.

Figure 8-1 shows the cause of short-circuit power. In this figure, there is a slow rising signal at the gate input IN. As the signal makes a transition from low to high, the N-type transistor turns on and the P-type transistor turns off. However, during signal transition, both the P- and N-type transistors can be on simultaneously for a short time. During this time, current flows from VDD to GND, resulting in short-circuit power.
Short-circuit power varies according to the circuit. For circuits with fast transition times, the amount of short-circuit power can be small. For circuits with slow transition times, short-circuit power can account for up to 30 percent of the total power dissipated. Short-circuit power is also affected by the dimensions of the transistors and the load capacitance at the output of the gate.

In most simple library cells, internal power is due primarily to short-circuit power. For this reason, the terms internal power and short-circuit power are often considered synonymous.

Note:
A transition implies either a rising or a falling signal; therefore, if the power characterization involves running a full-cycle simulation, which includes both rising and falling signals, then you must average the energy dissipation measurement by dividing by 2.

Switching Power
The switching power, or capacitance power, of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the output of the cell. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitances on the driver.

Because such charging and discharging is the result of the logic transitions at the output of the cell, switching power increases as logic transitions increase. The switching power of a
cell is the function of both the total load capacitance at the cell output and the rate of logic transitions.

*Figure 8-1* shows how the capacitance (C_load) is charged and discharged as the N or P transistor turns on. Switching power accounts for 70 to 90 percent of the power dissipation of an active CMOS circuit.

### Switching Activity

Switching activity is a metric used to measure the number of transitions (0-to-1 and 1-to-0) for every net in a circuit when input stimuli are applied. Switching activity is the average activity of the circuit with a set of input stimuli.

A circuit with higher switching activity is likely to dissipate more power than a circuit with lower switching activity.

### Modeling for Leakage Power

Regardless of the physical reasons for leakage power, library developers can annotate gates with the approximate total leakage power dissipated by the gate.

### Representing Leakage Power Information

You can represent leakage power information in the library as:

- Cell-level state-independent leakage power with the `cell_leakage_power` attribute
- Cell-level state-dependent leakage power with the `leakage_power` group
- The associated library-level attributes that specify scaling factors, units, and a default for both leakage and power density

### cell_leakage_power Simple Attribute

This attribute specifies the leakage power of a cell. If this attribute is missing or negative, the value of the `default_cell_leakage_power` attribute is used.

**Syntax**

```
cell_leakage_power : value ;
```
Note:
You must define this attribute for cells with state-dependent leakage power to provide the leakage power value for those states where the state-specific leakage power has not been specified using the leakage_power group.

Using the leakage_power Group for a Single Value
This group specifies the leakage power of a cell when the leakage power depends on the logical condition of that cell. This type of leakage power is called state-dependent. To model state-dependent leakage power, use the following attributes:

- mode
- when
- value

Syntax
leakage_power ( ) {
    mode (mode_name, mode_value);
    when : "Boolean expression";
    value: float;
}

mode Complex Attribute
You define the mode attribute within a leakage_power group. A mode attribute pertains to an individual cell. The cell is active when mode is instantiated with a name and a value. You can specify multiple instances of the mode attribute but only one instance for each cell.

Syntax
mode (mode_definition_name, mode_value);

Example
cell (my_cell) {
    mode_definition (mode_definition_name) {
        mode_value(namestring) {
            when : "Boolean expression";
            sdf_cond : "Boolean expression";
        }
    }
    leakage_power (namestring) {
        mode (mode_name, mode_value);
        /*when : "Boolean expression";*/
        value : float;
        ...
    }/* end leakage_power() */
leakage_current() { /* without the when statement */
  /* default state */
...
}
} /* end pin B */
} /* end cell */

Example 8-1  A mode Instance Description

library (my_library) {
  technology ( cmos ) ;
  delay_model : table_lookup;
  ...
  cell(inv0d0) {
    area : 0.75;
    mode_definition(rw) {
      mode_value(read) {
        when : "I";
        sdf_cond : "I == 1";
      }
      mode_value(write) {
        when : "!I";
        sdf_cond : "I == 0";
      }
    }
    leakage_power () {
      mode(rw, read);
      value : 2;
    }
    leakage_power () {
      mode(rw, write);
      value : 2.2;
    }
    pin(I) {
      direction : input;
      max_transition : 2100.0;
      capacitance : 0.002000;
      fanout_load : 1;
      ...
    }
  ...
} /* cell(inv0d0) */
} /* library */

when Simple Attribute

This attribute specifies the state-dependent condition that determines whether the leakage power is accessed.

Syntax

when : "Boolean expression";
**Boolean expression**

The name or names of pins, buses, and bundles with their corresponding Boolean operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{'} )</td>
<td>invert previous expression</td>
</tr>
<tr>
<td>(!)</td>
<td>invert following expression</td>
</tr>
<tr>
<td>( ^)</td>
<td>logical XOR</td>
</tr>
<tr>
<td>( *)</td>
<td>logical AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td>space</td>
<td>logical AND</td>
</tr>
<tr>
<td>( +)</td>
<td>logical OR</td>
</tr>
<tr>
<td>|</td>
<td>logical OR</td>
</tr>
<tr>
<td>1</td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td>0</td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>

The order of precedence of the operators is left to right, with inversion performed first, then XOR, then AND, then OR.

You must define mutually exclusive conditions for state-dependent leakage power and internal power. Mutually exclusive means that only one condition defined in the `when` attribute can be met at any given time.

You cannot directly reference a bus or a bundle in the `when` attribute in the `leakage_power` group. However, you can reference the pins of buses and bundles in the `when` attribute in the `leakage_power` group, as shown here:

```plaintext
Example

cell(Z) {
  ...
  cell_leakage_power : 3.0 ;
  leakage_power () {
    value : 2.0 ;
  }
}
```
value Simple Attribute

The value attribute represents the leakage power for a given state of a cell. The value for this attribute is a floating-point number.

Example

cell (my cell) {
    ...
    leakage_power () {
        when : "! A";
        value : 2.0;
    }
    cell_leakage_power : 3.0;
}

leakage_power_unit Simple Attribute

This attribute indicates the units of the leakage-power values in the library.

Table 8-2 Valid Unit Values and Mathematical Equivalents

<table>
<thead>
<tr>
<th>Text entry</th>
<th>Mathematical equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1mW</td>
<td>1mW</td>
</tr>
<tr>
<td>100uW</td>
<td>100 micro W</td>
</tr>
<tr>
<td>10uW</td>
<td>10 micro W</td>
</tr>
<tr>
<td>1uW</td>
<td>1 micro W</td>
</tr>
<tr>
<td>100nW</td>
<td>100nW</td>
</tr>
<tr>
<td>10nW</td>
<td>10nW</td>
</tr>
<tr>
<td>1nW</td>
<td>1nW</td>
</tr>
<tr>
<td>100pW</td>
<td>100pW</td>
</tr>
</tbody>
</table>
default_cell_leakage_power Simple Attribute

The default_cell_leakage_power attribute indicates the default leakage power for those cells for which you have not set the cell_leakage_power attribute. This attribute must be a nonnegative floating-point number. The default is 0.0.

Example

default_cell_leakage_power : 0.5;

Example

library(leakage) {
   delay_model : table_lookup;

   /* unit attributes */
   time_unit : "1ns";
   voltage_unit : "1V";
   current_unit : "1mA";
   pulling_resistance_unit : "1kohm";
   leakage_power_unit : "1pW";
   capacitive_load_unit (1.0,pf);

   cell (NAND2) {

      cell_leakage_power : 1.0;
      leakage_power() {
         related_pg_pin : "VDD1";
         when : "!A1 !A2";
         value : 1.5;
      }
      leakage_power() {
         related_pg_pin : "VDD1";
      }
   }
}
when : "!A1 A2";
value : 2.0;
}

leakage_power() {
related_pg_pin : "VDD1";
when : "A1 !A2";
value : 3.0;
}

leakage_power() {
related_pg_pin : "VDD1";
when : "A1 A2";
value : 4.0;
}

leakage_power() {
related_pg_pin : "VDD2";
when : "!A1 !A2";
value : 3.5;
}

leakage_power() {
related_pg_pin : "VDD2";
when : "!A1 A2";
value : 3.0;
}

leakage_power() {
related_pg_pin : "VDD2";
when : "A1 !A2";
value : 4.0;
}

leakage_power() {
related_pg_pin : "VDD2";
when : "A1 A2";
value : 5.0;
}

area : 1.0;

pin(A1) {
direction : input;
capacitance : 0.1;
}

pin(A2) {
direction : input;
capacitance : 0.1;
}

pin(ZN) {
direction : output;
max_capacitance : 0.1;
function : "(A1*A2)’";
timing() {
timing_sense : "negative_unate"
related_pin : "A1"
cell_rise( scalar ) {
values("0.0");
}
ris transition( scalar ) {
values("0.0");
}
cell_fall( scalar ) {
    values("0.0");
}
fall_transition( scalar ) {
    values("0.0");
}
internal_power() {
    related_pin : "A1"
    related_pg_pin : "VDD1"
    rise_power( scalar ) {
        values("0.0");
    }
    fall_power( scalar ) {
        values("0.0");
    }
} /* end of internal power */
internal_power() {
    related_pin : "A1"
    related_pg_pin : "VDD2"
    rise_power( scalar ) {
        values("0.0");
    }
    fall_power( scalar ) {
        values("0.0");
    }
} /* end of internal power */
} /* end of timing for related pin A1 */
timing() {
    timing_sense : "negative_unate"
    related_pin : "A2"
    cell_rise( scalar ) {
        values("0.0");
    }
    rise_transition( scalar ) {
        values("0.0");
    }
    cell_fall( scalar ) {
        values("0.0");
    }
    fall_transition( scalar ) {
        values("0.0");
    }
} internal_power() {
    related_pin : "A2"
    related_pg_pin : "VDD1"
    rise_power( scalar ) {
        values("0.0");
    }
    fall_power( scalar ) {
        values("0.0");
    }
Threshold Voltage Modeling

Multiple threshold power saving flows require library cells to be categorized according to the transistor’s threshold voltage characteristics, such as high threshold voltage cells and low threshold voltage cells. High threshold voltage cells exhibit lower power leakage but run slower than low threshold voltage cells.

You can specify low threshold voltage cells and high threshold voltage cells in the same library, simplifying library management and setup, by using the following optional attributes:

- **default_threshold_voltage_group**

  Specify the `default_threshold_voltage_group` attribute at the library level, as shown, to specify a cell’s category based on its threshold voltage characteristics:

  ```
  default_threshold_voltage_group : group_nameid ;
  ```

  The `group_name` value is a string representing the name of the category, such as `high_vt_cell` to represent a high voltage cell.

- **threshold_voltage_group**

  Specify the `threshold_voltage_group` attribute at the cell level, as shown, to specify a cell’s category based on its threshold voltage characteristics:

  ```
  threshold_voltage_group : group_nameid ;
  ```

  The `group_name` value is a string representing the name of the category. Typically, you would specify a pair of `threshold_voltage_group` attributes, one representing the high voltage and one representing the low voltage. However, there is no limit to the number of `threshold_voltage_group` attributes you can have in a library. If you omit this attribute, the value of the `default_threshold_voltage_group` attribute is applied to the cell.
Example

library ( mixed_vt_lib ) {
  ...
  default_threshold_voltage_group : "high_vt_cell" ;
  ...
  cell(ht_cell) {
    threshold_voltage_group : "high_vt_cell" ;
    ...
  }
  cell(lt_cell) {
    threshold_voltage_group : "low_vt_cell" ;
    ...
  }
}

Modeling for Internal and Switching Power

These are two compatible definitions of internal or short-circuit power:

- Short-circuit power is the power dissipated by the instantaneous short-circuit connection between Vdd and GND while the gate is in transition.

- Internal power is all the power dissipated within the boundary of the gate. This definition does not distinguish between the cell’s short-circuit power and the component of switching power that is being dissipated internally to the cell as a result of the drain-to-substrate capacitance that is being charged and discharged. In this definition, the interconnect switching power is the power dissipated because of lumped wire capacitance and input pin capacitances but not because of the output pin capacitance.

Library developers must choose one of these definitions and specify internal power and capacitance numbers accordingly. Library developers can choose

- To include the effect of the output capacitance in the internal_power attribute, which gives the output pins zero capacitance

- To give the output pins a real capacitance, which causes them to be included in the switching power, and model only short-circuit power as the cell’s internal power

Together, internal power and switching power contribute to the total dynamic power dissipation. Like switching power, internal power is dissipated whenever an output pin makes a transition.

Some power is also dissipated as a result of internal transitions that do not cause output transitions. However, those are relatively minor in comparison (they consume much less power) and should be ignored.
In Figure 8-2 Case 1, input B of the AND2 gate undergoes a 0-to-1 transition but the output remains stable at 0. This might consume a small amount of power as one of the N-transistors opens, but the current flow is very small.

In Figure 8-2 Case 2, input D of the multilevel gate AO22 undergoes a 1-to-0 transition, causing a 1-to-0 transition at internal pin Y. However, output Z remains stable at 1. The significance of the power dissipation in this case depends on the load of the internal wire connected to Y. In Case 1, power dissipation is negligible, but in Case 2, power dissipation might result in some inaccuracy.

You can set the internal_power group attribute so that multiple input or output pins that share logic can transition together within the same time period.

Pins transitioning within the same time period can lower the level of power consumption.

---

**Modeling Internal Power Lookup Tables**

The library group supports a one-, two-, or three-dimensional internal_power lookup table indexed by the total output load capacitances (best model), the input transition time, or both. The internal power lookup table uses the same syntax as the nonlinear lookup table for delay.

Figure 8-3 is an example of the two-dimensional lookup table for modeling output pin power in a cell.
Representing Internal Power Information

You can describe power dissipation in your libraries by using lookup tables.

Specifying the Power Model

Use the library level `power_model` attribute to specify the power model for your library. The valid value is `table_lookup`.

Using Lookup Table Templates

To represent internal power, you can create templates of common information that multiple lookup tables can use. Use the following groups and attributes to define your lookup tables:

- The library-level `power_lut_template` group
- The `internal_power` group (see “Defining Internal Power Groups” on page 8-18)
- The associated library-level attributes that specify the scaling factors and a default
power_lut_template Group

Use this library-level group to create templates of common information that multiple lookup tables can use. A table template specifies the table parameters and the breakpoints for each axis. Assign each template a name. Make the template name the group name of a fall_power group, power group, or rise_power group in the internal_power group.

Syntax

```
power_lut_template(name) {
  variable_1 : string ;
  variable_2 : string ;
  variable_3 : string ;
  index_1("float, ..., float") ;
  index_2("float, ..., float") ;
  index_3("float, ..., float") ;
}
```

Template Variables

The lookup table template for specifying power uses three associated variables to characterize cells in the library for internal power:

- `variable_1`, which specifies the first dimensional variable
- `variable_2`, which specifies the second dimensional variable
- `variable_3`, which specifies the third dimensional variable

These variables indicate the parameters used to index into the lookup table along the first, second, and third table axes.

Following are valid values for `variable_1`, `variable_2`, and `variable_3`:

- **total_output_net_capacitance**
  - The loading of the output net capacitance of the pin specified in the pin group that contains the internal_power group.

- **equal_or_opposite_output_net_capacitance**
  - The loading of the output net capacitance of the pin specified in the equal_or_opposite_output attribute in the internal_power group.

- **input_transition_time**
  - The input transition time of the pin specified in the related_pin attribute in the internal_power group.

For information about the `related_pin` attribute, see “Defining Internal Power Groups” on page 8-18.
Template Breakpoints

The index statements define the breakpoints for an axis. The breakpoints defined by `index_1` correspond to the parameter values indicated by `variable_1`. The breakpoints defined by `index_2` correspond to the parameter values indicated by `variable_2`. The breakpoints defined by `index_3` correspond to the parameter values indicated by `variable_3`.

You can overwrite the `index_1`, `index_2`, and `index_3` attribute values by providing the same `index_x` attributes in the `fall_power` group, `power` group, or `rise_power` group in the `internal_power` group.

The index values are lists of floating-point numbers greater than or equal to 0.0. The values in the list must be in an increasing order.

The number of floating-point numbers in the indexes determines the size of each dimension, as Example 8-2 illustrates.

For each `power_lut_template` group, you must define at least one `variable_1` and one `index_1`.

Example 8-2  `power_lut_template` Groups With One, Two, and Three-Dimensional Templates

```plaintext
power_lut_template (output_by_cap) {
  variable_1 : total_output_net_capacitance ;
  index_1 ("0.0, 5.0, 20.0") ;
}

power_lut_template (output_by_cap_and_trans) {
  variable_1 : total_output_net_capacitance ;
  variable_2 : input_transition_time ;
  index_1 ("0.0, 5.0, 20.0") ;
  index_2 ("0.1, 1.0, 5.0") ;
}

power_lut_template (input_by_trans) {
  variable_1 : input_transition_time ;
  index_1 ("0.0, 1.0, 5.0") ;
}

power_lut_template (output_by_cap2_and_trans) {
  variable_1 : total_output_net_capacitance ;
  variable_2 : input_transition_time ;
  variable_3 : equal_or_output_net_capacitance ;
  index_1 ("0.0, 5.0, 20.0") ;
  index_2 ("0.1, 1.0, 5.0") ;
  index_3 ("0.1, 0.5, 1.0") ;
}
```
Scalar power_lut_template Group

Use this group to model cells with no power consumption.

The syntax has a predefined template named scalar; its value size is 1. You can specify scalar as the group name of a fall_power group, power group, or rise_power group in the internal_power group.

Note:
You can use the scalar template with an assigned value of 0 (indicating that no power is consumed) for an internal_power group with a rise_power table and a fall_power table. When one group contains the scalar template, the other must contain one-, two-, or three-dimensional power values.

Defining Internal Power Groups

To specify the cell’s internal power consumption, use the internal_power group within the pin group in the cell.

If the internal_power group is not present in a cell, it is assumed that the cell does not consume any internal power. You can define the optional complex attribute index_1, index_2, or index_3 in this group to overwrite the index_1, index_2, or index_3 attribute defined in the library-level power_lut_template to which it refers.

Naming Power Relationships, Using the internal_power Group

Within the internal_power group you can identify the name or names of different power relationships. A single power relationship can occur between an identified pin and a single related pin identified with the related_pin attribute. Multiple power relationships can occur in many ways.

This list shows seven possible multiple power relationships. These relationships are described in more detail in the following sections:

• Between a single pin and a single related pin
• Between a single pin and multiple related pins
• Between a bundle and a single related pin
• Between a bundle and multiple related pins
• Between a bus and a single related pin
• Between a bus and multiple related pins
• Between a bus and related bus pins
Power Relationship Between a Single Pin and a Single Related Pin

Identify the power relationship that occurs between a single pin and a single related pin by entering a name in the `internal_power` group attribute as shown in the following example:

**Example**

```
cell (my_inverter) {
  ...
  pin (A) {
    direction : input;
    capacitance : 1;
  }
  pin (B) {
    direction : output
    function : "A'";
    internal_power (A_B) {
      related_pin : "A";
      ...
    } /* end internal_power() */
  } /* end pin B */
} /* end cell */
```

The power relationship is as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A_B</td>
</tr>
</tbody>
</table>

Power Relationships Between a Single Pin and Multiple Related Pins

This section describes how to identify the power relationships when an `internal_power` group is within a `pin` group and the power relationship has more than a single related pin. You identify the multiple power relationships on a name list entered with the `internal_power` group attribute as shown in the following example:

**Example**

```
cell (my_and) {
  ...
  pin (A) {
    direction : input;
    capacitance : 1;
  }
  pin (B) {
    direction : input;
    capacitance : 2;
  }
  pin (C) {
```
direction : output
function : "A B";
internal_power (A_C, B_C) {
   related_pin : "A B";
   ...
} /* end internal_power() */
} /* end pin B */
} /* end cell */

The power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>A_C</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>B_C</td>
</tr>
</tbody>
</table>

**Power Relationships Between a Bundle and a Single Related Pin**

When the `internal_power` group is within a `bundle` group that has several members that have a single related pin, enter the names of the resulting multiple power relationships in a name list in the `internal_power` group.

**Example**

```lua
... 
bundle (Q) {
   members (Q0, Q1, Q2, Q3);
   direction : output;
   function : "IQ";
   internal_power (G Q0, G Q1, G Q2, G Q3) {
      related_pin : "G";
   }
}
```

If G is a pin, as opposed to another `bundle` group, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Q0</td>
<td>G_Q0</td>
</tr>
<tr>
<td>G</td>
<td>Q1</td>
<td>G_Q1</td>
</tr>
<tr>
<td>G</td>
<td>Q2</td>
<td>G_Q2</td>
</tr>
<tr>
<td>G</td>
<td>Q3</td>
<td>G_Q3</td>
</tr>
</tbody>
</table>
If G is another bundle of member size 4 and G0, G1, G2, and G3 are members of bundle G, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0</td>
<td>Q0</td>
<td>G_Q0</td>
</tr>
<tr>
<td>G1</td>
<td>Q1</td>
<td>G_Q1</td>
</tr>
<tr>
<td>G2</td>
<td>Q2</td>
<td>G_Q2</td>
</tr>
<tr>
<td>G3</td>
<td>Q3</td>
<td>G_Q3</td>
</tr>
</tbody>
</table>

Note:
If G is a bundle of a member size other than 4, it is an error due to incompatible width.

Power Relationships Between a Bundle and Multiple Related Pins

When the internal_power group is within a bundle group that has several members, each having a corresponding related pin, enter the names of the resulting multiple power relationships as a name list in the internal_power group.

Example

```plaintext
bundle (Q){
    members (Q0, Q1, Q2, Q3);
    direction : output;
    function : "IQ";
    internal_power (G_Q0, H_Q0, G_Q1, H_Q1, G_Q2, H_Q2, G_Q3, H_Q3) {
        related_pin : "G H";
    }
}
```
If $G$ is a pin, as opposed to another bundle group, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G$</td>
<td>$Q_0$</td>
<td>$G_{Q0}$</td>
</tr>
<tr>
<td>$H$</td>
<td>$Q_0$</td>
<td>$H_{Q0}$</td>
</tr>
<tr>
<td>$G$</td>
<td>$Q_1$</td>
<td>$G_{Q1}$</td>
</tr>
</tbody>
</table>

If $G$ is another bundle of member size 4 and $G_0, G_1, G_2, \text{and } G_3$ are members of bundle $G$, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_0$</td>
<td>$Q_0$</td>
<td>$G_{Q0}$</td>
</tr>
<tr>
<td>$H$</td>
<td>$Q_0$</td>
<td>$H_{Q0}$</td>
</tr>
<tr>
<td>$G_1$</td>
<td>$Q_1$</td>
<td>$G_{Q1}$</td>
</tr>
<tr>
<td>$H$</td>
<td>$Q_1$</td>
<td>$H_{Q1}$</td>
</tr>
<tr>
<td>$G_2$</td>
<td>$Q_2$</td>
<td>$G_{Q2}$</td>
</tr>
<tr>
<td>$H$</td>
<td>$Q_2$</td>
<td>$H_{Q2}$</td>
</tr>
<tr>
<td>$G_3$</td>
<td>$Q_3$</td>
<td>$G_{Q3}$</td>
</tr>
<tr>
<td>$H$</td>
<td>$Q_3$</td>
<td>$H_{Q3}$</td>
</tr>
</tbody>
</table>

The same rule applies if $H$ is a size 4 bundle.

Note:

If $G$ is a bundle of a member size other than 4, it’s an error due to incompatible width.
Power Relationships Between a Bus and a Single Related Pin

This section describes how to identify the power relationships created when an internal_power group is within a bus group that has several bits that have the same single related pin. You identify the resulting multiple power relationships by entering a name list, using the internal_power group attribute.

Example

...  
bus (X){  
/*assuming MSB is X[0] */  
bus_type : bus4;  
direction : output;  
capacitance : 1;  
   pin (X[0:3]){  
      function : "B’";  
      internal_power (B_X0, B_X1, B_X2, B_X3){  
         related_pin : "B";  
      }  
   }  
}

If B is a pin, as opposed to another 4-bit bus, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>X[0]</td>
<td>B_X0</td>
</tr>
<tr>
<td>B</td>
<td>X[1]</td>
<td>B_X1</td>
</tr>
<tr>
<td>B</td>
<td>X[2]</td>
<td>B_X2</td>
</tr>
<tr>
<td>B</td>
<td>X[3]</td>
<td>B_X3</td>
</tr>
</tbody>
</table>
If B is another 4-bit bus and B[0] is the MSB for bus B, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[0]</td>
<td>X[0]</td>
<td>B_X0</td>
</tr>
</tbody>
</table>

**Power Relationships Between a Bus and Multiple Related Pins**

This section describes the power relationships created when an internal_power group is within a bus group that has several bits, where each bit has its own related pin. You identify the resulting multiple power relationships by entering a name list, using the internal_power group attribute.

**Example**

```plaintext
bus (X) { /*assuming MSB is X[0] */
    bus_type : bus4;
    direction : output;
    capacitance : 1;
    pin (X[0:3]){
        function : "B'";
        internal_power (B_X0, C_X0, B_X1, C_X1, B_X2, C_X2, B_X2, C_X2, B_X3, C_X3){
            related_pin : "B C";
        }
    }
}
```
If B and C are pins, as opposed to another 4-bit bus, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>X[0]</td>
<td>B_X0</td>
</tr>
<tr>
<td>C</td>
<td>X[0]</td>
<td>C_X0</td>
</tr>
<tr>
<td>B</td>
<td>X[1]</td>
<td>B_X1</td>
</tr>
<tr>
<td>C</td>
<td>X[1]</td>
<td>C_X1</td>
</tr>
<tr>
<td>B</td>
<td>X[2]</td>
<td>B_X2</td>
</tr>
<tr>
<td>C</td>
<td>X[2]</td>
<td>C_X2</td>
</tr>
<tr>
<td>B</td>
<td>X[3]</td>
<td>B_X3</td>
</tr>
<tr>
<td>C</td>
<td>X[3]</td>
<td>C_X3</td>
</tr>
</tbody>
</table>

If B is another 4-bit bus and B[0] is the MSB for bus B, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[0]</td>
<td>X[0]</td>
<td>B_X0</td>
</tr>
<tr>
<td>C</td>
<td>X[0]</td>
<td>C_X0</td>
</tr>
<tr>
<td>C</td>
<td>X[1]</td>
<td>C_X1</td>
</tr>
<tr>
<td>C</td>
<td>X[2]</td>
<td>C_X2</td>
</tr>
<tr>
<td>C</td>
<td>X[3]</td>
<td>C_X3</td>
</tr>
</tbody>
</table>

The same rule applies if C is a 4-bit bus.
Power Relationships Between a Bus and Related Bus Pins

This section describes the power relationships created when an internal_power group is within a bus group that has several bits that have to be matched with several related bus pins of a required width. Identify the resulting multiple power relationships by entering a name list, using the internal_power group attribute.

Example

/* assuming related_bus_pins is width of 2 bits */
basis (X){
    /*assuming MSB is X[0] */
    bus_type : bus4;
direction : output;
capacitance : 1;

    pin (X[0:3]){  
        function : "B’";
        internal_power (B0_X0, B0_X1, B0_X2, B0_X3, B1_X0, B1_X1, B1_X2, B1_X3){
            related_bus_pins : "B";
        }
    }
}

If B is another 2-bit bus and B[0] is its MSB, the power relationships are as follows:

<table>
<thead>
<tr>
<th>From pin</th>
<th>To pin</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[0]</td>
<td>X[0]</td>
<td>B0_X0</td>
</tr>
<tr>
<td>B[0]</td>
<td>X[1]</td>
<td>B0_X1</td>
</tr>
<tr>
<td>B[0]</td>
<td>X[2]</td>
<td>B0_X2</td>
</tr>
<tr>
<td>B[0]</td>
<td>X[3]</td>
<td>B0_X3</td>
</tr>
<tr>
<td>B[1]</td>
<td>X[0]</td>
<td>B1_X0</td>
</tr>
</tbody>
</table>

internal_power Group

To define an internal_power group in a pin group, use these simple attributes, complex attributes, and groups:
Simple attributes:

- equal_or_opposite_output
- falling_together_group
- related_pg_pin
- related_pin
- rising_together_group
- switching_interval
- switching_together_group
- when

Complex attribute:

- mode

Groups:

- fall_power
- power
- rise_power

equal_or_opposite_output Simple Attribute

This attribute designates an optional output pin or pins whose capacitance can be used to access a three-dimensional table in the internal_power group.

Syntax

equal_or_opposite_output : "name | name_list" ;

name | name_list

Name of output pin or pins.

Note:
This pin (or these pins) have to be functionally equal to or the opposite of the pin named in the same pin group.

Example

equal_or_opposite_output : "Q" ;
Note:

The output capacitance of this pin (or pins) is used as the equal_or_opposite_output_net_capacitance value in the internal power lookup table.

**falling_together_group Simple Attribute**

Use the **falling_together_group** attribute to identify the list of two or more input or output pins that share logic and are falling together during the same time period. Set this time period with the **switching_interval** attribute. See "switching_interval Simple Attribute" on page 8-32 for details.

Together, the **falling_together_group** attribute and the **switching_interval** attribute settings determine the level of power consumption.

Define a **falling_together_group** attribute in the **internal_power_group** in a **pin_group**, as shown here.

```plaintext
cell (namestring) {
  pin (namestring) {
    internal_power () {
      falling_together_group : "list of pins" ;
      rising_together_group : "list of pins" ;
      switching_interval : float;
      rise_power () {
        ...
      }
      fall_power () {
        ...
      }
    }
  }
}
```

*list of pins*

The names of the input or output pins that share logic and are falling during the same time period.

**power_level Simple Attribute**

This attribute is used for multiple power supply modeling. In the **internal_power_group** at the **pin_level**, you can specify the power level used to characterize the tables.

**Syntax**

```plaintext
power_level : "name" ;
```
name

Name of the power rail defined in the power_supply group.

Example

```
power_level : "VDD1" ;
```

For an output pin within a multiple power supply cell, regardless of the output_signal_level value, you must define internal_power tables for all the rail_connection of the cell.

Example

```
cell ( cell7 ) {
  area : 6.000 ;
  rail_connection(PV1, VDD1);
  rail_connection(PV2, VDD2);
  pin ( Z ) {
    direction : output ;
    capacitance : 0.000 ;
    function : "((A & B))";
    output_signal_level : VDD1;
    internal_power() {
      related_pin : "A";
      power_level : VDD1;  /* must be there */
      power (power_load) {
        values (" 0.111111, 0.111111, 0.111111, 0.111111, 0.111111");
      }
    }
  internal_power() {
    related_pin : "A";
    power_level : VDD2;  /* must be there */
    power (power_load) {
      values (" 0.111111, 0.111111, 0.111111, 0.111111, 0.111111");
    }
  }
  ...
}
```

For an input pin within a multiple power supply cell, you must define an internal_power table to match the input_signal_level value. The rest of the rail connections are optional.

Example

```
cell ( cell1 ) {
  rail_connection(PV1, VDD1);
  rail_connection(PV2, VDD2);
  rail_connection(PV3, VDD3);
  pin ( CP ) {
    direction : input ;
    input_signal_level : VDD1;

    internal_power() {
      power_level : VDD1;  /* this table is required */
      power (power_ramp) {
```
If you want to use the same Boolean expression for multiple `when` statements in an `internal_power` group, you must specify a different power rail for each `internal_power` group, as shown in this example.

**Example**

```liberty
library (example) {
  ...
  power_supply() {
    default_power_rail : vdd;
    power_rail(VDD1, 1.95);
    power_rail(VDD2, 1.85);
    power_rail(VDD3, 1.75);
  }
  ...
  cell (cell1) {
    rail_connection(PV1, VDD1);
    rail_connection(PV2, VDD2);
    rail_connection(PV3, VDD3);
    pin(A) {
      ...
    }
    pin(B) {
      ...
    }
    pin(CP) {
      direction : input;
      input_signal_level : VDD1;
      ...
      internal_power() {
        power_level : VDD1;
        when : "A !B";
        power (power_ramp) {
          values ("1.934150, 2.148130, 2.449420, 3.345050, 4.305690");
        }
      }
      internal_power() {
        power_level : VDD2;
        when : "A !B";
        power (power_ramp) {
          values ("1.934150, 2.148130, 2.449420, 3.345050, 4.305690");
        }
      }
    }
  }
}
```
/* no table for VDD3 */
}
}
}

related_pin Simple Attribute

This attribute associates the internal_power group with a specific input or output pin. If related_pin is an output pin, it must be functionally equal to or the opposite of the pin in that pin group.

If related_pin is an input pin or output pin, the pin’s transition time is used as a variable in the internal power lookup table.

**Syntax**

related_pin : "name | name_list" ;

*name | name_list*

Name of the input or output pin or pins

**Example**

related_pin : "A B" ;

The pin or pins in the related_pin attribute denote the path dependency for the internal_power group.

If you want to define a two-dimensional or three-dimensional table, specify all functionally related pins in a related_pin attribute.

rising_together_group Simple Attribute

The rising_together_group attribute identifies the list of two or more input or output pins that share logic and are rising during the same time period. This time period is defined with the switching_interval attribute. See the following "switching_interval Simple Attribute," section for details.

Together, the rising_together_group attribute and switching_interval attribute settings determine the level of power consumption.

Define the rising_together_group attribute in the internal_power group, as shown here.

cell (name_string) {
    pin (name_string) {
        internal_power () {
            falling_together_group : "list of pins" ;
            rising_together_group : "list of pins" ;
            switching_interval : float;
list of pins

The names of the input or output pins that share logic and are rising during the same time period.

switching_interval Simple Attribute

The `switching_interval` attribute defines the time interval during which two or more pins that share logic are falling, rising, or switching (either falling or rising) during the same time period.

Set the `switching_interval` attribute together with the `falling_together_group`, `rising_together_group`, or `switching_together_group` attribute. Together with one of these attributes, the `switching_interval` attribute defines a level of power consumption.

For details about the attributes that are set together with the `switching_interval` attribute, see “`falling_together_group` Simple Attribute” on page 8-28; “`rising_together_group` Simple Attribute” on page 8-31; and the following “`switching_together_group` Simple Attribute,” section.

Syntax

```
switching_interval : float ;
```

float

A floating-point number that represents the time interval during which two or more pins that share logic are switching together.

Example

```python
pin (Z) {
    direction : output;
    internal_power () {
        switching_together_group : "A B";
        /*if pins A, B, and Z switch*/
        switching_interval : 5.0;
        /*switching within 5 time units*/
        power () {
            ...
        }
    }
}
```
switching_together_group Simple Attribute

The `switching_together_group` attribute identifies the list of two or more input or output pins that share logic, are either falling or rising during the same time period, and are not affecting power consumption.

Define the time period with the `switching_interval` attribute. See “`switching_interval Simple Attribute`” on page 8-32 for details.

Define the `switching_together_group` attribute in the `internal_power` group, as shown here.

```
cell (namestring) {
  pin (namestring) {
    internal_power () {
      switching_together_group : "list of pins" ;
      switching_interval : float;
      power () {
        ...
      }
    }
  }
}
```

`list of pins`

The names of the input or output pins that share logic, are either falling or rising during the same time period, and are not affecting power consumption.

when Simple Attribute

This attribute specifies a state-dependent condition that determines whether the internal power table is accessed.

You can use the `when` attribute to define one-, two-, or three-dimensional tables in the `internal_power` group.

Syntax

```
when : "Boolean expression" ;
```

**Boolean expression**

Name or names of input and output pins, buses, and bundles with corresponding Boolean operators.

Table 8-1 on page 8-7 lists the Boolean operators valid in a `when` statement.
Example
when : "A B";

mode Complex Attribute
You define the mode attribute within an internal_power group. A mode attribute pertains to an individual pin. The pin is active when mode is instantiated with a name and a value. You can specify multiple instances of the mode attribute but only one instance for each pin.

Syntax
mode (mode_definition_name, mode_value);

Example
cell (my_cell) {
  mode_definition (mode_definition_name) {
    mode_value(namestring) {
      when : "Boolean expression";
      sdf_cond : "Boolean expression";
    } ...

    pin (pin_name) {
      direction : output;
      function : "Boolean expression";
      internal_power () {
        related_pin : "pin_name";
        when : "Boolean expression";
        /*when : "Boolean expression";*/
        mode (mode_definition_name, mode_value);
        ...
    } /* end internal_power() */
  } /* end pin */
} /* end cell */

Example 8-3 A mode Instance Description
library (my_library) {
  technology ( cmos );
  delay_model : table_lookup;
...
  cell(inv0d0) {
    area : 0.75;
    mode_definition(rw) {
      mode_value(read) {
        when : "A";
        sdf_cond : "A == 1";
      }
      mode_value(write) {
        when : "!A";
        sdf_cond : "A == 0";
      }
    }
    pin(A) {
      direction : input;
max_transition : 2100.0;
capacitance : 0.002000;
fanout_load : 1;
...
}
pin(I) {
  direction : input;
  max_transition : 2100.0;
capacitance : 0.002000;
fanout_load : 1;
...
}
pin(Z) {
  direction : output;
  max_capacitance : 0.175000;
  max_fanout : 58;
  max_transition : 1400.0;
  function : "(I)'";
  internal_power () {
    related_pin : "I";
    mode(rw, read);
    /* when : "A";*/
    ...
  }
  internal_power () {
    related_pin : "I";
    mode(rw, write);
    /* when : "!A";*/
    ...
  }
timing() {
  related_pin : "I";
  timing_sense : negative_unate;
  ...
} /* timing I -> Z */
} /* I */
...}
/* cell(inv0d0) */
} /* library */

fall_power Group

Use a fall_power group to define a fall transition for a pin. If you specify a fall_power group, you must also specify a rise_power group.

You define a fall_power group in an internal_power group in a cell-level pin group, as shown here:

cell (name_string) {
pin (name_string) {
  internal_power () {
    fall_power (template name) {
      ...
      fall power description ...
    }
  }
}
Complex Attributes

index_1 ("float, ..., float") ; /* lookup table */
index_2 ("float, ..., float") ; /* lookup table */
index_3 ("float, ..., float") ; /* lookup table */
values ("float, ..., float") ; /* lookup table */

index_1, index_2, index_3 Attributes

These attributes identify internal cell consumption per fall transition. Define these attributes in the internal_power group or in the library-level power_lut_template group.

values Attribute

This attribute defines internal cell consumption per fall transition.

Example

values ("2.2, 3.7, 4.3", "1.7, 2.1, 3.5", "1.0, 1.5, 2.8");

power Group

The power group is defined within an internal_power group in a pin group at the cell level, as shown here:

library (name) {
  cell (name) {
    pin (name) {
      internal_power () {
        power (template name) {
          ... power template description ...
        }
      }
    }
  }
}

Complex Attributes

index_1 ("float, ..., float") ; /* lookup table */
index_2 ("float, ..., float") ; /* lookup table */
index_3 ("float, ..., float") ; /* lookup table */
values ("float, ..., float") ; /* lookup table */

index_1, index_2, index_3 Attributes

These attributes identify internal cell consumption per transition. Define these attributes in the internal_power group or in the library-level power_lut_template group.
values Attribute

This attribute defines internal cell power consumption per rise or fall transition.

This power information is accessed when the pin has a rise transition or a fall transition. The values in the table specify the average power per transition.

rise_power Group

A rise_power group is defined in an internal_power group at the cell level, as shown here:

cell (name) {
  pin (name) {
    internal_power () {
      rise_power (template name) {
        ... rise power description ...
      }
    }
  }
}

Rise power is accessed when the pin has a rise transition. If you have a rise_power group, you must have a fall_power group.

Complex Attributes

index_1 ("float, ..., float") ; /* lookup table */
index_2 ("float, ..., float") ; /* lookup table */
index_3 ("float, ..., float") ; /* lookup table */
values ("float, ..., float") ; /* lookup table */

index_1, index_2, index_3 Attributes

These attributes identify internal cell consumption per rise transition. Define these attributes in the internal_power group or in the library-level power_lut_template group.

values Attribute

This attribute defines internal cell power consumption per rise transition.

Syntax for One-Dimensional, Two-Dimensional, and Three-Dimensional Tables

You can define a one-, two-, or three-dimensional table in the internal_power group in either of the following two ways:

• Using the power group. For two- and three-dimensional tables, define the power group and the related_pin attribute.

• Using a combination of the related_pin attribute, the fall_power group, and the rise_power group.
The syntax for a one-dimensional table using the `power` group is shown here:

```plaintext
internal_power() {
    power (template name) {
        values("float, ..., float") ;
    }
}
```

The syntax for a one-dimensional table using the `fall_power` and `rise_power` groups is shown here:

```plaintext
internal_power() {
    fall_power (template name) {
        values("float, ..., float");
    }
    rise_power (template name) {
        values("float, ..., float");
    }
}
```

The syntax for a two-dimensional table using the `power` and `related_pin` groups is shown here:

```plaintext
internal_power() {
    related_pin : "name | name_list" ;
    power (template name) {
        values("float, ..., float") ;
    }
}
```

The syntax for a two-dimensional table using the `related_pin`, `fall_power`, and `rise_power` groups is shown here:

```plaintext
internal_power() {
    related_pin : "name | name_list" ;
    fall_power (template name) {
        values("float, ..., float");
    }
    rise_power (template name) {
        values("float, ..., float");
    }
}
```

The syntax for a three-dimensional table using the `power` and `related_pin` groups is shown here:

```plaintext
internal_power() {
    related_pin : "name | name_list" ;
    power (template name) {
        values("float, ..., float") ;
    }
    equal_or_opposite_output : "name | name_list" ;
}
The syntax for a three-dimensional table using the `related_pin`, `fall_power`, and `rise_power` groups is shown here:

```plaintext
internal_power() {
    related_pin : "name | name_list";
    fall_power (template name) {
        values ("float, ..., float") ;
    }
    rise_power (template name) {
        values ("float, ..., float") ;
    }
    equal_or_opposite_output : "name | name_list" ;
}
```

### Internal Power Examples

The examples in this section show how to describe the internal power of the 2-input sequential gate in Figure 8-4, using one-, two-, and three-dimensional lookup tables.

#### Example 8-4 One-Dimensional Internal Power Table

Example 8-4 is the library description of the cell shown in Figure 8-4, a cell with a one-dimensional internal power table defined in the `pin` groups.

```
library(internal_power_example) {

    ... 
    power_lut_template(output_by_cap) {
        variable_1 : total_output_net_capacitance ;
        index_1("0.0, 5.0, -20.0") ;
    }

    ...

```
power_lut_template(input_by_trans) {
  variable_1 : input_transition_time;
  index_1 ("0.0, 1.0, 2.0");
}

... cell(FLOP1) {
  pin(CP) {
    direction : input;
    internal_power() {
      power(input_by_trans) {
        values("1.5, 2.6, 4.7");
      }
    }
  }
  ...
  pin(Q) {
    direction : input;
    internal_power() {
      power(output_by_cap) {
        values("9.0, 5.0, 2.0");
      }
    }
  }
}

Two-Dimensional Power Lookup Table

Example 8-5 is the library description of the cell in Figure 8-4, a cell with a two-dimensional internal power table defined in the pin groups.

Example 8-5 Two-Dimensional Internal Power Table

library(internal_power_example) {
  ...
  power_lut_template(output_by_cap_and_trans) {
    variable_1 : total_output_net_capacitance;
    variable_2 : input_transition_time;
    index_1 ("0.0, 5.0, 20.0");
    index_2 ("0.0, 1.0, 20.0");
  }
  ...
  cell(AN2) {
    pin(Z) {
      direction : output;
      internal_power {
        power(output_by_cap_and_trans) {
          values ("2.2, 3.7, 4.3", "1.7, 2.1, 3.5", "1.0, 1.5, 2.8");
        }
        related_pin : "A B";
      }
    }
    pin(A) {
      direction : input;
      ...

Three-Dimensional Power Lookup Table

Example 8-6 is the library description for the cell in Figure 8-4, a cell with a three-dimensional internal power table.

Example 8-6 Three-Dimensional Internal Power Table

library(internal_power_example) {
  ...
  power_lut_template(output_by_cap1_cap2_and_trans) {
    variable_1 : total_output1_net_capacitance;
    variable_2 : equal_or_opposite_output_net_capacitance;
    variable_3 : input_transition_time;
    index_1 ("0.0, 5.0, 20.0");
    index_2 ("0.0, 5.0, 20.0");
    index_3 ("0.0, 1.0, 2.0");
  }
  ...
  cell(FLOP1) {
    pin(CP) {
      direction : input;
      ...
    }
    pin(D) {
      direction : input;
      ...
    }
    pin(S) {
      direction : input;
      ...
    }
    pin(R) {
      direction : input;
      ...
    }
    pin(Q) {
      direction : output;
      internal_power() {
        power(output_by_cap1_cap2_and_trans) {
          values("2.2, 3.7, 4.3", "1.7, 2.1, 3.5", "1.0, 1.5, .8", "2.1, 3.6, 4.2", "1.6, 2.0, 3.4", "0.9, 1.5, .7" \
          "2.0, 3.5, 4.1", "1.5, 1.9, 3.3", "0.8, 1.4, 2.6");
        equal_or_opposite_output : "QN";
        related_pin : "CP";
      }
    }
  }
  ...
}
Modeling Libraries With Integrated Clock-Gating Cells

Power optimization achieved at high levels of abstraction has a significant impact on reduction of power in the final gate-level design. Clock gating is an important high-level technique for reducing power.

What Clock Gating Does

Clock gating provides a power-efficient implementation of register banks that are disabled during some clock cycles.

A register bank is a group of flip-flops that share the same clock and synchronous control signals and that are inferred from the same HDL variable. Synchronous control signals include synchronous load-enable, synchronous set, synchronous reset, and synchronous toggle.

Figure 8-5 shows a simple implementation of a register bank using a multiplexer and a feedback loop.

Figure 8-5  Synchronous Load-Enable Register Using a Multiplexer

When the synchronous load-enable signal (EN) is at logic state 0, the register bank is disabled. In this state, the circuit uses the multiplexer to feed the Q output of each storage element in the register bank back to the D input. When the EN signal is at logic state 1, the register is enabled, allowing new values to load at the D input.
Such feedback loops can use some power unnecessarily. For example, if the same value is reloaded in the register throughout multiple clock cycles (EN equals 0), the register bank and its clock net consume power while values in the register bank do not change. The multiplexer also consumes power.

By controlling the clock signal for the register, you can eliminate the need for reloading the same value in the register through multiple clock cycles. Clock gating inserts a 2-input gate into the register bank’s clock network, creating the control to eliminate unnecessary register activity.

Clock gating reduces the clock network’s power dissipation and often relaxes the datapath timing. If your design has large multibit registers, clock gating can save power and reduce the number of gates in the design. However, for smaller register banks, the overhead of adding logic to the clock tree might not compare favorably to the power saved by eliminating a few feedback nets and multiplexers.

Using integrated clock-gating cell functionality, you have the option of doing the following:

- Use latch-free or latch-based clock gating.
- Insert logic to increase testability.

For details, see “Using an Integrated Clock-Gating Cell” and “Setting Pin Attributes for an Integrated Cell” on page 8-45.

---

**Looking at a Gated Clock**

Clock gating saves power by eliminating the unnecessary activity associated with reloading register banks. Clock gating eliminates the feedback net and multiplexer shown in Figure 8-5, by inserting a 2-input gate in the clock net of the register. The 2-input clock gate selectively prevents clock edges, thus preventing the gated clock signal from clocking the gated register.

Clock gating can also insert inverters or buffers to satisfy timing or clock waveform and duty requirements.

Figure 8-6 shows the 2-input clock gate as an AND gate; however, depending on the type of register and the gating style, gating can use NAND, OR, and NOR gates instead.
Figure 8-6  Latch-Based Clock Gating

The clock input to the register bank, ENCLK, is gated on or off by the AND gate. ENL is the enabling signal that controls the gating; it derives from the EN signal on the multiplexer shown in Figure 8-5. The register is triggered by the rising edge of the ENCLK signal.

The latch prevents glitches on the EN signal from propagating to the register's clock pin. When the CLK input of the 2-input AND gate is at logic state 1, any glitching of the EN signal could, without the latch, propagate and corrupt the register clock signal. The latch eliminates this possibility, because it blocks signal changes when the clock is at logic state 1.

In latch-based clock gating, the AND gate blocks unnecessary clock pulses, by maintaining the clock signal's value after the trailing edge. For example, for flip-flops inferred by HDL constructs of positive-edge clocks, the clock gate forces the clock-gated signal to maintain logic state 0 after the falling edge of the clock.

Using an Integrated Clock-Gating Cell

Consider using an integrated clock-gating cell if you are experiencing timing problems caused by the introduction of random logic on the clock line.
Create an integrated clock-gating cell that integrates the various combinational and sequential elements of the clock-gating circuitry into a single cell, which is compiled to gates and located in the technology library.

---

**Setting Pin Attributes for an Integrated Cell**

The clock-gating software requires the pins of your integrated cells to be set with the attributes listed in Table 8-3. Setting some of the pin attributes, such as those for test and observability, is optional.

**Table 8-3 Pin Attributes for Integrated Clock-Gating Cells**

<table>
<thead>
<tr>
<th>Integrated cell pin name</th>
<th>Data direction</th>
<th>Required attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>in</td>
<td>clock_gate_clock_pin</td>
</tr>
<tr>
<td>enable</td>
<td>in</td>
<td>clock_gate_enable_pin</td>
</tr>
<tr>
<td>test_mode or scan_enable</td>
<td>in</td>
<td>clock_gate_test_pin</td>
</tr>
<tr>
<td>observability</td>
<td>out</td>
<td>clock_gate_obs_pin</td>
</tr>
<tr>
<td>enable_clock</td>
<td>out</td>
<td>clock_gate_out_pin</td>
</tr>
</tbody>
</table>

**clock_gate_clock_pin Attribute**

The `clock_gate_clock_pin` attribute identifies an input pin connected to a clock signal.

**Syntax**

```
clock_gate_clock_pin : true | false ;
```

**true | false**

A true value labels the pin as a clock pin. A false value labels the pin as *not* a clock pin.

**Example**

```
clock_gate_clock_pin : true;
```

**clock_gate_enable_pin Simple Attribute**

The `clock_gate_enable_pin` attribute a design containing gated clocks.
The `clock_gate_enable_pin` attribute identifies an input pin connected to an enable signal for nonintegrated clock-gating cells and integrated clock-gating cells.

**Syntax**

```
clock_gate_enable_pin : true | false;
```

`true | false`

A true value labels the input pin of a clock-gating cell connected to an enable signal as the enable pin. A false value does not label the input pin as an enable pin.

**Example**

```
clock_gate_enable_pin : true;
```

For clock-gating cells, you can set this attribute to `true` on only one input port of a 2-input AND, NAND, OR, or NOR gate. If you do so, the other input port is the clock.

**clock_gate_test_pin Attribute**

The `clock_gate_test_pin` attribute identifies an input pin connected to a `test_mode` or `scan_enable` signal.

**Syntax**

```
clock_gate_test_pin : true | false;
```

`true | false`

A true value labels the pin as a test (`test_mode` or `scan_enable`) pin. A false value labels the pin as not a test pin.

**Example**

```
clock_gate_test_pin : true;
```

**clock_gate_obs_pin Attribute**

The `clock_gate_obs_pin` attribute identifies an output pin connected to an observability signal.

**Syntax**

```
clock_gate_obs_pin : true | false;
```

`true | false`

A true value labels the pin as an observability pin. A false value labels the pin as not an observability pin.
Example

clock_gate_obs_pin : true;

clock_gate_out_pin Attribute

The clock_gate_out_pin attribute identifies an output port connected to an enable_clock signal.

Syntax

clock_gate_out_pin : true | false ;

true | false

A true value labels the pin as a clock-gated output (enable_clock) pin. A false value labels the pin as not a clock-gated output pin.

Example

clock_gate_out_pin : true;

Clock-Gating Timing Considerations

The clock gate must not alter the waveform of the clock—other than turning the clock signal on and off.

Figure 8-7 and Figure 8-8 show the relationship of setup and hold times to a clock waveform for AND and OR clock gates.
The setup time specifies how long the clock-gate input (ENL) must be stable before the clock input (CLK) makes a transition to a noncontrolling value. The hold time ensures that the clock-gate input (ENL) is stable for the time you specify after the clock input (CLK) returns to a controlling value. The setup and hold times ensure that the ENL signal is stable for the entire time the CLK signal has a noncontrolling value, which prevents clipping or glitching of the ENCLK clock signal.
Timing Considerations for Integrated Cells

Clock gating requires certain timing arcs on your integrated cell.

For latch-based clock gating,
- Define setup and hold arcs on the enable pins with respect to the same controlling edge of the clock.
- Define combinational arcs from the clock and enable inputs to the output.

For latch-free clock gating,
- Define no-change arcs on the enable pins. These arcs must be no-change arcs, because they are defined with respect to different clock edges.
- Define combinational arcs from the clock and enable inputs to the output.

Set the setup and the hold arcs on the enable pin as specified by the clock_gate_enable_pin attribute with respect to the value entered for the clock_gating_integrated_cell attribute.
For the latch- and flip-flop-based styles, the setup and hold arcs are the conventional type and are set with respect to the same clock edge. However, for the latch-free style, the setup and hold arcs are set with respect to different clock edges and therefore must be specified as no-change arcs. Note that all arcs for integrated cells must be combinational arcs.

### Table 8-4 Values of the `clock_gating_integrated_cell` Attribute for Setup and Hold Arcs

<table>
<thead>
<tr>
<th><code>clock_gating_integrated_cell</code> attribute value</th>
<th>Setup arc</th>
<th>Hold arc</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>latch_posedge</code></td>
<td>rising</td>
<td>rising</td>
</tr>
<tr>
<td><code>latch_negedge</code></td>
<td>falling</td>
<td>falling</td>
</tr>
<tr>
<td><code>none_posedge</code></td>
<td>falling</td>
<td>rising</td>
</tr>
<tr>
<td><code>none_negedge</code></td>
<td>rising</td>
<td>falling</td>
</tr>
<tr>
<td><code>ff_posedge</code></td>
<td>falling</td>
<td>falling</td>
</tr>
<tr>
<td><code>ff_negedge</code></td>
<td>rising</td>
<td>rising</td>
</tr>
</tbody>
</table>

### Integrated Clock-Gating Cell Example

Example 8-7 uses the `latch_posedge_precontrol_obs` value option for the `clock_gating_integrated_cell` attribute.

#### Example 8-7 Integrated Clock-Gating Cell

```plaintext
cell(CGLPCO) {
    area : 1;
    clock_gating_integrated_cell : "latch_posedge_precontrol_obs";
    dont_use : true;
    statetable("CLK EN SE", "IQ") {
        table : ", L L L : - : L , ,
            L L H : - : H , ,
            L H L : - : H , ,
            L H H : - : H , ,
            H - - : - : N ;
    } pin(IQ) {
        direction : internal;
        internal_node : "IQ";
    } pin(EN) {
        direction : input;
        capacitance : 0.017997;
        clock_gate_enable_pin : true;
        timing() {
            timing_type : setup_rising;
            cell_rise(scalar) {
                values( "0.4");
            }
        }
    }
}
```
cell_fall(scalar) {
    values("0.4");
}
related_pin: "CLK";

timing() {
    timing_type: hold_rising;
    cell_rise(scalar) {
        values("0.4");
    }
    cell_fall(scalar) {
        values("0.4");
    }
    related_pin: "CLK";
}
	pin(SE) {
    direction: input;
    capacitance: 0.017997;
    clock_gate_test_pin: true;
}
	pin(CLK) {
    direction: input;
    capacitance: 0.031419;
    clock_gate_clock_pin: true;
    min_pulse_width_low: 0.319;
}
	pin(GCLK) {
    direction: output;
    state_function: "CLK * IQ";
    max_capacitance: 0.500;
    clock_gate_out_pin: true;
    timing() {
        timing_sense: positive_unate;
        cell_rise(scalar) {
            values("0.4");
        }
        rise_transition(scalar) {
            values("0.1");
        }
        cell_fall(scalar) {
            values("0.4");
        }
        fall_transition(scalar) {
            values("0.1");
        }
        related_pin: "EN CLK";
    }
}

internal_power() {
    rise_power(li4X3) {
        index_1("0.0150, 0.0400, 0.1050, 0.3550");
        index_2("0.050, 0.451, 1.501");
        values("0.141, 0.148, 0.256", \
            "0.162, 0.145, 0.234", \
            "0.192, 0.200, 0.284", \
            "0.199, 0.219, 0.297");
    }
    fall_power(li4X3) {
Modeling Electromigration

When high-density current passes through a thin metal wire, the high-energy electrons exert forces upon the atoms that causes electromigration of the atoms. Electromigration can drastically reduce the lifetime of a chip by increasing the resistivity of the metal wire or creating a short circuit between adjacent lines.

Controlling Electromigration

You can control electromigration by establishing an upper boundary for the output toggle rate. You achieve this by annotating cells with electromigration characterization tables representing the net’s maximal toggle rates.

Specifically, do the following to control electromigration:

Use the `em_lut_template` group to name an index template to be used by the `em_max_toggle_rate` group, which you use to set the net’s maximum toggle rates. The `em_lut_template` group has `variable_1`, `variable_2`, `index_1`, and `index_2` attributes.

- Use the `variable_1` and `variable_2` attributes to specify the variables. Both `variable_1` and `variable_2` can take the `input_transition_time` or the `total_output_net_capacitance` value.

- Use the `index_1` attribute to specify the points along the `variable_1` table axis, and the `index_2` attribute to specify the points along the `variable_2` table axis.

The pin-level `electromigration` group contains the `em_max_toggle_rate` group, which you use to specify the maximum toggle rate and the `related_pin` and `related_bus_pins` attributes.
• Use the `related_pin` attribute to identify the input pin with which the electromigration group is associated.

• Use the `related_bus_pins` attribute to identify the bus group of the pin or pins with which the electromigration group is associated.

• To select the template you want the `em_max_toggle_rate` group to use, assign the name of the library-level `em_lut_template` group to the pin-level `em_max_toggle_rate` group.

The `em_max_toggle_rate` group contains the `values` complex attribute; the `related_pin` and `related_bus_pins` simple attributes; and, optionally, the `index_1` and `index_2` complex attributes.

• Use the `values` complex attribute to specify the nets’ maximum toggle rates for every `index_1` breakpoint along the `variable_1` axis if the table is one-dimensional.

• If the table is two-dimensional, use `values` to specify the nets’ maximum toggle rates for all points where the breakpoints of `index_1` intersect with the breakpoints of `index_2`. The value for these points is equal to n\(\text{index}_1\) \(\times n\text{index}_2\), where `index_1` and `index_2` are the size to which the `em_lut_template` group’s `index_1` and `index_2` attributes are set.

• You can also use the `em_max_toggle_rate` group’s optional `index_1` and `index_2` attributes to overwrite the `em_lut_template` group’s `index_1` and `index_2` values.

• State dependency in both lookup tables and polynomials.

Use the optional `em_temp_degradation_factor` at the library level or the cell level to specify the electromigration temperature exponential degradation factor. If this factor is not defined, the nominal temperature electromigration tables are used for all operating temperatures.

---

definition of em_lut_template Group

Use the `em_lut_template` group at the library level to specify the name of the index template to be used by the `em_max_toggle_rate` group, which you use to set the net’s maximum toggle rates to control electromigration.

**Syntax**

```
library (name) {
  em_lut_template(name_string) {
    variable_1: input_transition_time | total_output_net capacitance ;
    variable_2: input_transition_time | total_output_net capacitance ;
    index_1("float, ..., float");
    index_2("float, ..., float");
  }
}
```
variable_1 and variable_2 Simple Attributes

Use variable_1 to assign values to the first dimension and variable_2 to assign values to the second dimension for the templates on electromigration tables. The values can be either input_transition_time or total_output_net_capacitance.

Syntax

variable_1: input_transition_time | total_output_net_capacitance ;
variable_2: input_transition_time | total_output_net_capacitance ;

The value you assign to variable_1 is determined by how the index_1 complex attribute is measured, and the value you assign to variable_2 is determined by how the index_2 complex attribute is measured.

Assign input_transition_time for variable_1 if the complex attribute index_1 is measured with the input net transition time of the pin specified in the related_pin attribute or the pin associated with the electromigration group. Assign total_output_net_capacitance to variable_1 if the complex attribute index_1 is measured with the loading of the output net capacitance of the pin associated with the em_max_toggle_rate group.

Assign input_transition_time for variable_2 if the complex attribute index_2 is measured with the input net transition time of the pin specified in the related_pin attribute, in the related_bus_pins attribute, or in the pin associated with the electromigration group.

Assign total_output_net_capacitance to variable_2 if the complex attribute index_2 is measured with the loading of the output net capacitance of the pin associated with the electromigration group.

Example

variable_1 : total_output_net_capacitance ;
variable_2 : input_transition_time ;

index_1 and index_2 Complex Attributes

You can use the index_1 optional attribute to specify the breakpoints of the first dimension of an electromigration table used to characterize cells for electromigration within the library. You can use the index_2 optional attribute to specify the breakpoints of the second dimension of an electromigration table used to characterize cells for electromigration within the library.
Syntax

index_1 : ("float, ..., float") ;
index_2 : ("float, ..., float") ;

float

Floating-point numbers that identify the maximum toggle rate frequency from 1 to 0 and from 0 to 1.

You can overwrite the values entered for the `em_lut_template` group’s `index_1`, by entering a value for the `em_max_toggle_rate` group’s `index_1`. You can overwrite the values entered for the `em_lut_template` group’s `index_2`, by entering a value for the `em_max_toggle_rate` group’s `index_2`.

The following are the rules for the relationship between variables and indexes:

- If you have `variable_1`, you can have only `index_1`.
- If you have `variable_1` and `variable_2`, you can have `index_1` and `index_2`.
- The value you enter for `variable_1` (used for one-dimensional tables) is determined by how `index_1` is measured. The value you enter for `variable_2` (used for two-dimensional tables) is determined by how `index_2` is measured.

Example

index_1 ("0.0, 5.0, 20.0") ;
index_2 ("0.0, 1.0, 2.0") ;

________________________

electromigration Group

An electromigration group is defined in a pin group, as shown here:

library (name) {
  cell (name) {
    pin (name) {
      electromigration () {
        ... electromigration description ...
      }
    }
  }
}

Simple Attributes

related_pin : "name | name_list" ;/* path dependency */
related_bus_pins : "list of pins" ;/* list of pin names */
related_pin Simple Attribute

This attribute associates the electromigration group with a specific input pin. The input pin’s input transition time is used as a variable in the electromigration lookup table.

If more than one input pin is specified in this attribute, the weighted input transition time of all input pins specified is used to index the electromigration table.

Syntax

related_pin : "name | name_list" ;

name | name_list

Name of input pin or pins.

Example

related_pin : "A B" ;

The pin or pins in the related_pin attribute denote the path dependency for the electromigration group. A particular electromigration group is accessed if the input pin or pins named in the related_pin attribute cause the corresponding output pin named in the pin group to toggle. All functionally related pins must be specified in a related_pin attribute if two-dimensional tables are being used.

Group Statements

dm_max_toggle_rate (em_template_name) {}

These attributes and groups are described in the following sections.

related_bus_pins Simple Attribute

This attribute associates the electromigration group with the input pin or pins of a specific bus group. The input pin’s input transition time is used as a variable in the electromigration lookup table.

If more than one input pin is specified in this attribute, the weighted input transition time of all input pins specified is used to index the electromigration table.

Syntax

related_bus_pins : "name1 [name2 name3 ... ]" ;

Example

related_bus_pins : "A" ;

The pin or pins in the related_bus_pins attribute denote the path dependency for the electromigration group. A particular electromigration group is accessed if the input pin or pins named in the related_bus_pins attribute cause the corresponding output pin
named in the pin group to toggle. All functionally related pins must be specified in a related_bus_pins attribute if two-dimensional tables are being used.

em_max_toggle_rate Group

The em_max_toggle_rate group is a pin-level group that is defined within the electromigration pin group. The toggle rate is the number of toggles per unit of time where the unit of time is specified by the library-level time_unit attribute.

```
library (name) {
  cell (name) {
    pin (name) {
      electromigration () {
        em_max_toggle_rate(em_template_name) {
          ... em_max_toggle_rate description ...
        }
      }
    }
  }
}
```

Simple Attribute

current_type

The current_type attribute is defined in the following section.

current_type Simple Attribute

The optional current_type attribute specifies the type of current for the em_max_toggle_rate lookup table. Valid values are average, rms, and peak.

Syntax

current_type: average | rms | peak ;

Example

current_type: average ;

Complex Attributes

```
index_1 ("float, ..., float") ; /*this attribute is optional*/
index_2 ("float, ..., float") ; /*this attribute is optional*/
values ("float, ..., float ") ;
```

These attributes are defined in the following sections.
Index_1 and Index_2 Complex Attributes

You can use the index_1 optional attribute to specify the breakpoints of the first dimension of an electromigration table to characterize cells for electromigration within the library. You can use the index_2 optional attribute to specify breakpoints of the second dimension of an electromigration table used to characterize cells for electromigration within the library.

You can overwrite the values entered for the em_lut_template group’s index_1, by entering values for the em_max_toggle_rate group’s index_1. You can overwrite the values entered for the em_max_toggle_rate group’s index_2, by entering values for the em_lut_template group’s index_2.

Syntax

index_1 ("float, ..., float") ; /*this attribute is optional*/
index_2 ("float, ..., float") ; /*this attribute is optional*/

float

Floating-point numbers that identify the maximum toggle rate frequency.

Example

index_1 ("0.0, 5.0, 20.0") ;
index_2 ("0.0, 1.0, 2.0") ;

values Complex Attribute

This complex attribute is used to specify the net’s maximum toggle rates.

This attribute can be a list of nindex_1 positive floating-point numbers if the table is one-dimensional.

This attribute can also be nindex_1 x nindex_2 positive floating-point numbers if the table is two-dimensional, where nindex_1 is the size of index_1 and nindex_2 is the size of index_2 that is specified for these two indexes in the em_max_toggle_rate group or in the em_lut_template group.

Syntax

values("float, ..., float") ;

float

Floating-point numbers that identify the maximum toggle rate frequency.

Example (One-Dimensional Table)

values : ("1.5, 1.0, 0.5") ;
Example (Two-Dimensional Table)
values : ("2.0, 1.0, 0.5", "1.5, 0.75, 0.33","1.0, 0.5, 0.15",) ;

em_temp_degradation_factor Simple Attribute
The em_temp_degradation_factor attribute specifies the electromigration exponential degradation factor.

Syntax
em_temp_degradation_factor : value float ;

value
A floating-point number in centigrade units consistent with other temperature specifications throughout the library.

Example
em_temp_degradation_factor : 40.0 ;
Advanced low-power design methodologies such as multivoltage and multithreshold-CMOS require special library cells. These cells include power management attributes to drive implementation tools during library cell selection. This chapter describes the power and ground (PG) pin syntax and provides modeling examples of the special cells, such as the level-shifter, isolation, switch, and retention cells.

Note:
To model very deep submicron (VDSM) technology libraries, you must use the PG pin syntax.

This chapter includes the following sections:

• Power and Ground (PG) Pins
• PG Pin Power Mode Modeling
• Feedthrough Signal Pin Modeling
• Silicon-on-Insulator (SOI) Cell Modeling
• Level-Shifter Cells in a Multivoltage Design
• Isolation Cell Modeling
• Switch Cell Modeling
• Retention Cell Modeling
• Always-On Cell Modeling
• Macro Cell Modeling
• Modeling Antenna Diodes
Power and Ground (PG) Pins

The syntax supports power and ground (PG) library pins. A power pin is a current source pin, and a ground pin is a current sink pin. This section provides an overview of PG pins.

Example Libraries With Multiple Power Supplies

Example 9-1 shows a library with multiple power supplies defined at the library, cell, and pin levels.

Example 9-1  Library With a power_supply Group and Power Supplies Defined at the Library, Cell, and Pin Levels

library(pow) {
   ...
   voltage_unit : "1V";
   power_supply() {
      default_power_rail : VDD0;
      power_rail(VDD1, 5.0);
      power_rail(VDD2, 3.3);
   }
   operating_conditions(WCCOM) {
      process : 1.5 ;
      temperature : 70 ;
      voltage : 4.75 ;
      tree_type : "worst_case_tree" ;
      power_rail(VDD1, 4.8);
      power_rail(VDD2, 2.9);
   }
   cell(IBUF1) {
      ...
      rail_connection(PV1, VDD1);
      rail_connection(PV2, VDD2);
      pin(A) {
         direction : input;
         ...
         input_signal_level : VDD1;
      }
      pin(EN) {
         direction : input;
         ...
         input_signal_level : VDD1;
      }
      pin(Z) {
         direction : output;
         output_signal_level : VDD2;
         ...
      }
      pin(Z1) {
         direction : inout;
Example 9-2 Library Defining All Power Supplies With Nominal Operating Conditions

```lisp
power_supply() {
  default_power_rail : VDD0;
  power_rail : (VDD1, 5.0);
  power_rail : (VDD2, 3.3 );
}
operating_conditions (MPSCOM) {
  process : 1.5;
  temperature : 70;
  voltage : 4.75;
  tree_type : worst_case_tree;
  power_rail : (VDD1, 5.8);
  power_rail : (VDD2, 3.3);
}
```

Example 9-3 Cell With Default Power Supply

```lisp
cell(with_no_power_supply) {
  area : 10;
  pin (A) {
    direction: input;
    capacitance: 1.0;
  }
  pin (Z) {
    direction: output;
    function : "!A"
  }
} /* end cell */
```

Example 9-4 Cell With One Power Supply

```lisp
cell(with_one_power_supply) {
  area : 10;
  rail_connection (PV1, VDD1);
  pin (A) {
    direction: input;
    capacitance: 1.0;
  }
  pin (Z) {
    direction: output;
    function : "!A"
  }
} /* end cell */
```
Example 9-5  Cell With Two Power Supplies

```liberty
cell(with_2_power_supplies) {
    area : 0;
    rail_connection (PV1, VDD1);
    rail_connection (PV2, VDD2);
    pin (A) {
        direction: input;
        capacitance: 1.0;
        input_signal_level: VDD1;
    }
    pin (EB) {
        direction: input;
        capacitance: 1.0;
        input_signal_level: VDD1;
    }
    pin (Z) {
        direction: output;
        function: "A";
        three_state: "EB";
        output_signal_level: VDD2;
    }
} /* end cell */
```

Example 9-6  Cell With Two Power Supplies and a Bidirectional Pin

```liberty
cell(bidir_with_2_power_supplies) {
    area : 0;
    rail_connection (PV1, VDD1);
    rail_connection (PV2, VDD2);
    pin(PAD) {
        direction: inout;
        function: "A";
        three_state: "EN";
        input_signal_level: VDD1;
        output_signal_level: VDD2;
    }
    pin(A) {
        direction: input;
        capacitance: 3;
        input_signal_level: VDD1;
    }
    pin(EN) {
        direction: input;
        capacitance: 3;
        input_signal_level: VDD1;
    }
    pin(X) {
        direction: output;
        function: "PAD";
        output_signal_level: VDD2;
    }
```
Partial PG Pin Cell Modeling

Partial PG pin cells are cells that have only power pins, only ground pins, or do not have power or ground PG pins.

Table 9-1  Categories of Partial PG Pin Cells

<table>
<thead>
<tr>
<th>Partial PG Pin Cell</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells with one power pin and one ground pin</td>
<td>These cells are defined as having</td>
</tr>
<tr>
<td></td>
<td>• One or more primary power PG pins</td>
</tr>
<tr>
<td></td>
<td>• One or more primary ground PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, backup or internal power PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, backup or internal ground PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, nwell bias PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, pwell bias PG pins</td>
</tr>
<tr>
<td>Cells with one power pin and no ground pins</td>
<td>These cells are defined as having</td>
</tr>
<tr>
<td></td>
<td>• One or more primary power PG pins</td>
</tr>
<tr>
<td></td>
<td>• No primary ground PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, backup or internal power PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, backup or internal ground PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, nwell bias PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, pwell bias PG pins</td>
</tr>
<tr>
<td>Cells with no power pins and one ground pin</td>
<td>These cells are defined as having</td>
</tr>
<tr>
<td></td>
<td>• No primary power PG pins</td>
</tr>
<tr>
<td></td>
<td>• At least one primary ground PG pin</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, backup or internal power PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, backup or internal ground PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, nwell bias PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, pwell bias PG pins</td>
</tr>
</tbody>
</table>
The following types of partial PG pin cells are acceptable with certain conditions.

- **ETM cells**
  
  ETM cells do not need a pair of PG pins. A cell is identified as an ETM cell if it has the `interface_timing` or `timing_model_type` attribute.

- **Black box cells**
  
  A black box cell with no timing, noise, or power information does not need at least one power pin and one ground PG pin.

- **Metal fills and antenna cells**
  
  Black box cells without functions do not need at least power pin and one ground pin.

- **Cells without signal pins**
  
  Cells without signal pins do not need at least one power pin and one ground PG pin.

- **Load cells**
  
  Cells without inout or output signal pins require either a power or a ground pin, but it is not necessary to have both.

- **Tied-off cells and extensions**
  
  The following types of cells can be specified with one power pin and no ground PG pins:
  
  - Cells with at least one inout or output pin with the `function` attribute set to 1.
  - Cells with a pull-up signal, for example with the `driver_type` attribute set to `pull_up` or `pull_up_function`.
  - Cells with a `resistive_1` signal, for example with the `driver_type` attribute set to `resistive_1` or `resistive_1_function`.

---

### Special Partial PG Pin Cells

The following types of partial PG pin cells are acceptable with certain conditions.

<table>
<thead>
<tr>
<th>Partial PG Pin Cell</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells with no power pins or ground pins</td>
<td>These cells are defined as having</td>
</tr>
<tr>
<td></td>
<td>• No primary power PG pins</td>
</tr>
<tr>
<td></td>
<td>• No primary ground PG pins</td>
</tr>
<tr>
<td></td>
<td>• No backup or internal power PG pin</td>
</tr>
<tr>
<td></td>
<td>• No backup or internal ground PG pin</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, nwell bias PG pins</td>
</tr>
<tr>
<td></td>
<td>• Zero, or at least one, pwell bias PG pins</td>
</tr>
</tbody>
</table>

Table 9-1 Categories of Partial PG Pin Cells

---

*Cells with no power pins or ground pins*

These cells are defined as having:

- No primary power PG pins
- No primary ground PG pins
- No backup or internal power PG pin
- No backup or internal ground PG pin
- Zero, or at least one, nwell bias PG pins
- Zero, or at least one, pwell bias PG pins
The following types of cells can be specified with no power pins and one ground PG pin:

- Cells with at least one inout or output pin with the function attribute set to 0.
- Cells with a pull-down signal, for example with the driver_type attribute set to pull_down or pull_down_function.
- Cells with a resistive_0 signal, for example with the driver_type attribute set to resistive_0 or resistive_0_function.

**Supported Attributes**

Cells with at least one power pin and no ground pins, cells with no power pins and at least one ground pin, and cells with no power pins or ground pins support the following attributes:

- To prevent a cell from being automatically inserted into the netlist, specify the dont_touch or the dont_use attribute. The dont_touch attribute set to true indicates that all instances of the cell must remain in the network. The dont_use attribute set to true indicates that a cell must not be added to a design during optimization.

- Cells with partial PG pins are reported using the following attributes:
  - 1p0g
    Reports cells with at least one power pin and no ground PG pins.
  - 0p1g
    Reports cells with no power pins and at least one ground PG pin.
  - 0p0g
    Reports cells with no power pins or ground PG pins.

**Partial PG Pin Cell Example**

Example 9-7 shows a cell with only one primary ground pin.

*Example 9-7  Partial PG Pin Cell With Only One Primary Ground Pin*

cell (PULLDOWN) {
  pg_pin ( VSS ) {
    voltage_name : VSS;
    pg_type : primary_ground;
  }

  area : 1.0;
  dont_touch : true;
  dont_use : true;

  pin (X) {
    related_ground_pin : VSS;
  }
}
direction : output;
function : "0";
three_state : "!A";
max_capacitance : 0.19;
timing() {
    related_pin : "A";
    timing_type : three_state_enable;
    cell_rise(scalar) { values ("0.1");}
    rise_transition(scalar) { values ("0.1");}
    cell_fall(scalar) { values ("0.1");}
    fall_transition(scalar) { values ("0.1");}
}
timing() {
    related_pin : "A";
    timing_type : three_state_disable;
    cell_rise(scalar) { values ("0.1");}
    rise_transition(scalar) { values ("0.1");}
    cell_fall(scalar) { values ("0.1");}
    fall_transition(scalar) { values ("0.1");}
}

pin (A) {
    related_ground_pin : VSS;
direction : input;
capacitance : 0.1;
rise_capacitance : 0.1;
rise_capacitance_range (0.1, 0.2);
fall_capacitance : 0.1;
fall_capacitance_range (0.1, 0.2);
}

---

**PG Pin Syntax**

The power and ground pin syntax for generic cells is as follows:

```plaintext
library(library_name) {
...
  voltage_map(voltage_name, voltage_value);
  voltage_map(voltage_name, voltage_value);
...
  operating_conditions(oc_name) {
    ...
    voltage : value;
    ...
  }
...
  default_operating_conditions : oc_name;
  cell(cell_name) {
    pg_pin (pg_pin_name_pl) {
      voltage_name : voltage_name_pl;
    }
  }
}
```
pg_type : type_value;
}
p_g (pg_pin_name_g1) {
  voltage_name : voltage_name_g1;
  pg_type : type_value;
}
p_g (pg_pin_name_p2) {
  voltage_name : voltage_name_p2;
  pg_type : type_value;
}
p_g (pg_pin_name_g2) {
  voltage_name : voltage_name_g2;
  pg_type : type_value;
}
...
leakage_power() {
  related_pg_pin : pg_pin_name_p1;
  ...
}
...
pin (pin_name1) {
  direction : input | inout;
  related_power_pin : pg_pin_name_p1;
  related_ground_pin : pg_pin_name_g1;
  ...
}
...
pin (pin_name2) {
  direction : inout | output;
  power_down_function : (!pg_pin_name_p1 + !pg_pin_name_p2 +
  !pg_pin_name_g1 + !pg_pin_name_g2);
  related_power_pin : pg_pin_name_p2;
  related_ground_pin : pg_pin_name_g2;
  output_signal_voltage_low : float;
  output_signal_voltage_high : float;
  timing() {
    ...
    output_signal_voltage_low : float;
    output_signal_voltage_high : float;
  }
  internal_power() {
    related_pg_pin : pg_pin_name_p2;
    ...
  } /* end internal_power group */
  ...
}/* end pin group*/
... /* end cell group*/
}/* end library group*/
Library-Level Attributes

This section describes library-level attributes.

voltage_map Complex Attribute

The `voltage_map` attribute associates the voltage name with the relative voltage values. These voltage names are referenced by the `pg_pin` groups defined at the cell level. When specified in a library, this attribute identifies the library as a power and ground pin library. At least one voltage map in the library should have a value of 0, which becomes the reference value to which other voltage map values relate.

default_operating_conditions Simple Attribute

The `default_operating_conditions` attribute specifies the name of the default `operating_conditions` group in the library, which helps to identify the operating condition process, voltage, and temperature (PVT) points that are used during library characterization.

Cell-Level Attributes

This section describes cell-level attributes for `pg_pin` groups.

pg_pin Group

The `pg_pin` groups are used to represent the power and ground pins of a cell. Library cells can have multiple power and ground pins. The `pg_pin` groups are mandatory for each cell using the power and ground pin syntax, and a cell must have at least one `primary_power` power pin and at least one `primary_ground` ground pin.

is_pad Simple Attribute

The `is_pad` attribute identifies a pad pin on any I/O cell. The valid values are `true` and `false`. You can also specify the `is_pad` attribute on a PG pin. If the cell-level `pad_cell` attribute is specified on an I/O cell, you must set the `is_pad` attribute to `true` in either a `pg_pin` group or on a signal pin for that cell.

voltage_name Simple Attribute

The `voltage_name` string attribute is mandatory in all `pg_pin` groups except for a level-shifter cell not powered by the switching power domains. The `voltage_name` attribute specifies the associated voltage name of the power and ground pin defined using the `voltage_map` complex attribute at the library level.
Note:
To allow implementation tools to include a level-shifter cell not powered by the switching domains in any other power domain, the voltage value of the PG pin with the std_cell_main_rail attribute is ignored. So for this cell, the voltage_name attribute is optional in the pg_pin group that has the std_cell_main_rail attribute and not connected to the signal pins.

**pg_type Simple Attribute**

The pg_type attribute, optional in pg_pin groups, specifies the type of power and ground pin. The pg_type attribute can have the following values: primary_power, primary_ground, backup_power, backup_ground, internal_power, internal_ground, pwell, nwell, deepnwell and deeppwell.

The pg_type attribute also supports substrate-bias modeling. Substrate bias is a technique in which a bias voltage is varied on the substrate terminal of a CMOS device. This increases the threshold voltage, the voltage required by the transistor to switch, which helps reduce transistor power leakage. The pg_type attribute provides the pwell, nwell, deepnwell and deeppwell values to support substrate-bias modeling. The pwell and nwell values specify regular wells, and deepnwell and deeppwell specify isolation wells.

Table 9-2 describes the pg_type values.

*Table 9-2  pg_type Values*

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary_power</td>
<td>Specifies that pg_pin is a primary power source (the default). If the pg_type attribute is not specified, primary_power is the pg_type value.</td>
</tr>
<tr>
<td>primary_ground</td>
<td>Specifies that pg_pin is a primary ground source.</td>
</tr>
<tr>
<td>backup_power</td>
<td>Specifies that pg_pin is a backup (secondary) power source (for retention registers, always-on logic, and so on).</td>
</tr>
<tr>
<td>backup_ground</td>
<td>Specifies that pg_pin is a backup (secondary) ground source (for retention registers, always-on logic, and so on).</td>
</tr>
<tr>
<td>internal_power</td>
<td>Specifies that pg_pin is an internal power source for switch cells.</td>
</tr>
<tr>
<td>internal_ground</td>
<td>Specifies that pg_pin is an internal ground source for switch cells.</td>
</tr>
<tr>
<td>pwell</td>
<td>Specifies regular p-wells for substrate-bias modeling.</td>
</tr>
</tbody>
</table>
physical_connection Simple Attribute

The physical_connection attribute can have the following values: device_layer and routing_pin. The device_layer value specifies that the bias connection is physically external to the cell. In this case, the library provides biasing tap cells that connect through the device layers. The routing_pin value specifies that the bias connection is inside a cell and is exported as a physical geometry and a routing pin. Macros with pin access generally use the routing_pin value if the cell has bias pins with geometry that is visible in the physical view.

related_bias_pin Attribute

The related_bias_pin attribute defines all bias pins associated with a power or ground pin within a cell. The related_bias_pin attribute is required only when the attribute is declared in a pin group but it does not specify a complete relationship between the bias pin and power and ground pin for a library cell.

The related_bias_pin attribute also defines all bias pins associated with a signal pin. To associate substrate-bias pins to signal pins, use the related_bias_pin attribute to specify one of the following pg_type values: pwell, nwell, deepwell, deepnwell. Figure 9-1 shows transistors with p-well and n-well substrate-bias pins.

Figure 9-1 Transistors With p-well and n-well Substrate-Bias Pins

Table 9-2 pg_type Values (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nwell</td>
<td>Specifies regular n-wells for substrate-bias modeling.</td>
</tr>
<tr>
<td>deepnwell</td>
<td>Specifies isolation n-wells for substrate-bias modeling.</td>
</tr>
<tr>
<td>deeppwell</td>
<td>Specifies isolation p-wells for substrate-bias modeling.</td>
</tr>
</tbody>
</table>
**user_pg_type Simple Attribute**

The `user_pg_type` optional attribute allows you to customize the type of power and ground pin. It accepts any string value. The following example shows a `pg_pin` library with the `user_pg_type` attribute specified.

---

**Pin-Level Attributes**

This section describes pin-level attributes.

**power_down_function Attribute**

The `power_down_function` string attribute specifies the Boolean condition under which the cell’s output pin is switched off by the state of the power and ground pins (when the cell is in off mode due to the external power pin states).

You specify the `power_down_function` attribute for combinational and sequential cells. For simple and complex sequential cells, `power_down_function` also determines the condition of the cell’s internal state.

For more information about using the `power_down_function` attribute for sequential cells, see Chapter 5, “Defining Sequential Cells”.

**related_power_pin and related_ground_pin Attributes**

The `related_power_pin` and `related_ground_pin` attributes associate a predefined power and ground pin with the signal pin, in which they are defined. This behavior only applies to standard cells. For special cells, explicitly specify this relationship.

The `pg_pin` groups are mandatory for each cell. Because a cell must have at least one `primary_power` and at least one `primary_ground` pin, a default `related_power_pin` and `related_ground_pin` always exists in any cell.

**output_signal_level_low and output_signal_level_high Attributes**

You can define the `output_signal_level_low` and `output_signal_level_high` attributes for output and inout pins. The regular signal swings are derived for regular cells using the `related_power_pin` and `related_ground_pin` specifications.

You can also define the `output_signal_level_low` and `output_signal_level_high` attributes in timing arcs. See “`output_signal_level_low` and `output_signal_level_high` Attributes” on page 9-15.
input_signal_level_low and input_signal_level_high Attributes

The `input_signal_level_low` and `input_signal_level_high` attributes can be defined at the pin level for the input pins. The regular signal swings are derived for regular cells using the `related_power_pin` and `related_ground_pin` specifications.

related_pg_pin Attribute

The `related_pg_pin` attribute is used to associate a power and ground pin with leakage power and internal power tables. (The leakage power and internal power tables must be associated with the cell's power and ground pins.)

In the absence of a `related_pg_pin` attribute, the `internal_power` and `leakage_power` specifications apply to the whole cell (cell-specific power specification).

Timing-Level Attributes

This section describes timing-level attributes.

output_signal_level_low and output_signal_level_high Attributes

The optional `output_signal_level_low` and `output_signal_level_high` attributes specify the actual output voltages of an output pin after a transition through a timing arc.

The `output_signal_level_low` attribute specifies the minimum voltage after a falling transition to state 0.

The `output_signal_level_high` attribute specifies the maximum voltage value after a rising transition to state 1.

Define the `output_signal_level_low` and `output_signal_level_high` attributes in the `timing` group when the following occur together:

- The cell output exhibits partial voltage swing (and not rail-to-rail swing).
- The voltages are different in different timing arcs.

Specify the voltages with respect to the `voltage_unit` defined in the library. The voltage values that you specify in the `timing` group override the voltages specified under the `pin` group. The attributes do not have a default.
Specifying Delay and Slew Attributes in Voltage Range

When partial voltage swing is defined in a pin or a timing group, the nonlinear delay model (NLDM) data in that group is for the partial swing. You must apply the following threshold and trip point attributes only in the voltage range from the output_signal_level_low value to the output_signal_level_high value:

- input_threshold_pct_rise
- output_threshold_pct_rise
- input_threshold_pct_fall
- output_threshold_pct_fall
- slew_lower_threshold_pct_rise
- slew_upper_threshold_pct_rise
- slew_lower_threshold_pct_fall
- slew_upper_threshold_pct_fall

The following example shows a library description with the output_signal_level_low and output_signal_level_high attributes defined in both the pin and the timing groups.

```lib1 {
  ...
  cell (cell1) {
    ...
    pin(pin1) {
      direction: output;
      output_signal_level_low: 0.15;
      output_signal_level_high: 0.75;
      timing() {
        ...
        related_pin : "CLKIN";
        timing_type : rising_edge;
        output_signal_level_low: 0.1;
        output_signal_level_high: 0.7;
        ...
      } /* end of timing */
      ...
    } /* end of pin */
    ...
  } /* end of cell */
  ...
} /* end of library */```
Standard Cell With One Power and Ground Pin Example

Figure 9-2 shows a standard cell with a power and ground pin. The figure is followed by an example.

Figure 9-2 Standard Cell Buffer Schematic

library(standard_cell_library_example) {
    voltage_map(VDD, 1.0);
    voltage_map(VSS, 0.0);

    operating_conditions(XYZ) {
        voltage : 1.0;
        ...
    }
    default_operating_conditions : XYZ;

    cell(BUF) {
        pg_pin(VDD) {
            voltage_name : VDD;
            pg_type : primary_power;
        }
        pg_pin(VSS) {
            voltage_name : VSS;
            pg_type : primary_ground;
        }

        leakage_power() {
            related_pg_pin : VDD;
            when : "!A";
            value : 1.5;
        }
    }
}
pin(A) {
    related_power_pin : VDD;
    related_ground_pin : VSS;
}

pin(Y) {
    direction : output;
    power_down_function : "!VDD + VSS";
    related_power_pin : VDD;
    related_ground_pin : VSS;

timing() {
    related_pin : A;
    cell_rise(template) {
        ...
    }
    cell_fall(template) {
        ...
    }
    rise_transition(template) {
        ...
    }
    fall_transition(template) {
        ...
    }
}

internal_power() {
    related_pin : A;
    related_pg_pin : VDD;
    ...
}

}  /* end pin group*/

}  /* end cell group*/

}  /* end library group*/
Inverter With Substrate-Bias Pins Example

Figure 9-3 shows an inverter with substrate-bias pins. The figure is followed by an example.

Figure 9-3  Inverter With Substrate-Bias Pins

```liberty
library(low_power_cells) {

delay_model : table_lookup;

  /* unit attributes */
  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1mA";
  pulling_resistance_unit : "1kohm";
  leakage_power_unit : "1pW";
  capacitive_load_unit (1.0,pf);

  voltage_map(VDD, 0.8); /* primary power */
  voltage_map(VSS, 0.0); /* primary ground */
  voltage_map(VNW, 0.8); /* bias power */
  voltage_map(VPW, 0.0); /* bias ground */

  /* operation conditions */
  operating_conditions(XYZ) {
    process: 1;
    temperature: 125;
    voltage: 0.8;
    tree_type: balanced_tree
  }
  default_operating_conditions : XYZ;
```
/* threshold definitions */
slew_lower_threshold_pct_fall : 30.0;
slew_upper_threshold_pct_fall : 70.0;
slew_lower_threshold_pct_rise : 30.0;
slew_upper_threshold_pct_rise : 70.0;
input_threshold_pct_fall      : 50.0;
input_threshold_pct_rise      : 50.0;
output_threshold_pct_fall     : 50.0;
output_threshold_pct_rise     : 50.0;

/* default attributes */
default_cell_leakage_power: 0.0;
default_fanout_load: 1.0;
default_output_pin_cap: 0.0;
default_inout_pin_cap: 0.1;
default_input_pin_cap: 0.1;
default_max_transition: 1.0;

cell(std_cell_inv) {
    cell_footprint : inv;
    area : 1.0;
    pg_pin(VDD) {
        voltage_name : VDD;
        pg_type : primary_power;
        related_bias_pin : "VNW";
    }
    pg_pin(VSS) {
        voltage_name : VSS;
        pg_type : primary_ground;
        related_bias_pin : "VPW";
    }
    pg_pin(VNW) {
        voltage_name : VNW;
        pg_type : nwell;
        physical_connection : device_layer;
    }
    pg_pin(VPW) {
        voltage_name : VPW;
        pg_type : pwell;
        physical_connection : device_layer;
    }

    pin(A) {
        direction : input;
        capacitance : 1.0;
        related_power_pin : VDD;
        related_ground_pin : VSS;
        related_bias_pin : "VPW VNW";
    }
    pin(Y) {
        direction : output;
        function : "A'";
        related_power_pin : VDD;
    }
related_ground_pin : VSS;
related_bias_pin : "VPW VNW";
power_down_function : "!VDD + VSS + VNW' + VPW";
internal_power() {
related_pg_pin : VDD;
related_pin : "A";
rise_power(scalar) { values ( "1.0" );}
fall_power(scalar) { values ( "1.0" );}
}
timing() {
related_pin : "A";
timing_sense : positive_unate;
cell_rise(scalar) { values ( "0.1" );}
rise_transition(scalar) { values ( "0.1" );}
cell_fall(scalar) { values ( "0.1" );}
fall_transition(scalar) { values ( "0.1" );}
}
max_capacitance : 0.1;
}
cell_leakage_power : 1.0;
leakage_power() {
when : "!A";
value : 1.5;
}
leakage_power() {
when : "A";
value : 0.5;
}

---

Insulated Bias Modeling

Example 9-8 is a .lib description of a low-power library with the following two cells:

- An always-on cell with an insulated bias PG pin
- A cell where the bias PG pin is tied to the primary power supply
Figure 9-4 shows the schematics of these cells.

**Figure 9-4 An Always-On Cell With Insulated Bias and Cell Where the Bias PG pin is Tied to Primary Power**

Example 9-8 A Low-Power Library Using the `is_insulated` and `tied_to` Attributes

```liberty
library (mylib) {
  delay_model : table_lookup;

  /* library unit and slew attributes */
  voltage_map(VDD, 1.0);   /* Primary Power */
  voltage_map(VSS, 0.0);   /* Primary Ground */
  voltage_map(VDDB, 1.0);  /* Backup Power */
  voltage_map(VSSB, 0.0);  /* Backup Ground */
  voltage_map(VPW, 1.0);   /* nwell */
  voltage_map(VNW, 0.0);   /* pwell */
  voltage_map(VPW_INS, 1.0); /* insulated nwell */
  voltage_map(VNW_INS, 0.0); /* insulated pwell */
```
/* operation conditions */
nom_process     : 1.0;
nom_temperature : 25;
nom_voltage     : 1.0;
operating_conditions(XYZ) {
  process     : 1.0;
  temperature : 25;
  voltage     : 1.0;
  tree_type   : balanced_tree
}
default_operating_conditions : XYZ;

/* AO cell with insulated bias pg_pins */
cell(AO_with_insulated_bias) {
  area : 1.0;
...
  pg_pin(VDD) {
    voltage_name : VDD;
    pg_type : primary_power;
    related_bias_pin : VPW;
  }
  pg_pin(VSS) {
    voltage_name : VSS;
    pg_type : primary_ground;
    related_bias_pin : VNW;
  }
  pg_pin(VDDB) {
    voltage_name : VDDB;
    pg_type : backup_power;
    related_bias_pin : VPW_INS;
  }
  pg_pin(VSSB) {
    voltage_name : VSSB;
    pg_type : backup_ground;
    related_bias_pin : VNW_INS;
  }
  pg_pin(VPW) {
    voltage_name : VPW;
    pg_type : nwell;
    direction : inout;
    physical_connection : device_layer;
  }
  pg_pin(VNW) {
    voltage_name : VNW;
    pg_type : pwell;
    direction : inout;
    physical_connection : device_layer;
  }
  pg_pin(VNW_INS) {
    pg_type : pwell;
    voltage_name : VNW_INS;
    is_insulated : true;
  }

tied_to : VSSB;
direction : internal;
}
pin(A) {
direction : input;
related_power_pin : VDDB;
related_ground_pin : VSSB;
...
}
pin(Y) {
direction : output;
function : "A";
related_power_pin : VDDB;
related_ground_pin : VSSB;
power_down_function : "!VDDB + !VPW_INS + VSSB + VNW_INS";
...
} /* end pin group */
} /* end cell group */

/* cell with bias pg_pin tied to Primary Power */
cell(cell_with_bias_pg_pin_tied_to_power) {
area : 1.0;
...
pin(VDD) {
voltage_name : VDD;
pg_type : primary_power;
related_bias_pin : VPW;
}
pin(VSS) {
voltage_name : VSS;
pg_type : primary_ground;
related_bias_pin : VNW;
}
pin(VDDB) {
voltage_name : VDDB;
pg_type : backup_power;
related_bias_pin : VPW_INS;
}
pin(VPW) {
voltage_name : VPW;
pg_type : nwell;
direction : inout;
tied_to : VDD;
}
pin(VNW) {
voltage_name : VNW;
In complex library cells such as retention and macro cells, power pins can have more power states (or PG modes) apart from the regular ON and OFF states. The following syntax allows you to model such power states of PG pins and the relationship between these states, so that complex power states or modes can be captured.

The syntax enables you to

- Extend power supply state and voltage range definitions
- Define system power states as function of power supplies
- Define transitions between system power states
- Associate a PG mode to each conditional timing, power, and noise group

**Syntax**

```plaintext
library (library_name) {
  voltage_map (voltage_name, voltage_value);
  voltage_state_range_list (voltage_name) {
    voltage_state_range(pg_state, nom_voltage);
    voltage_state_range(pg_state, min_voltage, max_voltage);
    voltage_state_range(pg_state, min_voltage, nom_voltage, max_voltage);
    ...
  }
}
```
Library-Level Groups and Attributes

Use the following syntax to model power and ground (PG) modes other than the regular on and off states.
voltage_state_range_list Group

The voltage_state_range_list group defines the voltage range of all the states for a specified power rail. It only applies to a power rail with multiple states. The name of the group (voltage_name) is a power rail name, which must also be defined in the same library using the voltage_map attribute. Each voltage_name can have only one corresponding voltage_state_range_list group.

voltage_state_range Attribute

The voltage_state_range attribute defines the corner-independent operating voltages for the state, pg_state, of the power rail. Specify this attribute in the voltage_state_range_list group.

Syntax

voltage_state_range_list (voltage_name) {
    voltage_state_range(pg_state, nom_voltage);
    voltage_state_range(pg_state, min_voltage, max_voltage);
    voltage_state_range(pg_state, min_voltage, nom_voltage, max_voltage);
}

• pg_state is the state name of the power rail
• The value can be a nominal voltage, a pair specifying the minimum and maximum voltages, or a triplet specifying the minimum, nominal and maximum voltages.
  • min_voltage and max_voltage are floating-point numbers for the minimum and maximum operating voltage of the state.
  • nom_voltage is a floating-point number and the default operating voltage that applies to all designs. The instance-specific nominal voltage specification at the top-design level overrides the default nominal voltage.
  • These values represent the nominal voltage range and do not include the power supply uncertainty or tolerance.
• For each pg_state defined in the voltage_state_range_list group:
  • min_voltage must be less than or equal to max_voltage or nom_voltage.
  • nom_voltage must be less than or equal to max_voltage.
  • Different pg_state can have overlapped operating voltage range.

Example

voltage_map (VDD, 0.8);
voltage_state_range_list(VDD) {
    voltage_state_range(normal, 0.81, 0.9, 0.99);
    voltage_state_range(under_drive, 0.665, 0.77);
    voltage_state_range(on, 0.7);
Cell-Level Groups and Attributes

Use the following syntax to define the system power states as a function of the power supply states, and the legality of transition between these power states.

**pg_setting_definition Group**

The `pg_setting_definition` group defines the following:

- The system power states (`pg_setting_value` group).
- The legality of transitions (`pg_setting_transition` group) between the system power states.
- The default power state of the cell.
- The legality of undefined transitions.

`psd_name` is the name of the `pg_setting_definition` group. The system power states must be mutually exclusive.

**Syntax**

```
pg_setting_definition(psd_name) {
  ...
}
```

`psd_name` is the name of the `pg_setting_definition` group. The system power states must be mutually exclusive.

**Example**

```
pg_setting_definition(psd) {
  ...
}
```

**pg_setting_value Group**

The `pg_setting_value` group defines a system power state named `psv_name`. Specify this group in the `pg_setting_definition` group.

- You must define all the legal power states. Undefined power states are considered illegal.
- A legal power state is a state that is expected to occur during normal operation.
- An illegal power state is a state that is not expected to occur during normal operation.
Liberty User Guide, Volume 1

Chapter 9: Advanced Low-Power Modeling

PG Pin Power Mode Modeling

Syntax

```plaintext
pg_setting_value (psv_name) {
    pg_pin_active_state (pg_pin, pg_state);
    pg_pin_condition : Boolean expression of pg_pin and signal pin;
    pg_setting_active_state (psd_name2, psv_name2);
    pg_setting_condition : Boolean expression of psd_name2 and signal pin;
}
```

Example

```plaintext
pg_setting_definition (psd) {
    pg_setting_value (psv1) {
        ...
        }
    }
```

The `pg_setting_value` group contains the following attributes.

**pg_pin_active_state and pg_pin_condition Attributes**

These attributes define a system power state as the function of the PG pins and the signal pins in the `pg_setting_value` group.

The `pg_pin_active_state` complex attribute defines the active power supply state, `pg_state`, for a specified PG pin. The attribute only applies to a multi-state power supply. For a PG pin with a single state, the pin is active when it is on.

The `pg_pin_condition` attribute specifies the logic condition when the system is in the power state named `psv_name`. For a single-state PG pin, this condition evaluates to true when the PG pin is in the on state. For a multi-state PG pin, this condition evaluates to true whenever the PG pin is in the active state specified by the `pg_pin_active_state` attribute.

Syntax

```plaintext
pg_setting_value (psv_name) {
    pg_pin_active_state (pg_pin, pg_state);
    pg_pin_condition : Boolean expression of pg_pin and signal pin;
}
```

Example

```plaintext
pg_setting_definition (psd) {
    pg_setting_value (on) {
        pg_pin_active_state (VDD, normal);
        pg_pin_condition : "VDD * !VSS" ;
    }
    pg_setting_value (sleep) {
        pg_pin_active_state (VDD, under_drive);
        pg_pin_condition : "VDD * !VSS * sig_sleep" ;
    }
}```
pg_setting_active_state and pg_setting_condition Attributes

These attributes define a system power state as a function of other system power states and signal pins in the pg_setting_value group. Because they define the system power states hierarchically, you do not need to repeat the complete list of PG pin conditions and states for a large block.

The pg_setting_active_state complex attribute specifies the active system power state, psv_name2, of another pg_setting_definition group (named psd_name2) in the system power state, psv_name.

The pg_setting_condition attribute specifies the logic condition when the system is in the power state, psv_name. This condition evaluates to true whenever psd_name2 is in the active state specified by the pg_setting_active_state attribute.

Syntax

pg_setting_value (psv_name) {
  pg_setting_active_state (psd_name2, psv_name2);
  pg_setting_condition : Boolean expression of psd_name2 & signal pin;
}

Example

pg_setting_definition(psd) {
  pg_setting_value (psv1) {... }
  pg_setting_value (psv2) {... }
}

pg_setting_definition(psd2) {
  pg_setting_value (psv1) {... }
  pg_setting_value (psv2) {... }
}

pg_setting_definition(top_psd) {
  pg_setting_value (psv1) {
    pg_setting_active_state(psd, psv1);
    pg_setting_active_state(psd2, psv2);
    pg_setting_condition : "psd * psd2" ;
  }
  pg_setting_value (psv2) {
    pg_setting_active_state(psd, psv2);
    pg_setting_active_state(psd2, psv1);
    pg_setting_condition : "psd * psd2" ;
  }
}

default_pg_setting Attribute

The default_pg_setting attribute specifies a default power state to match when the explicitly defined power states do not completely cover the state space of the pg_setting_definition group. The default power state also acts as the initial power state of the pg_setting_definition group.
psv_name is the name of a pg_setting_value group specified in the same pg_setting_definition group.

Syntax
pg_setting_definition (psd_name) {
    ...
    default_pg_setting : psv_name ;
}

Example
pg_setting_definition (psd) {
    default_pg_setting : "psv1" ;
}

pg_setting_transition Group

The pg_setting_transition group specifies the legal and illegal transitions between the PG modes defined in the pg_setting_definition group.

pst_name is the name you assign to the transition.

Syntax
pg_setting_transition (pst_name) {
    is_illegal : [ true | false ];
    start_setting : psv_name1;
    end_setting : psv_name2;
}

pst_name is the name you assign to the transition.

Example
pg_setting_transition (sleep) {
    start_setting : on;
    end_setting : sleep;
}

The pg_setting_transition group contains the following attributes.

is_illegal Attribute

The is_illegal attribute specifies if the transition, pst_name, is illegal. The default is false.

start_setting and end_setting Attributes

The start_setting attribute specifies the power state from where the transition, pst_name, starts. The end_setting attribute specifies the power state where the transition, pst_name, ends. Both the power states must be defined in the same pg_setting_definition group.
If you do not specify the `start_setting` attribute, it means that all the power states in the same `pg_setting_definition` group can be the start states for the transition, `pst_name`.

If you do not specify the `end_setting` attribute, it means that all the power states in the same `pg_setting_definition` group can be the end states for the transition, `pst_name`.

**illegal_transition_if_undefined Attribute**

The `illegal_transition_if_undefined` attribute, if set to `true`, means that all the undefined transitions in the `pg_setting_definition` group are illegal or invalid. By default, all undefined transitions are valid.

**Syntax**

```plaintext
pg_setting_definition(psd_name) {
  Illegal_transition_if_undefined : true | false;
}
```

**Specifying Power States in Timing, Power, and Noise Models**

Use the following syntax to make the conditional timing, power, and noise models power-state aware and to address frequency-scaled power states.

**pg_setting Attribute in mode_value Group**

The `pg_setting` complex attribute specifies the power state of a cell mode by referencing a power state (`psv_name`) defined in the `pg_setting_definition` group of the cell.

You can define multiple `pg_setting` attributes in a `mode_value` group provided each of the `pg_setting` attributes references a power state from a different `pg_setting_definition` group.

**Syntax**

```plaintext
cell (cell_name) {
  mode_definition (mode_def) {
    mode_value (mode_name) {
      pg_setting (psd_name, psv_name);
    }
  }
}
```

**mode Attribute in minimum_period and min_pulse_width Groups**

The `mode` complex attribute enables conditional data modeling in the frequency-scaled groups, that is, the `minimum_period` and `min_pulse_width` groups.
The `mode` attribute uses the `mode_name` to reference a `mode_value` group defined in the cell. When you specify the `pg_setting` attribute in the referenced `mode_value` group, the frequency-scaled group where the mode is referenced also becomes power-state aware.

**Syntax**

```plaintext
pin (pin_name) {
    minimum_period(minimum_period_name){
        mode(mode_def_name, mode_name);
    }
    min_pulse_width(min_pulse_width_name) {
        mode(mode_def_name, mode_name);
    }
}
```

---

**Defining PG Modes in Macro Cells**

To include the PG mode syntax in your current design flow, remember the following:

- PG mode syntax is corner-independent.
  - You must define the operating voltage ranges for all the PG states, if multi-state.
  - To maintain consistency between different corners, you must define all the PG modes for a cell.
- For a power rail with only on and off states, you do not need to specify a voltage range. The PG modes are active in all corners.
- The default nominal voltage of a PG state (`nom_voltage` in the `voltage_state_range` attribute definition) is the default voltage that applies to all the designs of your intellectual property (IP).

**Examples**

The following examples illustrate the use of the PG mode syntax in modeling different cells.
Cell With Different Power States

The following is an example of a cell with multiple power states. The cell has two power pins, VDD and VDDS and one ground pin, VSS. The following table lists the voltage ranges of the PG pins.

<table>
<thead>
<tr>
<th>PG pin</th>
<th>Voltage range in state1</th>
<th>Voltage range in state 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD</td>
<td>1.08</td>
<td>0.92~1.00 (nominal: 0.96)</td>
</tr>
<tr>
<td>VDDS</td>
<td>1.08</td>
<td>X</td>
</tr>
<tr>
<td>VSS</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

```liberty
library (mylib) {
    voltage_map("VM1", 1.08);
    voltage_map("VM2", 1.08);
    voltage_map("VG", 0);
    voltage_state_range_list(VM1) {
        /* state1 */
        voltage_state_range(HV, 1.08);
        /* state2 */
        voltage_state_range(HV_L, 0.92, 0.96, 1.00);
        /* state not used in cell(mycell) */
        voltage_state_range(HV_H, 0.98, 1.08);
    }

cell (mycell) {
    pg_pin (VDD) {
        pg_type : primary_power;
        voltage_name: "VM1";
    }
    pg_pin(VDDS) {
        pg_type : primary_power;
        voltage_name: "VM2";
    }
    pg_pin(VSS) {
        pg_type : primary_ground;
        voltage_name: "VG";
    }

    pg_setting_definition (pst) {
        pg_setting_value(ps_1) {
            pg_pin_condition : "VDD * VDDS * !VSS";
            pg_pin_active_state(VDD, HV);
        }
        pg_setting_value(ps_2) {
            pg_pin_condition : "VDD * !VDDS * !VSS";
        }
    }
}
```
Power State Transition

The following is an example of a cell with transition between the power states. The cell has two blocks: one powered by VDD, and the other powered by VDDS. The table lists the voltage ranges of the PG pins.

<table>
<thead>
<tr>
<th>PG pin</th>
<th>Voltage range in state1</th>
<th>Voltage range in state 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD</td>
<td>0.7~0.9</td>
<td>0.6</td>
</tr>
<tr>
<td>VDDS</td>
<td>0.8~1.0</td>
<td>X</td>
</tr>
<tr>
<td>VSS1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

library (mylib) {

voltage_map("VM1", 0.8);
voltage_map("VM2", 0.9);
voltage_map("VG", 0);
voltage_state_range_list(VM1) {
    voltage_state_range(on, 0.7, 0.9);
    voltage_state_range(pon, 0.6);
}
voltage_state_range_list(VM2) {
    voltage_state_range(on, 0.8, 1.0);
    /* this state not used in cell(mycell) */
    voltage_state_range(pon, 0.6);
}

cell (mycell) {
    pg_pin (VDD) {
        pg_type : primary_power;
        voltage_name: "VM1";
    }
    pg_pin(VDDS){
        pg_type : primary_power;
        voltage_name: "VM2";
    }
    pg_pin(VSS) {

pg_type : primary_ground;
voltage_name: "VG";
}

/* power state of block 1 */
pg_setting_definition(PD_SSAH) {

/* state1 */
pg_setting_value(GO) {
    pg_pin_condition : "VDD * !VSS";
    pg_pin_active_state(VDD, on);
}
/* state2 */
pg_setting_value(SLEEP) {
    pg_pin_condition: "VDD * !VSS * sp_on";
    pg_pin_active_state(VDD, pon);
}
/* state off */
pg_setting_value(OFF) {
    pg_pin_condition: "!VDD + VSS ";
}
/* transition */
pg_setting_transition(turn_sleep) {
    start_setting : GO;
    end_setting : SLEEP;
}
illegal_transition_if_undefined : true ;
} /* end of PD_SSAH */

/* power state of block 2 */
pg_setting_definition (PD_SSBH) {
    pg_setting_value (ON) {
        pg_pin_condition : "VDDS * !VSS";
        pg_pin_active_state(VDDS, on);
    }
    pg_setting_value (OFF) {
        pg_pin_condition : "!VDDS + VSS";
    }
} /* end of PD_SSBH */

/* power state of the cell */
pg_setting_definition (PD) {
    pg_setting_value (S1) {
        pg_setting_condition : "PD_SSAH * PD_SSBH" ;
        pg_setting_active_state(PD_SSAH, GO);
        pg_setting_active_state(PD_SSBH, ON);
    }
    pg_setting_value (S2) {
        pg_setting_condition : "PD_SSAH * PD_SSBH" ;
        pg_setting_active_state(PD_SSAH, SLEEP);
        pg_setting_active_state(PD_SSBH, OFF);
    }
    pg_setting_value (OFF) {

Macro Cell With PG Modes

The following example shows a Liberty model of a macro cell that contains two blocks (block1 and block2). block1 has primary power pin, VDD1, and backup power pin, VDD3. block2 has primary power pin, VDD2.

<table>
<thead>
<tr>
<th>PG pin</th>
<th>Voltage range in state1</th>
<th>Voltage range in state 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD1</td>
<td>0.81~0.99</td>
<td>0.9~1.05</td>
</tr>
<tr>
<td></td>
<td>(nominal: 0.9)</td>
<td>(nominal: 1.0)</td>
</tr>
<tr>
<td>VDD2</td>
<td>0.72~0.88</td>
<td>0.9~1.15</td>
</tr>
<tr>
<td>VDD3</td>
<td>0.9</td>
<td>X</td>
</tr>
<tr>
<td>VSS1/VSS2</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

library (mylib) {
  voltage_map("VM1", 0.9);
  voltage_map("VM2", 0.8);
  voltage_map("VM3", 0.9);
  voltage_map("VG", 0);
  voltage_state_range_list(VM1) {
    voltage_state_range(v1_l, 0.81, 0.9, 0.99);
    voltage_state_range(v1_h, 0.9, 1.0, 1.05);
  }
  voltage_state_range_list(VM2) {
    voltage_state_range(v2_l, 0.72, 0.88);
    voltage_state_range(v2_h, 0.9, 1.15);
  }
  cell (my_macro_cell) {
    pg_pin(VDD1) {
      pg_type : primary_power;
      voltage_name : "VM1";
    }
    pg_pin(VDD2) {
      pg_type : primary_power;
      voltage_name : "VM2";
    }
    pg_pin(VDD3) {
      pg_type : backup_power;
      voltage_name : "VM3";
    }
  }
}
pg_pin(VSS1) {
    pg_type : primary_ground;
    voltage_name : "VG";
}

pg_pin(VSS2) {
    pg_type : primary_ground;
    voltage_name : "VG";
}

/* Block1 power states: VDD1/VDD3/VSS1 */
pg_setting_definition(P1) {
    pg_setting_value(sleep) {
        pg_pin_condition : "!VDD1 * VDD3 * !VSS1";
    }
    pg_setting_value(normal) {
        pg_pin_condition : "VDD1 * VDD3 * !VSS1";
        pg_pin_active_state(VDD1, v1_l);
    }
    pg_setting_value(power_mode) {
        pg_pin_condition : "VDD1 * VDD3 * !VSS1";
        pg_pin_active_state(VDD1, v1_h);
    }
    pg_setting_value(off) {
        pg_pin_condition : "!VDD1 * VDD3 + VSS";
    }
    default_pg_setting : sleep ;
    pg_setting_transition(turn_on) {
        end_setting : normal ;
    }
    pg_setting_transition(no_way) {
        is_illegal : true ;
        start_setting : power_mode ;
        end_setting : sleep ;
    }
    pg_setting_transition(power_save) {
        start_setting : normal ;
        end_setting : sleep ;
    }
    illegal_transition_if_undefined : true ;
}

/* Block2 power states: VDD2/VSS2 */
pg_setting_definition(P2) {
    pg_setting_value(low_speed) {
        pg_pin_condition : "VDD2 * !VSS2";
        pg_pin_active_state(VDD2, v2_l);
    }
    pg_setting_value(high_speed) {
        pg_pin_condition : "VDD2 * !VSS2";
        pg_pin_active_state(VDD2, v2_h);
    }
    pg_setting_value(off) {
        pg_pin_condition : "!VDD2 + VSS";
    }
pg_setting_transition(power_mode) {
    start_setting : low_speed;
    end_setting : high_speed;
}

pg_setting_transition(power_save) {
    start_setting : high_speed;
    end_setting : low_speed;
}

default_pg_setting: low_speed;

/* Cell power states: block1 & block2 */
pg_setting_definition(TOP) {
    pg_setting_value(power_save) {
        pg_setting_condition : "P1 * P2";
        pg_setting_active_state (P1, sleep);
        pg_setting_active_state (P2, low_speed);
    }

    pg_setting_value(normal_mode) {
        pg_setting_condition : "P1 * P2";
        pg_setting_active_state (P1, normal);
        pg_setting_active_state (P2, low_speed);
    }

    pg_setting_value(power_mode) {
        pg_setting_condition : "P1 * P2";
        pg_setting_active_state (P1, high_speed);
        pg_setting_active_state (P2, high_speed);
    }

    pg_setting_value(off) {
        pg_setting_condition : "P1 * P2";
        pg_setting_active_state (P1, off);
        pg_setting_active_state (P2, off);
    }

    default_pg_setting : normal_mode;
}

mode_definition(block1) {
    mutually_exclusive_mode_values : true;
    initial_mode : "Idle";
    mode_value(Idle) {
        when : "!Enable";
        pg_setting(p1, sleep);
    }

    mode_value(Active) {
        when : "Enable & CS1";
        pg_setting(p1, normal);
    }

    mode_value(Standby) {
        when : "Enable & !CS1";
        pg_setting(p1, normal);
    }
}
Retention Cell Leakage Power in Different Power Modes

Retention Cell With Two Modes

The cell operates in the normal mode when VDD is on, and in the retention mode when VDD is off. VDD is the primary power supply and VDDG is the always-on (backup) power supply. RETN is the save or restore pin. The data is saved when the pin is high and restored when the pin is low.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Working voltage (V)</th>
<th>pg_setting_value</th>
<th>Normal</th>
<th>Retention</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETN</td>
<td>-</td>
<td>1/0/r/f</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CK</td>
<td>-</td>
<td>1/0/r/f</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>-</td>
<td>1/0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>VDD</td>
<td>0.63</td>
<td>1</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>VDDG</td>
<td>0.63</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>VSS/VSSG</td>
<td>0/0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

To model different leakage power values when VDD is on or off, specify the PG pin and signal pin states for the different modes of operation of the cell.

library(mylib) {
  voltage_map (VDD, 0.63); /* Switchable Power */
  voltage_map (VDDG, 0.63); /* Backup Power */
  voltage_map (VSS, 0); /* Primary Ground */
  voltage_map (VSSG, 0); /* Backup Ground */
...
  cell (my_retention_cell) {
    ...
    pg_pin (VDD) {
      pg_type : primary_power;
      voltage_name : "VDD";
    }
    pg_pin (VDDG) {
      pg_type : backup_power;
      voltage_name : "VDDG";
    }
  }
}
pg_pin (VSS) {
    pg_type : primary_ground;
    voltage_name : "VSS";
}
pg_pin (VSSG) {
    pg_type : backup_ground;
    voltage_name : "VSSG";
}

pg_setting_definition (PD) {
    pg_setting_value (normal) {
        pg_pin_condition : "VDDG & VDD & !VSS & !VSSG";
    }
    pg_setting_value (retention) {
        pg_pin_condition : "VDDG & !VDD & !VSS & !VSSG";
    }
    pg_setting_value (off) {
        pg_pin_condition : "!VDDG & !VDD + VSS + VSSG";
    }
}

mode_definition (power_state){
    /* signal: CK/D/RETN could be 1/0 */
    mode_value (active) {
        pg_setting (PD, normal);
    }
    /* signal: CK/D are X, RETN is 0 */
    mode_value (retained) {
        when : "!RETN";
        pg_setting (PD, retention);
    }
    /* pin definitions are omitted */
    /* only 2 leakage power groups possible in retention mode */
    leakage_power () {
        mode (power_state, retained);
        when : "!(RETN)";
        value : 0;
        related_pg_pin : VDD;
    }
    leakage_power () {
        mode (power_state, retained);
        when : "!(RETN)";
        value : 1.31337e-06;
        related_pg_pin : VDDG;
    }
    /* 16 leakage power groups with different CK/D/RETN signal pin states */
    leakage_power () {
        mode (power_state, active);
        value : 6.87551e-06;
        when : "(CK&D&RETN)";
        related_pg_pin : VDD;
    }
    leakage_power () {
mode (power_state, active);
value : 9.57963e-07;
when : "(CK&D&RETN)"
related_pg_pin : VDDG;
}
leakage_power () {
    mode (power_state, active);
    value : 7.02861e-06;
    when : "(CK&D!RETN)"
related_pg_pin : VDD;
}
leakage_power () {
    mode (power_state, active);
    value : 1.21337e-06;
    when : "(CK&D!RETN)"
related_pg_pin : VDDG;
}
......
}/* end cell group */
}/* end library group */

Retention Cell With Three Modes

Compared to the retention cell with two modes, the retention cell in this example has an extra mode to save more power in the deep retention mode. The cell has three modes: normal, retention, and retention_low.

VDDG is the always-on (backup) power supply that works at two voltage values:

- A high voltage of 0.63 V (operating range is 0.56 ~ 0.70)
- A low voltage of 0.45 V (operating range is 0.41 ~ 0.49) to save more power

<table>
<thead>
<tr>
<th>Pin</th>
<th>pg_setting_value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal</td>
</tr>
<tr>
<td>RETN</td>
<td>1/0/r/f</td>
</tr>
<tr>
<td>CK</td>
<td>1/0/r/f</td>
</tr>
<tr>
<td>D</td>
<td>1/0</td>
</tr>
<tr>
<td>VDD</td>
<td>1</td>
</tr>
<tr>
<td>VDDG</td>
<td>1</td>
</tr>
<tr>
<td>VSS/VSSG</td>
<td>0</td>
</tr>
</tbody>
</table>
To model this retention cell, the following two corner libraries are needed:

- **PVT1.lib**: VDD = 0.63 V; VDDG = 0.63 V
- **PVT2.lib**: VDD = 0.58 V; VDDG = 0.45 V

**PVT1.lib**

```liberty
library(mylib) {
  voltage_map (VDD, 0.63);
  voltage_map (VDDG, 0.63); /* characterization voltage */
  voltage_map (VSS, 0);
  voltage_map (VSSG, 0);
  voltage_state_range_list(VDDG) {
    voltage_state_range(high, 0.56, 0.63, 0.70); /* min/nom/max */
    voltage_state_range(low, 0.41, 0.45, 0.49); /* min/nom/max */
  }
  cell (my_retention_cell) {
    pg_setting_definition (PD) {
      pg_setting_value (normal) {
        pg_pin_condition : " VDDG & VDD & !VSS & !VSSG";
        pg_active_state(VDDG, high);
      }
      pg_setting_value (retention) {
        pg_pin_condition : " VDDG & !VDD & !VSS & !VSSG";
        pg_active_state(VDDG, high);
      }
      pg_setting_value (retention_low) {
        pg_pin_condition : " VDDG & !VDD & !VSS & !VSSG";
        pg_active_state(VDDG, low);
      }
      pg_setting_value (off) {
        pg_pin_condition : "!VDDG & !VDD + VSS + VSSG";
      }
      /* PG mode cannot directly move from active to retention_low mode or vice-versa */
      pg_setting_transition(normal_to_low) {
        is_illegal : true;
        start_setting : normal;
        end_setting : retention_low;
      }
      pg_setting_transition(low_to_normal) {
        is_illegal : true;
        start_setting : retention_low;
        end_setting : normal;
      }
    }
    mode_definition (power_state){
      /* signal: CK/D RETN could be 1/0 */
      mode_value (active) {
        pg_setting (PD, normal);
      }
      /* signal: CK/D are X, RETN is 0 */
  }
```
mode_value (retained) {
    when : "!RETN";
    pg_setting (PD, retention);
}
/* signal: CK/D are X, RETN is 0 */
mode_value (retained_low) {
    when : "!RETN";
    pg_setting (PD, retention_low);
}
/* retained mode works in this corner */
leakage_power () {
    mode (power_state, retained);
    when : "(!RETN)"
    value : 0;
    related_pg_pin : VDD;
}
leakage_power () {
    mode (power_state, retained);
    when : "(!RETN)"
    value : 1.31337e-06;
    related_pg_pin : VDDG;
}
/* active mode is same as example 12.1.4.1 */
} /* end cell group */
} /* end library group */

PVT2.lib
library(mylib) {
    voltage_map (VDD, 0.58);
    voltage_map (VDDG, 0.45); /* characterization voltage */
    voltage_map (VSS, 0);
    voltage_map (VSSG, 0);
    voltage_state_range_list(VDDG) {
        voltage_state_range(high, 0.56, 0.63, 0.70); /* min/nom/max */
        voltage_state_range(low, 0.41, 0.45, 0.49); /* min/nom/max */
    }
    cell (my_retention_cell) {
        pg_setting_definition (PD) {
            pg_setting_value (normal) {
                pg_pin_condition : " VDDG & VDD & !VSS & !VSSG";
                pg_active_state(VDDG, high);
            }
            pg_setting_value (retention) {
                pg_pin_condition : " VDDG & !VDD & !VSS & !VSSG";
                pg_active_state(VDDG, high);
            }
            pg_setting_value (retention_low) {
                pg_pin_condition : " VDDG & !VDD & !VSS & !VSSG";
                pg_active_state(VDDG, low);
            }
            pg_setting_value (off) {
pg_pin_condition : "!VDDG & !VDD + VSS + VSSG";
}

/* PG mode cannot directly move from active to retention_low mode and vice-versa */
pg_setting_transition(normal_to_low) {
  is_illegal : true;
  start_setting : normal;
  end_setting : retention_low;
}
pg_setting_transition(low_to_normal) {
  is_illegal : true;
  start_setting : retention_low;
  end_setting : normal;
}

mode_definition (power_state) {
  /* signal: CK/D/RETN could be 1/0 */
  mode_value (active) {
    pg_setting (PD, normal);
  }
  /* signal: CK/D are X, RETN is 0 */
  mode_value (retained) {
    when : "!RETN";
    pg_setting (PD, retention);
  }
  /* signal: CK/D are X, RETN is 0 */
  mode_value (retained_low) {
    when : "!RETN";
    pg_setting (PD, retention_low);
  }
}
/* retained_low mode works in this corner */
leakage_power () {
  mode (power_state, retained_low);
  when : "(!RETN)"
  value : 0;
  related_pg_pin : VDD;
}

leakage_power () {
  mode (power_state, retained_low);
  when : "(!RETN)"
  value : 1.31337e-07;
  related_pg_pin : VDDG;
}
/* No leakage power group in active mode as VDDG is in low state */
/* end cell group */
}/* end library group */
Feedthrough Signal Pin Modeling

A feedthrough is created when two or more pins of a cell share the same physical or unbuffered logical net. Some feedthroughs have no electrical connections with the cell power pins and do not affect cell behavior.

Feedthrough pins can be of two types:

- Multipin, also known as logical feedthroughs or shorted pins
- Single-pin, also known as a pass-through, physical feedthrough, or floating pin

Feedthrough pins can also be part of buses.

Multipin Feedthroughs

Figure 9-5 is a schematic of a two-pin feedthrough cell with feedthrough pins, A and Y. Pins A and Y appear in all the model views (such as Verilog) as two separate external pins.

Figure 9-5  A Two-Pin Feedthrough

Multipin Feedthrough Syntax

library (library_name) {
  cell(cell_name) {
    ...
    short(pin_name, pin_name, ...);
    ...
    pin (pin_name) {
      direction : inout;
    ...
    } /* end pin group */
    ...
  } /* end cell group */
} /* end library group */
To model multipin feedthroughs as shown in Figure 9-5, use the short attribute in the cell group.

**short Attribute**

The short complex attribute lists the shorted ports or pins that are connected by metal or polytrace. You must list at least two pin names.

To model multiple sets of feedthrough pins in a cell, specify multiple short attributes. For example, if pins a and b are shorted to create a feedthrough, and pins c and d are shorted to create another feedthrough, use:

```plaintext
short (a, b);
short (c, d);
```

However, if pin a is shorted with pin b, and pin a is also shorted with pin c, that is, pins a, b, and c are shorted, use:

```plaintext
short (a, b, c);
```

This is equivalent to:

```plaintext
short (a, b);
short (a, c);
```

**Example 9-9** is a library model of the cell shown in Figure 9-5.

**Example 9-9  A Two-Pin Feedthrough Cell Model**

```plaintext
library(mylib) {
  cell(mycell) {
    ...
    /* signal pins A and Y are feedthrough pins */
    short(A, Y);
    ...
    pin(A) { /* Do not specify related_power_pin or related_ground_pin */
      /* set pin direction based on actual cell characteristics */
      direction : inout;
    ...
    }
    pin(Y) { /* Do not specify related_power_pin or related_ground_pin */
      direction : inout;
    ...
    }
  } /* end cell */
}
```

```plaintext
pin (B C) {
  related_power_pin : "VDD";
  related_ground_pin : "VSS";
  direction : "input";
}
```

```plaintext
pin (Z) {
  related_power_pin : "VDD";
}
```
related_ground_pin : "VSS";
direction : "output";
power_down_function : "!VDD + VSS";
function : "B * C";
timing () {
    related_pin : "B C";
    ...
    } /* end pin group */
} /* end cell group */
} /* end library group */

---

**Single-Pin Feedthrough**

*Figure 9-6* is a schematic of a single-pin feedthrough cell, with feedthrough pin A. In all the model views, only one external pin, bus, or bundle, A, is declared even though the signal can establish one or more physical connections with other cells.

*Figure 9-6  Single-Pin Feedthrough*

![Single-Pin Feedthrough Diagram](image)

**Single-Pin Feedthrough Syntax**

```liberty
library (library_name) {
    cell(cell_name) {
        ...
        pin (single_pin_feedthrough_name) {
            direction : inout;
        } /* end pin group */
        ...
        } /* end cell group */
    } /* end library group */
}
```

To model a single-pin feedthrough, as shown in Figure 9-6:

- Specify the feedthrough pin as a floating or unused signal pin without parasitics.
• Do not specify any related_power_pin and related_ground_pin associations for these pins.

Example 9-10 is a library model of the cell shown in Figure 9-6.

Example 9-10  A Single-Pin Feedthrough Model

```liberty
library(mylib) {
  cell (mycell) {
    pg_pin (VDD) {
      voltage_name : "VDD";
      pg_type : "primary_power";
    }
    pg_pin (VSS) {
      voltage_name : "VSS";
      pg_type : "primary_ground";
    }
    pin (B C) {
      related_power_pin : "VDD";
      related_ground_pin : "VSS";
      direction : "input";
      ...
    }
    pin (A) { /* Do not specify related_power_pin or related_ground_pin */
      /* set pin direction based on actual cell characteristics */
      direction : "inout";
      ...
    }
    pin (Z) {
      related_power_pin : "VDD";
      related_ground_pin : "VSS";
      direction : "output";
      function : "B * C";
      timing () {
        related_pin : "B C";
        ...
      }
      ...
    } /* end pin group */
  } /* end cell group*/
} /* end library group */
```
Silicon-on-Insulator (SOI) Cell Modeling

Silicon-on-insulator (SOI) devices are fabricated on a silicon layer that rests upon an insulating layer on the substrate. The presence of the insulating layer makes the drain and source junctions shallow, with small capacitances. Therefore, the SOI devices have better electrical characteristics resulting in improved switching speed and reduced power dissipation.

The isolation of the silicon layer from the substrate improves the operating speed, reduces leakage to the substrate, and reduces the number of latch-ups. It is easy to closely pack these inherently insulated structures for high device densities. Further, the SOI fabrication process requires only minor changes to the bulk-CMOS process flow.

Depending on the thickness of the silicon layer, an SOI cell is of two types:

- Fully depleted SOI (FDSOI) cell, where the channel depletion width is greater than the thickness of the silicon layer. Therefore, the channel is fully depleted.
- Partially depleted SOI (PDSOI) cell, where the channel depletion width is less than the thickness of the silicon layer. Therefore, the channel is partially depleted.

Figure 9-7 shows a typical SOI cell schematic.

Figure 9-7  A Typical SOI Cell Schematic

The modeling syntax of the SOI cell is:

```liberty
library(library_name) {
    ... 
is_soi: true | false;
    ... 
 cell(cell_name) {
    ... 
```

![Silicon-on-Insulator (SOI) Cell Model Diagram](image-url)
Table 9-3 shows the differences in modeling substrate-bias pins between a bulk-CMOS and the SOI cell.

<table>
<thead>
<tr>
<th>Bulk-CMOS</th>
<th>SOI</th>
</tr>
</thead>
<tbody>
<tr>
<td>For a substrate-bias pin associated with a primary power pin, the <code>pg_type</code> attribute must be <code>nwell</code>.</td>
<td>For a substrate-bias pin associated with a primary power pin, the <code>pg_type</code> attribute can be <code>pwell</code> or <code>nwell</code>.</td>
</tr>
<tr>
<td>For a substrate-bias pin associated with a primary ground pin, the <code>pg_type</code> attribute must be <code>pwell</code>.</td>
<td>For a substrate-bias pin associated with a primary ground pin, the <code>pg_type</code> attribute can be <code>pwell</code> or <code>nwell</code>.</td>
</tr>
</tbody>
</table>

Example 9-11 shows a typical SOI library and cell model.

```liberty
library(SOI) {
  delay_model : table_lookup;
  /* unit attributes */
  ... is_soil : true; 
  ... /* operation conditions */
  ... /* threshold definitions */
  ... /* default attributes */
  ... cell(std_cell_inv) {
    is_soil : true;
    cell_footprint : inv;
    area : 1.0;
    pg_pin(vdd) { 
      voltage_name : vdd;
      pg_type : primary_power;
      related_bias_pin : "vpw";
    }
  }
}
```
Library-Level Attribute
This section describes a library-level attribute of SOI cells.

is_soi Attribute
The is_soi attribute specifies that all the cells in a library are SOI cells. The default is false, which means that the library cells are bulk-CMOS cells.

If the is_soi attribute is specified at both the library and cell levels, the cell-level value overrides the library-level value.

Cell-Level Attribute
This section describes a cell-level attribute of SOI cells.

is_soi Attribute
The is_soi attribute specifies that the cell is an SOI cell. The default is false, which means that the cell is a bulk-CMOS cell.

Level-Shifter Cells in a Multivoltage Design
Level-shifter cells (or buffer-type level-shifter cells) and isolation cells are used to connect the netlist in different power domains, to meet design constraints. In multivoltage and shut-down designs, both level shifting and isolation are required. An enable level-shifter cell fulfills both these requirements because it can function as an isolation or a level-shifter cell.

Implementation tools need the following information about level-shifter cells from the cell library:

• Which power and ground pin of the level-shifter cell is used for voltage boundary hookup during its insertion. This information allows the optimization tools to determine on which side of the voltage boundary a particular level-shifter cell is allowed.

• Which voltage conversions the particular level-shifter cell can handle. Specifically, does the level-shifter cell work for conversion from high voltage to low voltage (HL), from low voltage to high voltage (LH), or both (HL_LH)?

• What the input and output voltage ranges are for a level-shifter cell under all operating conditions.
Operating Voltages

In a multivoltage design, each design instance can operate at its specified operating voltage. Therefore, each voltage must correspond to one or more logical hierarchies. All cells in a hierarchy operate at the same voltage except for level shifters.

The operating voltages are annotated on the top design, design instances, or hierarchical ports through PVT operating conditions.

Level Shifter Functionality

A level shifter functions like a buffer, except that the input pin and output pin voltages are different. These cells are necessary in a multivoltage design because the nets connecting pins at two different operating voltages can cause a design violation. Level shifters provide these nets with the needed voltage adjustments.

A level shifter has two components:

- Two power supplies
- Specified input and output voltages

The functionality of level shifters includes:

- Identifying nets in the design that need voltage adjustments
- Analyzing the target library for the availability of level shifters
- Ripping the net and instantiating level shifters where appropriate

Basic Level-Shifter Syntax

The basic syntax for level-shifter cells is as follows:

```plaintext
cell(level_shifter) {
    is_level_shifter : true;
    level_shifter_type : HL | LH | HL_LH;
    input_voltage_range ("float, float");
    output_voltage_range ("float, float");
    ...
    pg_pin(pg_pin_name_P) {
        pg_type : primary_power;
        std_cell_main_rail : true;
        ...
    }
    pg_pin(pg_pin_name_G) {
        pg_type : primary_ground;
    ...
```
Cell-Level Attributes

This section describes cell-level attributes for level-shifter cells.

is_level_shifter Attribute

The `is_level_shifter` simple attribute identifies a cell as a level shifter. The valid values of this attribute are `true` or `false`. If not specified, the default is `false`, meaning that the cell is a standard cell.

level_shifter_type Attribute

The `level_shifter_type` complex attribute specifies the supported voltage conversion type. The valid values are

- LH
  - Low to high
- HL
  - High to low
HL_LH

High to low and low to high

The level_shifter_type attribute is optional. The default is HL_LH.

input_voltage_range Attribute

The input_voltage_range attribute specifies the allowed voltage range of the level-shifter input pin and the voltage range for all input pins of the cell under all possible operating conditions (defined across multiple libraries). The attribute defines two floating values: the first is the lower bound, and the second is the upper bound.

The input_voltage_range syntax differs from the pin-level input_signal_level_low and input_signal_level_high syntax in the following ways:

- The input_signal_level_low and input_signal_level_high attributes are defined on the input pins under one operating condition (the default operating condition of the library).
- The input_signal_level_low and input_signal_level_high attributes specify the partial voltage swing of an input pin. Use these attributes to specify partial swings rather than the full rail-to-rail swing. The input_voltage_range attribute is not related to the voltage swing.

Note:

The input_voltage_range and output_voltage_range attributes must be defined together.

output_voltage_range Attribute

The output_voltage_range attribute is similar to the input_voltage_range attribute except that it specifies the allowed voltage range of the level shifter for the output pin instead of the input pin.

The output_voltage_range syntax differs from the pin-level output_signal_level_low and output_signal_level_high syntax in the following ways:

- The output_signal_level_low and output_signal_level_high attributes are defined on the output pins under one operating condition (the default operating condition of the library).
- The output_signal_level_low and output_signal_level_high attributes specify the partial voltage swing of an output pin. Use these attributes to specify partial swings rather than the full rail-to-rail swing. The output_voltage_range attribute is not related to the voltage swing.
Note:
The `input_voltage_range` and `output_voltage_range` attributes must be defined together.

**ground_input_voltage_range Attribute**
The `ground_input_voltage_range` attribute specifies the ground voltage range of all the input pins of a level-shifter cell, under all possible operating conditions.

**ground_output_voltage_range Attribute**
The `ground_output_voltage_range` attribute specifies the ground voltage range of all the output pins of a level-shifter cell, under all possible operating conditions.

Note:
To specify the ground voltage range using the `ground_input_voltage_range` or `ground_output_voltage_range` attributes, define a lower bound and an upper bound of the ground voltage. Specify the lower bound before the upper bound and ensure that the lower bound is less than or equal to the upper bound.

---

**Pin-Level Attributes**
This section describes pin-level attributes for level-shifter cells.

**std_cell_main_rail Attribute**
The `std_cell_main_rail` attribute is defined in a primary_power power pin. When the attribute is set to `true`, the `pg_pin` is used to determine which power pin is the main rail in the cell.

**level_shifter_data_pin Attribute**
The `level_shifter_data_pin` attribute identifies a data pin of a level-shifter cell. The default is `false`, meaning that the pin is a regular signal pin.

**level_shifter_enable_pin Attribute**
The `level_shifter_enable_pin` attribute identifies an enable pin of a level-shifter cell. The default is `false`, meaning that the pin is a regular signal pin.
input_voltage_range and output_voltage_range Attributes

The input_voltage_range and output_voltage_range attributes specify the allowed voltage ranges of the input or an output pin of the level-shifter cell. The attributes define two floating values where the first value is the lower bound and the second value is the upper bound.

Note:
The pin-level attribute specifications always override the cell-level specifications.

ground_input_voltage_range Attribute

The ground_input_voltage_range attribute specifies the ground voltage range of an input pin of a level-shifter cell.

ground_output_voltage_range Attribute

The ground_output_voltage_range attribute specifies the ground voltage range of an output pin of a level-shifter cell.

Note:
To specify the ground voltage range using the ground_input_voltage_range or ground_output_voltage_range attributes, define a lower bound and an upper bound of the ground voltage. Specify the lower bound before the upper bound and ensure that the lower bound is less than or equal to the upper bound.

input_signal_level Attribute

The input_signal_level attribute is defined at the pin level and is used to specify which signal is driving the input pin. The attribute defines special overdrive cells that do not have a physical relationship with the power and ground on input pins.

If the input_signal_level, related_power_pin, and related_ground_pin attributes are defined on any input pin, the full voltage swing derived from the input_signal_level attribute takes precedence over the voltage swing derived from the related_power_pin and related_ground_pin attributes during timing calculations.

power_down_function Attribute

The power_down_function attribute identifies the Boolean condition under which the cell's signal output pin is switched off (when the cell is in the off mode due to the external power pin states).
alive_during_power_up Attribute

The optional alive_during_power_up attribute specifies an active data pin that is powered by a more always-on supply. The default is false. If set to true, it indicates that the data pin is active while its related power pin is active.

You can specify this attribute only in a pin group where the isolation_cell_data_pin or the level_shifter_data_pin attribute is set to true.

Enable Level-Shifter Cell

An enable level-shifter cell generates a voltage shift between the input and output. An additional enable input controls the output.

Differential Level-Shifter Cell

A differential level-shifter cell has a pair of complementary data input pins to generate a voltage difference or shift between the input and output levels, with only a single supply voltage. This voltage shift is significantly greater that generated by other level-shifter cells.

The syntax for modeling the differential level-shifter cell is as follows:

```liberty
cell(cell_name) {
   is_level_shifter : true;
   pin_opposite("pin_name", "pin_name");
   contention_condition : "boolean_expression";
   ...
   pin (pin_name) {
      direction : input;
      level_shifter_data_pin : true;
      input_signal_level : voltage_rail_name;
      ...
   }/* End pin group */
   pin (pin_name) {
      direction : input;
      level_shifter_data_pin : true;
      input_signal_level : voltage_rail_name;
      timing(){/* optional */
         related_pin : "pin_name";
         timing_type : non_seq_setup_rising | non_seq_setup_falling | non_seq_hold_rising | non_seq_hold_falling;
         rise_constraint(template_name) {
            index_1 ("float, ..., float");
            index_2 ("float, ..., float");
            values("float, ..., float", ..., "float, ..., float");
         }
         fall_constraint(template_name) {
            index_1 ("float, ..., float");
            index_2 ("float, ..., float");
      }
```
Note:
This syntax is applicable to all types of differential level-shifter cells including low-to-high and high-to-low differential level-shifter cells, and differential level-shifter cells with or without enable inputs.

Cell-Level Attributes
This section describes a cell-level attribute for differential level-shifter cells.

is_differential_level_shifter Attribute
The is_differential_level_shifter attribute identifies a level-shifter cell as a differential level-shifter cell. The is_differential_level_shifter attribute is automatically set on a level-shifter cell that has pins with the pin_opposite and contention_condition attributes.

Pin-Level Attributes
This section describes pin-level attributes for level-shifter cells.
**pin_opposite Attribute**

The `pin_opposite` attribute specifies the logical inverse pin groups of a differential level-shifter cell. All the input pins used in the `pin_opposite` attribute must be data pins with the `level_shifter_data_pin` attribute set to `true`.

**contention_condition Attribute**

The `contention_condition` attribute specifies the Boolean expression of the invalid condition for a differential level-shifter cell. All the input pins used in the `contention_condition` attribute must be data pins with the `level_shifter_data_pin` attribute set to `true`.

**Note:**

A cell that has pins with the `pin_opposite` and `contention_condition` attributes is identified as a differential level-shifter cell and the `is_differential_level_shifter` and `dont_use` attributes are automatically set on this cell.

**Asynchronous Timing Constraints**

Use the `timing` group to specify the nonsequential setup and hold constraints between the two complementary input signals on the arrival of data. To set the asynchronous setup and hold constraints at the input pins, model the timing arcs with the following values of the `timing_type` attribute:

- `non_seq_setup_rising`: Checks the setup constraint of the rising edge of the related pin to the signal pin.
- `non_seq_setup_falling`: Checks the setup constraint of the falling edge of the related pin to the signal pin.
- `non_seq_hold_rising`: Checks the hold constraint for the rising edge of the related pin to the signal pin.
- `non_seq_hold_falling`: Checks the hold constraint for the falling edge of the related pin to the signal pin.

**Note:**

The related pin is an input pin without the `level_shifter_enable_pin` attribute.

---

**Clamping Enable Level-Shifter Cell Outputs**

In enable level-shifter (ELS) cells with multiple enable pins, clamping the outputs might be required to maintain them at particular logic levels, such as 0, 1, z, or a previously-latched value.

The clamp function modeling syntax for an enable level-shifter cell and its pins is as follows:
cell (cell_name) {
    is_level_shifter : true;
    ...
    pin(input_pin) {
        direction : input;
        level_shifter_data_pin : true;
        ...
    }
    pin(input_pin) {
        direction : input;
        level_shifter_enable_pin : true;
        ...
    }
    pin(input_pin) {
        direction : input;
        level_shifter_enable_pin : true;
        ...
    }
    pin(output_pin_name) {
        direction : output;
        clamp_0_function : "Boolean expression";
        clamp_1_function : "Boolean expression";
        clamp_z_function : "Boolean expression";
        clamp_latch_function : "Boolean expression";
        illegal_clamp_condition : "Boolean expression";
        ...
    }
    ...
}

**Pin-Level Attributes**

This section describes the pin-level clamp function attributes to clamp the output pins of an enable level-shifter cell.

**clamp_0_function Attribute**

The `clamp_0_function` attribute specifies the input condition for the enable pins of an enable level-shifter cell when the output clamps to 0.

**clamp_1_function Attribute**

The `clamp_1_function` attribute specifies the input condition for the enable pins of an enable level-shifter cell when the output clamps to 1.

**clamp_z_function Attribute**

The `clamp_z_function` attribute specifies the input condition for the enable pins of an enable level-shifter cell when the output clamps to z.
clamp_latch_function Attribute

The clamp_latch_function attribute specifies the input condition for the enable pins of an enable level-shifter cell when the output clamps to the previously latched value.

illegal_clamp_condition Attribute

The illegal_clamp_condition attribute specifies the invalid condition for the enable pins of an enable level-shifter cell. If the illegal_clamp_condition attribute is not specified, the invalid condition does not exist.

Note:
All the input pins used in the Boolean expressions of the clamp_0_function, clamp_1_function, clamp_z_function, and clamp_latch_function attributes must be enable pins with the level_shifter_enable_pin attribute set to true. The Boolean expressions of the clamp_0_function, clamp_1_function, clamp_z_function, clamp_latch_function, and illegal_clamp_condition attributes must be mutually exclusive.

Level Shifter Modeling Examples

The following sections provide examples for modeling a simple buffer type low-to-high level-shifter cell, a simple buffer type high-to-low level-shifter cell, an overdrive high-to-low level-shifter cell, a level-shifter cell with virtual bias pins, and enable level-shifter cells.

Simple Buffer Type Low-to-High Level Shifter

Figure 9-8 shows a simple buffer type low-to-high level-shifter cell modeled using the power and ground pin syntax and level-shifter attributes. The figure is followed by an example.
Figure 9-8  Buffer Type Low-to-High Level-Shifter Cell

library(level_shifter_cell_library_example) {
  voltage_map(VDD1, 0.8);
  voltage_map(VDD2, 1.2);
  voltage_map(VSS, 0.0);
  operating_conditions(XYZ) {
    process : 1.0;
    voltage : 3.0;
    temperature : 25.0;
  }
  default_operating_conditions : XYZ;
}

cell(Buffer_Type_LH_Level_shifter) {
  is_level_shifter : true;
  level_shifter_type : LH ;

  pg_pin(VDD1) {
    voltage_name : VDD1;
    pg_type : primary_power;
    std_cell_main_rail : true;
  }

  pg_pin(VDD2) {
    voltage_name : VDD2;
    pg_type : primary_power;
  }

  pg_pin(VSS) {
    voltage_name : VSS;
    pg_type : primary_ground;
  }

  leakage_power() {
    when : "!A";
    value : 1.5;
    related_pg_pin : VDD1;
  }
}
leakage_power() {
  when : "!A";
  value : 2.7;
  related_pg_pin : VDD2;
}

pin(A) {
  direction : input;
  related_power_pin : VDD1;
  related_ground_pin : VSS;
  input_voltage_range ( 0.7, 0.9);
}

pin(Z) {
  direction : output;
  related_power_pin : VDD2;
  related_ground_pin : VSS;
  function : "A";
  power_down_function : "!VDD1 + !VDD2 + VSS";
  output_voltage_range (1.1, 1.3);
}

timing() {
  related_pin : A;
  cell_rise(template) {
    ...
  }
  cell_fall(template) {
    ...
  }
  rise_transition(template) {
    ...
  }
  fall_transition(template) {
    ...
  }
}

internal_power() {
  related_pin : A;
  related_pg_pin : VDD1;
  ...
}

internal_power() {
  related_pin : A;
  related_pg_pin : VDD2;
  ...
}

} /* end pin group*/

} /*end cell group*/

} /*end library group*/
Simple Buffer Type High-to-Low Level Shifter

**Figure 9-9** shows a simple buffer type high-to-low level-shifter cell. The cell is modeled using the power and ground pin syntax and level-shifter attributes. Shifting the signal level from high to low voltage can be useful for timing accuracy. When you do this, the level-shifter cell receives a higher voltage signal as its input, which is characterized in the delay tables of the cell description.

**Figure 9-9  Buffer Type High-to-Low Level-Shifter Cell**

``` liberty
library(level_shifter_cell_library_example) {
  voltage_map(VDD1, 1.2);
  voltage_map(VDD2, 0.8);
  voltage_map(VSS, 0.0);
  operating_conditions(XYZ) {
    process : 1.0;
    voltage : 3.0;
    temperature : 25.0;
  }
  default_operating_conditions : XYZ;
  cell(Buffer_Type_HL_Level_shifter) {
    is_level_shifter : true;
    level_shifter_type : HL ;
    pg_pin(VDD1) {
      voltage_name : VDD1;
      pg_type : primary_power;
      std_cell_main_rail : true;
    }
    pg_pin(VDD2) {
      voltage_name : VDD2;
      pg_type : primary_power;
    }
    pg_pin(VSS) {
  ```
voltage_name : VSS;
pg_type : primary_ground;
}

leakage_power() {
  when : "!A";
  value : 1.5;
  related_pg_pin : VDD1;
}

leakage_power() {
  when : "!A";
  value : 2.7;
  related_pg_pin : VDD2;
}

pin(A) {
  direction : input;
  related_power_pin : VDD1;
  related_ground_pin : VSS;
  input_voltage_range ( 0.7 , 0.9);
}

pin(Z) {
  direction : output;
  related_power_pin : VDD2;
  related_ground_pin : VSS;
  function : "A";
  power_down_function : "!VDD1 + !VDD2 + VSS";
  output_voltage_range ( 1.1 , 1.3);

  timing() {
    related_pin : A;
    cell_rise(template) {
      ...
    }
    cell_fall(template) {
      ...
    }
    rise_transition(template) {
      ...
    }
    fall_transition(template) {
      ...
    }
  }

  internal_power() {
    related_pin : A;
    related_pg_pin : VDD1;
    ...
  }

  internal_power() {
    related_pin : A;
  }
Power-and-Ground Level-Shifter Cell

Figure 9-10 shows the block diagram of a power-and-ground level-shifter cell. The cell connects the PD1 and PD2 power domains. PD1 and PD2 have power supply voltages, VDD\_IN and VDD\_OUT, and ground voltages, VSS\_IN and VSS\_OUT, respectively.

Level-shifter cell insertion is supported for the following conditions:

- VDD\_IN \geq VDD\_OUT; VSS\_IN \geq VSS\_OUT
- VDD\_IN \geq VDD\_OUT; VSS\_IN \leq VSS\_OUT
- VDD\_IN \leq VDD\_OUT; VSS\_IN \geq VSS\_OUT
- VDD\_IN \leq VDD\_OUT; VSS\_IN \leq VSS\_OUT

Example 9-12 shows the Liberty model of a simple power-and-ground level-shifter cell.
operating_conditions(XYZ) {
  process : 1;
  temperature : 125;
  voltage : 0.8;
  tree_type : balanced_tree;
}
default_operating_conditions : XYZ;
...
cell(up_shift ){
  area : 1.0;
  is_level_shifter : true;
  level_shifter_type : LH;
  pg_pin(VDD_IN) {
    voltage_name : VDD_IN;
    pg_type : primary_power;
    std_cell_main_rail : true;
  }
  pg_pin(VDD_OUT) {
    voltage_name : VDD_OUT;
    pg_type : primary_power;
  }
  pg_pin(VSS_IN) {
    voltage_name : VSS_IN;
    pg_type : primary_ground;
    std_cell_main_rail : true;
  }
  pg_pin(VSS_OUT) {
    voltage_name : VSS_OUT;
    pg_type : primary_ground;
  }
  pin(IN) {
    direction : input;
    capacitance : 1.0;
    related_power_pin : VDD_IN;
    related_ground_pin : VSS_IN;
    input_voltage_range (0.8, 1.0);
    ground_input_voltage_range (0.4, 0.5);
  }
  pin(OUT) {
    direction : output;
    related_power_pin : VDD_OUT;
    related_ground_pin : VSS_OUT;
    function : "IN";
    power_down_function : "!VDD_IN + !VDD_OUT + 
      VSS_IN + ~VSS_OUT" ;
    output_voltage_range (1.0, 1.2);
    ground_output_voltage_range (0.0, 0.1);
    timing() {
      related_pin : IN;
      cell_rise(scalar) {
        values ("1.0") ;
      }
      rise_transition (scalar) {
Multibit Level-Shifter Cell

Figure 9-11 shows a multibit buffer-type level-shifter cell with four data input and four data output pins. Each of the outputs is a linear function of the respective input. The input data pins are powered by the PG pin, VDD1, and the output data pins are powered by the PG pin, VDD2. The example following the figure is a model of the level-shifter cell.
pg_pin (VDD1) {
    pg_type : primary_power;
    voltage_name : VDD1;
}
pg_pin (VDD2) {
    pg_type : primary_power;
    voltage_name : VDD2;
}
pg_pin (VSS) {
    pg_type : primary_ground;
    voltage_name : VSS;
}
...
pin (a1) {
    level_shifter_data_pin : true;
    related_power_pin : VDD1;
    related_ground_pin : VSS;
    ...
}
pin (a2) {
    level_shifter_data_pin : true;
    related_power_pin : VDD1;
    related_ground_pin : VSS;
    ...
}
pin (a3) {
    level_shifter_data_pin : true;
    related_power_pin : VDD1;
    related_ground_pin : VSS;
    ...
}
pin (a4) {
    level_shifter_data_pin : true;
    related_power_pin : VDD1;
    related_ground_pin : VSS;
    ...
}
pin (z1) {
    direction : output;
    related_power_pin : VDD2;
    related_ground_pin : VSS;
    power_down_function : !VDD1+!VDD2+VSS;
    function : a1;
    ...
}
pin (z2) {
    direction : output;
    related_power_pin : VDD2;
    related_ground_pin : VSS;
    power_down_function : !VDD1+!VDD2+VSS;
    function : a2;
    ...
}
Overdrive Level-Shifter Cell

The cell in Figure 9-12 is known as an overdrive level-shifter cell. To model an overdrive level-shifter cell, specify the `related_ground_pin` attribute and the `input_signal_level` attribute, as shown in the example that follows the figure.

Note:
The area of a multiple-rail level-shifter cell (as shown in Figure 9-9) is larger than that of an overdrive level-shifter cell (as shown in Figure 9-12). Keep the area in mind when you design your cell.

Figure 9-12  Overdrive Level-Shifter Cell

```plaintext
library(level_shifter_cell_library_example) {
  voltage_map(VDD1, 1.2);
  voltage_map(VDD, 0.8);
  voltage_map(VSS, 0.0);
  operating_conditions(XYZ) {
  }
} /* end cell */
```
process : 1.0;
voltage : 3.0;
temperature : 25.0;
}
default_operating_conditions : XYZ;

cell(Buffer_Type_HL_Level_shifter) {
is_level_shifter : true;
level_shifter_type : HL;

pg_pin(VDD) {
voltage_name : VDD;
pg_type : primary_power;
std_cell_main_rail : true;
}
pg_pin(VSS) {
voltage_name : VSS;
pg_type : primary_ground;
}
leakage_power() {
when : "!A";
value : 1.5;
related_pg_pin : VDD;
}

pin(A) {
direction : input;
/* Defining the input_signal_level attribute identifies an Overdrive Level Shifter cell */
input_signal_level : VDD1;
related_power_pin : VDD;
related_ground_pin : VSS;
input_voltage_range (1.1, 1.3);
}

pin(Z) {
direction : output;
related_power_pin : VDD;
related_ground_pin : VSS;
function : "A";
power_down_function : "!VDD + VSS";
output_voltage_range (0.6, 0.9);

timing() {
related_pin : A;
cell_rise(template) {
...
}
cell_fall(template) {
...
}
rise_transition(template) {
...}
Level-Shifter Cell With Virtual Bias Pins

This section provides an example of a level-shifter cell with virtual bias pins and two n-well regular wells for substrate-bias modeling.

```liberty
library (example_multi_rail_with_bias_pins) {
  cell ( level_shifter ) {
    pg_pin ( vdd1 ) {
      pg_type : primary_power ;
      ...
    }
    pg_pin ( vdd2 ) {
      pg_type : primary_power ;
      ...
    }
    pg_pin ( vss ) {
      pg_type : primary_ground ;
      ...
    }
    pg_pin ( vpw ) {
      pg_type : pwell ;
      ...
    }
    pg_pin ( vnw1 ) {
      pg_type : nwell ;
      ...
    }
    pg_pin ( vnw2 ) {
      pg_type : nwell ;
      ...
    }
    pin ( I ) {
      direction : input;
      related_power_pin : vdd1
      related_ground_pin : vss
    } /*end cell group*/
  } /*end cell group*/
} /*end library group */
```
related_bias_pin : "vnw1 vpw"
...
}
pin ( Z ) {
  direction : output;
  function : "I";
  related_power_pin : "vdd2";
  related_ground_pin : "vss";
  related_bias_pin : "vnw2 vpw";
  power_down_function : "!vdd1 + !vdd2 + vss + !vnw1 + !vnw2 + vpw";
...
} /* End of cell group */
} /* End of library group */

Enable Level-Shifter Cell

Figure 9-13 shows the schematic of a basic enable level-shifter cell. The A signal pin is linked to the VDD1 and VSS power and ground pin pair, and the EN signal pin and the Y signal pin are linked to the VDD2 and VSS power and ground pin pair. The enable pin is linked to VDD2.

The example that follows the figure shows a library containing an enable level-shifter cell. To prevent a short between VDD1 and VSS, pin A has the alive_during_power_up attribute set to true and is active till VDD1 is on.

Figure 9-13 Enable Level-Shifter Cell Schematic

library(enable_level_shifter_library_example) {

  voltage_map(VDD1, 0.8);
  voltage_map(VDD2, 1.2);
  voltage_map(VSS, 0.0);

  operating_conditions(XYZ) {
    voltage : 3.0;
    ...
  }
  default_operating_conditions : XYZ;
cell(Enable_Level_Shifter) {
  is_level_shifter : true;
  level_shifter_type : LH;
}

pg_pin(VDD1) {
  voltage_name : VDD1;
  pg_type : primary_power;
  std_cell_main_rail : true;
}

pg_pin(VSS) {
  voltage_name : VSS;
  pg_type : primary_ground;
}

pg_pin(VDD2) {
  voltage_name : VDD2;
  pg_type : primary_power;
}

leakage_power() {
  when : "!A";
  value : 1.5;
  related_pg_pin : VDD1;
}

leakage_power() {
  when : "!A";
  value : 1.5;
  related_pg_pin : VDD2;
}

pin(A) {
  direction : input;
  related_power_pin : VDD1;
  related_ground_pin : VSS;
  input_voltage_range ( 0.7 , 0.9);
  level_shifter_data_pin : true;
}

pin(EN) {
  direction : input;
  related_power_pin : VDD2;
  related_ground_pin : VSS;
  input_voltage_range ( 1.1 , 1.3);
  level_shifter_enable_pin : true;
}

pin(Y) {
  direction : output;
  related_power_pin : VDD2;
  related_ground_pin : VSS;
  function : "A * EN";
  power_down_function : "!VDD1 + !VDD2 + VSS";
output_voltage_range (1.1, 1.3);
timing()
{
    related_pin : "A EN";
    cell_rise(template) {
        ...
    }
    cell_fall(template) {
        ...
    }
    rise_transition(template) {
        ...
    }
    fall_transition(template) {
        ...
    }
}

internal_power()
{
    related_pin : "A EN";
    related_pg_pin : VDD1;
    ...
}
internal_power()
{
    related_pin : "A EN";
    related_pg_pin : VDD2;
    ...
}

} /* end pin group*/
...
} /* end cell group*/
....
} /* end library group*/

Clamping in Latch-Based Enable Level-Shifter Cell

Table 9-4 shows the truth table for a latch-based enable level-shifter cell with two enable pins, EN1 and EN2. The example that follows the table shows the enable level-shifter cell modeled with the clamp function attributes.

Table 9-4 Truth Table for a Latch-Based Enable Level-Shifter Cell With Two Enable Pins

<table>
<thead>
<tr>
<th>A</th>
<th>EN1</th>
<th>EN2</th>
<th>Y</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/0</td>
<td>1</td>
<td>0</td>
<td>1/0</td>
<td>Level shifter</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Clamp to 1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Qn-1</td>
<td>Latch</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>Forbidden</td>
</tr>
</tbody>
</table>
cell(clamp_1_latch_ELS) {
  is_level_shifter : true;
  ...
  pg_pin(VDD1) {
    pg_type : primary_power;
    ...
  }
  pg_pin(VSS1) {
    pg_type : primary_ground;
    ...
  }
  pg_pin(VDD2) {
    pg_type : primary_power;
    std_cell_main_rail : true;
    ...
  }
  pg_pin(VSS2) {
    pg_type : primary_ground;
    ...
  }
  latch (IQ, IQN) {
    data_in : A;
    enable : EN1;
    preset : EN2;
  }
  pin (A) {
    related_power_pin : VDD1;
    related_ground_pin : VSS1;
    direction : input;
    level_shifter_data_pin : true;
    ...
  }
  pin (EN1) {
    related_power_pin : VDD2;
    related_ground_pin : VSS2;
    direction : input;
    level_shifter_enable_pin : true;
    ...
  }
  pin (EN2) {
    related_power_pin : VDD2;
    related_ground_pin : VSS2;
    direction : input;
    level_shifter_enable_pin : true;
    ...
  }
  pin (Y) {
    related_power_pin : VDD2;
    related_ground_pin : VSS2;
    direction : output;
    clamp_1_function: "!EN1 * EN2";
    clampLatch_function: "!EN1 * !EN2";
    illegal_clamp_function: "EN1 * EN2";
Differential Level-Shifter Cell

Figure 9-14 shows a schematic of a low-to-high differential level-shifter cell with the complementary input pins, A and AN, and an enable pin, EN. The differential level-shifter cell connects the VDD1 and VDD2 power domains. The differential level-shifter cell uses the supply voltage, VDD2. The supply voltage, VDD1, drives the inputs on the pins, A and AN. The example that follows the figure shows a typical model of the differential level-shifter cell.

```liberty
function : "IQ";
...
}
...
}

cell(DLS) {
  is_level_shifter : true;
  pin_opposite ("A", "AN");
  contention_condition : "!(A^AN)*EN";
  ...
  pg_pin (VDD2) {
    pg_type : primary_power;
    std_cell_main-rail : true;
    ...
  }
  pg_pin (GND) {
    pg_type : primary_ground;
  }

Figure 9-14  Differential Level-Shifter Schematic
```
...}
}

pin (A) {
  related_power_pin : VDD2 ;
  related_ground_pin : GND ;
  direction : input;
  level_shifter_data_pin : true ;
  input_signal_level : VDD1 ;
  ...
}

pin (AN) {
  related_power_pin : VDD2 ;
  related_ground_pin : GND ;
  direction : input;
  level_shifter_data_pin : true ;
  input_signal_level : VDD2 ;

  timing(){
    related_pin : "A" ;
    timing_type : non_seq_setup_rising ;
    rise_constraint(DLS_temp) {
      ...
    }
    fall_constraint(template_name) {
      ...
    }
  }

  timing(){
    related_pin : "A" ;
    timing_type : non_seq_setup_falling ;
    rise_constraint(DLS_temp) {
      ...
    }
    fall_constraint(template_name) {
      ...
    }
  }

  timing(){
    related_pin : "A" ;
    timing_type : non_seq_hold_rising ;
    rise_constraint(DLS_temp) {
      ...
    }
    fall_constraint(template_name) {
      ...
    }
  }

  timing(){
    related_pin : "A" ;
    timing_type : non_seq_hold_falling ;
Isolation Cell Modeling

In partition-based designs with multiple power supplies and voltages, the outputs from a shut-down partition to an active partition must be maintained at predictable signal levels. An isolation cell isolates the shut-down partition from the active partition. The isolation logic ensures that all the inputs to the active partition are clamped to a fixed value.

Example 9-13 shows the syntax for modeling the isolation cell, and its pins.

**Example 9-13 Isolation Cell Modeling Syntax**

```plaintext
cell(isolation_cell) {
  is_isolation_cell : true ;
  ...
  pg_pin(pg_pin_name_P) {
    pg_type : primary_power ;
    permit_power_down : true ;
    ...
  }
  pg_pin(pg_pin_name_G) {
    pg_type : primary_ground;
    ...
  }
}
```
Cell-Level Attribute

This section describes a cell-level attribute of the isolation cells.

is_isolation_cell Attribute

The `is_isolation_cell` attribute identifies a cell as an isolation cell. The valid values of this attribute are `true` or `false`. If not specified, the default is `false`, meaning that the cell is an ordinary standard cell.

Pin-Level Attributes

This section describes the pin-level attributes for the isolation cells.

isolation_cell_data_pin Attribute

The `isolation_cell_data_pin` attribute identifies the data pin of an isolation cell. The valid values are `true` or `false`. If not specified, all the input pins of the isolation cell are considered to be data pins.
isolation_cell_enable_pin Attribute

The `isolation_cell_enable_pin` attribute identifies an enable or control pin of an isolation cell including a clock isolation cell. The valid values are `true` or `false`. The default is `false`.

power_down_function Attribute

The `power_down_function` attribute identifies the Boolean condition to switch off the output pin of the cell (when the cell is inactive due to the external power-pin states).

permit_power_down Attribute

The `permit_power_down` attribute indicates that the power pin can be powered down while in the isolation mode. The valid values are `true` or `false`. The default is `true`.

alive_during_partial_power_down Attribute

The `alive_during_partial_power_down` attribute indicates that the pin with this attribute is active while the isolation cell is partially powered down, and the corresponding power and ground rails are not considered as the power reference. The valid values are `true` and `false`. The default is `true`, and in the default setting, the UPF isolation supply set is the power reference.

alive_during_power_up Attribute

The optional `alive_during_power_up` attribute specifies an active data pin that is powered by a more always-on supply. The default is `false`. If set to `true`, it indicates that the data pin is active while its related power pin is active.

You can specify this attribute only in a pin group where the `isolation_cell_data_pin` or the `level_shifter_data_pin` attribute is set to `true`. 
Attribute Dependencies

The isolation cell attributes have the following dependencies:

- When the control pin of an isolation cell is activated, the output becomes a constant.

- The control pin of an isolation cell, that permits partial power down must be included in the Boolean expression of the `power_down_function` attribute for the same output. The control pin blocks the terms that use the powered-down rail, to set to `true`. To generate the output in the isolation mode, the active power rail is used.

Therefore, an isolation cell cannot be partially powered down if the `power_down_function` expression of its power pin does not include the `isolation_cell_enable_pin` attribute set.

Isolation Cell Examples

Unlike level-shifter cells, isolation cells cannot shift voltage levels. All other characteristics are the same between a level-shifter cell and an isolation cell.

Figure 9-15 shows a schematic and a library description of an isolation cell. The library describes only the portion related to the power and ground pin syntax. In the figure, the A, EN, and Y signal pins are associated to the VDD and VSS power and ground pin pair. The example shows a library with an isolation cell.

Figure 9-15  Isolation Cell Schematic

```
library(isolation_cell_library_example) {
  voltage_map(VDD, 1.0);
  voltage_map(VSS , 0.0);

  operating_conditions(XYZ) {
    voltage : 1.0;
    ...
  }
  default_operating_conditions : XYZ;
```
cell(Isolation_Cell) {
  is_isolation_cell : true;
  dont_touch := true;
  dont_use := true;
}

pg_pin(VDD) {
  voltage_name : VDD;
  pg_type : primary_power;
}

pg_pin(VSS) {
  voltage_name : VSS;
  pg_type : primary_ground;
}

leakage_power() {
  when : "!A";
  value : 1.5;
  related_pg_pin : VDD;
}

pin(A) {
  direction : input;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  isolation_cell_data_pin : true;
}

pin(EN) {
  direction : input;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  isolation_cell_enable_pin : true;
}

pin(Y) {
  direction : output;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  function : "A * EN";
  power_down_function : "!VDD + VSS";
  timing() {
    related_pin : "A EN";
    cell_rise(template) {
      ...
    }
    cell_fall(template) {
      ...
    }
    rise_transition(template) {
      ...
    }
    fall_transition(template) {
      ...
Multibit Isolation Cell

Figure 9-16 shows a multibit isolation cell with four data input pins, four enable pins for each of the data input pins, and four data output pins. Each of the data outputs is a linear function of the respective data input. The example following the figure is a model of the isolation cell.

**Figure 9-16  4-bit Isolation Cell**

```plaintext
... } internal_power() { related_pin : A;
related_pg_pin : VDD;
...
} /* end pin group*/
... } /*end cell group*/
... } /*end library group*/

cell (4_bit_iso) {
    VDD
    a1
    e1
    a2
    e2
    a3
    e3
    a4
    e4
    z1
    z2
    z3
    z4
    VSS
```
is_isolation_cell : true;
pg_pin (VDD) {
  pg_type : primary_power;
  voltage_name : VDD;
}
pg_pin (VSS) {
  pg_type : primary_ground;
  voltage_name : VSS;
}
...
pin (a1) {
  isolation_cell_data_pin : true;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  ...
}
pin (a2) {
  isolation_cell_data_pin : true;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  ...
}
pin (a3) {
  isolation_cell_data_pin : true;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  ...
}
pin (a4) {
  isolation_cell_data_pin : true;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  ...
}
pin (e1) {
  isolation_cell_enable_pin : true;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  ...
}
pin (e2) {
  isolation_cell_enable_pin : true;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  ...
}
pin (e3) {
  isolation_cell_enable_pin : true;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  ...
}
Clock-Isolation Cell Modeling

Using combinational isolation cells to isolate clock signals might result in phase-clipped outputs. Figure 9-17 shows an AND isolation cell. The phase of the output clock signal, CK_OUT, becomes clipped, depending on the arrival time of the isolation-enable signal, ISO_EN, with respect to the input clock signal, CK.
To avoid phase-clipping, use clock-isolation cells for clock isolation. Figure 9-18 shows a schematic and a typical output of a clock-isolation cell.

**Figure 9-18 Clock Isolation Cell Schematic**

**Note:**

The phase of the output clock signal, CK_OUT, is not clipped.

The modeling syntax of the clock-isolation cell and its pins is as follows:

```plaintext
cell (cell_name) {
    is_clock_isolation_cell : true;
    pin(input pin) {
        clock_isolation_cell_clock_pin : true;
        direction : input;
        ...
    }
    pin(input pin) {
        isolation_cell_enable_pin : true;
        direction : input;
        ...
    }
}
```
Cell-Level Attribute
This section describes the cell-level attribute of clock-isolation cells.

**is_clock_isolation_cell Attribute**
The `is_clock_isolation_cell` attribute identifies a cell as a clock-isolation cell. The default is false, meaning that the cell is a standard cell.

Pin-Level Attributes
This section describes the pin-level attributes for clock-isolation cells.

**isolation_cell_enable_pin Attribute**
The `isolation_cell_enable_pin` attribute identifies an enable or control pin of a clock-isolation cell. The default is false.

**clock_isolation_cell_clock_pin Attribute**
The `clock_isolation_cell_clock_pin` attribute identifies an input clock pin of a clock-isolation cell. The default is false.

Clock Isolation Cell Examples
Example 9-14 shows a library description of the clock-isolation cell in Figure 9-18 on page 9-88.

Example 9-14  Library Description of the Clock-Isolation Cell

```liberty
library (my_lib) {
    ...
    cell(ck_iso_cell) {
        is_clock_Isolation_cell : true;
        ...
        pg_pin (VDD) {
            pg_type : primary_power;
            voltage_name : VDD;
        }
    }
```

Example 9-15 shows the clock-isolation cell also modeled as a clock-gating cell. To model a cell as both a clock-isolation cell and a clock-gating cell, use both the clock-isolation and clock-gating attributes.

Example 9-15  Example of a Cell Modeled as Both Clock-Isolation and Clock-Gating Cell

```liberty
 cell(ICG_CK_ISO_CELL) {
    is_clock_isolation_cell : true;
    clock_gating_integrated_cell : "latch_posedge";
    ...
    pg_pin (VDD) {
        pg_type : primary_power;
        voltage_name : VDD;
    }
}
```
pg_pin (VSS) {
    pg_type : primary_ground;
    voltage_name : VSS;
}
statetable(" CK EN ", "Q ") {
    table : " L L : - : L ,
           L H : - : H ,
           H - : - : N ";
}
pin(CK) {
    clock_isolation_cell_clock_pin : true;
    clock_gate_clock_pin : true;
    direction : input;
    related_ground_pin : VSS;
    related_power_pin : VDD;
    ...
}
pin(EN) {
    isolation_cell_enable_pin : true;
    clock_gate_enable_pin : true;
    direction : input;
    related_ground_pin : VSS;
    related_power_pin : VDD;
    ...
}
pin(CK_OUT) {
    direction : output;
    clock_gate_output_pin : true;
    power_down_function : "!VDD + VSS";
    related_ground_pin : VSS;
    related_power_pin : VDD;
    state_function : "CK * Q";
    timing() {
        related_pin : "CK";
        timing_sense : "positive_unate";
        ...
    }
    ...
}

---

**Clamping Isolation Cell Output Pins**

In isolation cells with multiple enable pins, clamping the outputs might be required to maintain them at particular logic levels, such as 0, 1, z, or a previously-latched value.

The clamp function modeling syntax for the isolation cell and its pins is as follows:

cell (cell_name) {
    is_isolation_cell : true;
    ...
}
pin(input_pin) {
    direction : input;
    isolation_cell_data_pin : true;
    ...
}
pin(input_pin) {
    direction : input;
    isolation_cell_enable_pin : true;
    ...
}
pin(input_pin) {
    direction : input;
    isolation_cell_enable_pin : true;
    ...
}
pin(output_pin_name) {
    direction : output;
    clamp_0_function : "Boolean expression";
    clamp_1_function : "Boolean expression";
    clamp_z_function : "Boolean expression";
    clamp_latch_function : "Boolean expression";
    illegal_clamp_condition : "Boolean expression";
    ...
}
...
...

Pin-Level Attributes

This section describes the pin-level clamp function attributes to clamp the isolation cell output pins.

clamp_0_function Attribute

The clamp_0_function attribute specifies the input condition for the enable pins of an isolation cell when the output clamps to 0.

clamp_1_function Attribute

The clamp_1_function attribute specifies the input condition for the enable pins of an isolation cell when the output clamps to 1.

clamp_z_function Attribute

The clamp_z_function attribute specifies the input condition for the enable pins of an isolation cell when the output clamps to z.
clamp_latch_function Attribute

The `clamp_latch_function` attribute specifies the input condition for the enable pins of an isolation cell when the output clamps to the previously latched value.

illegal_clamp_condition Attribute

The `illegal_clamp_condition` attribute specifies the invalid condition for the enable pins of an isolation cell. If the `illegal_clamp_condition` attribute is not specified, the invalid condition does not exist.

Note:

- All the input pins used in the Boolean expressions of the `clamp_0_function`, `clamp_1_function`, `clamp_z_function`, and `clamp_latch_function` attributes must be enable pins with the `isolation_cell_enable_pin` attribute set to `true`. The Boolean expressions of the `clamp_0_function`, `clamp_1_function`, `clamp_z_function`, `clamp_latch_function`, and `illegal_clamp_condition` attributes must be mutually exclusive.

Example of Clamping in Isolation Cell

Table 9-4 shows the truth table for an isolation cell with two enable pins, EN1 and EN2. Example 9-16 shows the isolation cell modeled with a clamp function attribute.

<table>
<thead>
<tr>
<th>A</th>
<th>EN1</th>
<th>EN2</th>
<th>Y</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/1</td>
<td>1</td>
<td>1</td>
<td>0/1</td>
<td>AND</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Clamp to 0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Clamp to 0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Clamp to 0</td>
</tr>
</tbody>
</table>

Example 9-16 Using a Clamp Function Attribute to Model an Isolation Cell

```liberty
cell(ISO_clamp_0) {
  is_isolation_cell : true;
  ...
  pin(A) {
    direction : input;
    capacitance : 1.0;
    isolation_cell_data_pin : true;
    ...
  }
```
Isolation Cells With Multiple Control Pins

In partition-based designs, output pins of an isolation-cell need to be maintained at predefined levels for a significant period of time. In such designs, using isolation cells with multiple control or enable pins reduces the leakage current. Multiple control pins allow an isolation cell to be partially powered down. For example, in an isolation cell with two control pins, one control pin controls the output level. The other control pin is used to partially power-down the isolation cell while the output level is maintained.

Figure 9-19 shows the gate-level diagrams of isolation cells with multiple control pins. In each of the three diagrams, the dotted-line arrow traces the path from the isolation-control pin to the output pin.
Figure 9-19  Gate-Level Diagrams of Isolation Cells With Multiple Control Pins

The isolation control pins permit the power pins (not shown in Figure 9-19) to power down. The `alive_during_partial_power_down` attribute on the isolation-control pins indicate that these pins remain active even when the corresponding power pins are powered down. Example 9-17 shows the partially powered down model of the cell, ISO1, depicted in Figure 9-19. Table 9-6 shows the Boolean expressions for the `function` and `power_down_function` attributes of the isolation cells depicted in Figure 9-19.

Example 9-17  Partially Powered Down Model of the ISO1 Cell

```plaintext
cell ( ISO1 ) {
  is_isolation_cell : true;
  pg_pin ( VDD ) {
    pg_type : primary_power;
    permit_power_down : true
  }
  pg_pin ( VSS ) {
```
\documentclass{article}
\usepackage{listings}
\usepackage{amsmath}
\usepackage{tabularx}

\begin{document}

\begin{verbatim}
pg_type : primary_ground;
}
pin (D) {
  isolation_cell_data_pin : true;
}
pin (EN) {
  direction : input
  isolation_cell_enable_pin : true;
  alive_during_partial_power_down : true;
}
pin (Y) {
  direction : output
  alive_during_partial_power_down : true;
  function : D * \overline{EN}
  power_down_function : (\overline{VDD} * \overline{EN}) + VSS
}
}

Note:
By default, the \texttt{related_power_pin} and \texttt{related_ground_pin} attributes are mapped to VDD and VSS, respectively. The \texttt{related_power_pin} and \texttt{related_ground_pin} attributes are not specified for the input and output pins in Example 9-17.

\begin{table}[h]
\centering
\caption{Boolean Expressions for the function and power\_down\_function Attributes of the Isolation Cells in Figure 9-19}
\begin{tabularx}{\textwidth}{|c|c|c|c|}
\hline
Cell & function (Output Y) & power\_down\_function (Output Y) \\
\hline
ISO1 & D * \overline{EN} & (\overline{VDD} * \overline{EN}) + VSS \\
ISO2 & D * \overline{EN1} * \overline{EN2} & (\overline{VDD} * \overline{EN2}) + VSS \\
ISO3 & D + \overline{EN1} + EN2 & \overline{VDD} + (VSS + EN1) \\
\hline
\end{tabularx}
\end{table}

Pins with the \texttt{alive_during_partial_power_down} attribute do not require the power pins to be active. For example, for the cell ISO2, the isolation-control pin, EN2, does not require VDD to be active. This pin controls the power-down of VDD through the Boolean expression of the \texttt{power\_down\_function} attribute as shown in Table 9-6. The other control pin, EN1, without the \texttt{alive_during_partial_power_down} attribute requires both VDD and VSS to be active, and is not included in the Boolean expression of the \texttt{power\_down\_function} attribute.

In each of the isolation cells in Figure 9-19, the \texttt{permit_power_down} attribute is set on a power pin when there is at least one pin with the \texttt{isolation\_cell\_enable\_pin} attribute. For example, for the cell ISO2, the \texttt{permit_power_pin} attribute is set on the power pin, VDD, given the \texttt{isolation\_cell\_enable\_pin} attribute is set on the pin, EN2.

\end{verbatim}

\end{document}
Switch Cell Modeling

Switch cells, also known as multithreshold complementary metal oxide semiconductor cells (power-switch cells), are used to reduce power. They are divided into the following two classes:

• Coarse grain

  There are two types of coarse-grain switch cells, header switch cells, and footer switch cells. The header switch cells control the power nets based on a PMOS transistor, and the footer switch cells control the ground nets based on an NMOS transistor. The coarse grain cell is a switch that drives the power to other logic cells. It is used as a big switch to the supply rails and to turn off design partitions when the relative logic is inactive. Therefore, coarse-grain switch cells can reduce the leakage of the inactive logic.

  In addition, coarse-grain switch cells can also have the properties of a fine-grain switch cell. For example, they can act as a switch and have output pins that might or might not be shut off by the internal switch pins.

• Fine grain

  To reduce the leakage power, the fine-grain switch cell includes an embedded switch pin that is used to turn off the cell when it is inactive.

  The syntax supports only fine-grain switch cells for macro cells.

Coarse-Grain Switch Cells

Coarse-grain switch cells must have the following properties:

• They must be able to model the condition under which the cell turns off the controlled design partition or the cells connected in the output power pin’s fanout logical cone. This is modeled with a switching function based on special switch signal pins as well as a separate but related power-down function based on power pin inputs.

• They must be able to model the “acknowledge” output pins (output pins whose signal is used to propagate the switch signal to the next-switch cell or to a power controller input's logic cloud). Timing is also propagated from the input switch pin to the acknowledge output pins.

• They must have at least one virtual output power and ground pin (virtual VDD or virtual VSS), one regular input power and ground pin (VDD or VSS), and one switch input pin. There is no limit on the number of pins a coarse-grain cell can have.

  The following describes a simple coarse-grain switch header cell and a simple coarse-grain switch footer cell:
Header cell: one switch input pin, one VDD (power) input power and ground pin, and one virtual VDD (internal power) output power and ground pin

Footer cell: one switch input pin, one virtual VSS (internal ground) output power and ground pin, and one VSS (ground) input power and ground pin

- They can have multiple switch pins and multiple acknowledge output pins.
- They must have the steady state current (I/V) information to determine the resistance value when the switch is on.
- The timing information can be specified for coarse-grain switch cells on the output pins, and it can be state dependent for switch pins.

The power output pins in a coarse-grain switch cell can have the following two states:

- Awake or On
  In this state, the input power level is transmitted through the cell to either a 1 or 0 on the output power pin, depending on other prior switch cells in series settings.
- Off
  In this state, the sleep pin deactivates the pass-through function, and the output power pin is set to X.

Coarse-Grain Switch Cell Syntax

The following syntax is a portion of the coarse-grain switch cell syntax:

```plaintext
library(coarse_grain_library_name) {
...

lu_table _template (template_name)
  variable_1 : input_voltage;
  variable_2 : output_voltage;
  index_1 ( float, ... );
  index_2 ( float, ... );
}
...
cell(cell_name) {
  switch_cell_type : coarse_grain;
  ...
  pg_pin (VDD/VSS pin name) {
    pg_type : primary_power | primary_ground;
    direction : input ;
    ...
  }
}

/* Virtual power and ground pins use "switch_function" to describe the logic to shut off the attached design partition */

pg_pin (virtual VDD/VSS pin name) {
```
pg_type : internal_power | internal_ground;
direction: output;
...
switch_function : "function_string";
pg_function : "function_string";

}
dc_current (dc_current_name) {
    related_switch_pin : input_pin_name;
    related_pg_pin : VDD pin name;
    related_internal_pg_pin : Virtual VDD;

    values("float, ...");
}
pin (input_pin_name) {
    direction : input;
    switch_pin : true;
}
...
/* The acknowledge output pin uses "function" to represent the propagated switching signal */

pin(acknowledge_output_pin_name) {
    ...
    function : "function_string";
    power_down_function : "function_string";
    direction : output;
    ...
} /* end pin group */
} /* end cell group */

You can use the following syntax for intrinsic parasitic models.

switch_cell_type : coarse_grain;
dc_current (template_name) {
    related_switch_pin : pin_name;
    related_pg_pin : pg_pin_name;
    related_internal_pg_pin : pg_pin_name;
    index_1 ( "float, ..." );
    index_2 ( "float, ..." );
    values ( "float, ..." );
}
Library-Level Group

The following attribute is a library-level attribute for switch cells.

**lu_table_template Group**

The library-level `lu_table_template` group models the templates for the steady state current information that is used within the `dc_current` group. The `input_voltage` value specifies input voltage values for the switch pin. The `output_voltage` value specifies the output voltage values of the switch pin. The `input_voltage` and `output_voltage` values are the absolute gate voltage and absolute drain voltage, respectively, when a CMOS transistor is used to model a power-switch cell.

The `dc_current` group, which is used for steady state current modeling, must be defined at the cell level. It is defined by two indexes: `index_1` and `index_2`. The `index_1` attribute represents a string that includes a comma-separated set of N values, where N represents the table rows. The values define the voltage levels measured at either the input voltage or the output voltage. When referring to the input voltage, the voltage level is measured at the related switch pin, and the set of N values must have at least two values for N. When referring to the output voltage, the voltage level is measured at the related internal PG pin, and the set of N values must have at least three values for N. This voltage level is related to a common reference ground for the cell. The set of N `index_1` values must increase monotonically.

The `index_2` attribute represents a string that includes a comma-separated set of M values, where M represents the table columns. The values define the voltage levels that are specified at either the input voltage or the output voltage. When referring to the input voltage, the voltage level is measured at the related switch pin, and the set of M values must have at least two values for M. When referring to the output voltage, the voltage level is measured at the related internal PG pin, and the set of M values must have at least three values for M. This voltage level is related to a common reference ground for the cell. The set of M `index_2` values must increase monotonically.

Cell-Level Attribute and Group

The following cell-level attribute and group apply to coarse-grain switch cells.

**switch_cell_type Attribute**

The `switch_cell_type` attribute specifies the switch cell type so that it is not indirectly derived from the cell modeling description. Valid values are `coarse_grain` and `fine_grain`. 
**dc_current Group**

The cell-level `dc_current` group models the steady state current information, similar to the `lu_table_template` group. The table is used to specify the DC current through the cell’s output pin (generally the `related_internal_pg_pin`) in the current units specified at the library level using the `current_unit` attribute.

The `dc_current` group includes the `related_switch_pin`, `related_pg_pin`, and `related_internal_pg_pin` attributes, which are described in the following sections.

**related_switch_pin Attribute**

The `related_switch_pin` string attribute specifies the name of the related switch pin for the coarse-grain switch cell.

**related_pg_pin Attribute**

The `related_pg_pin` string attribute is used to specify the name of the power and ground pin that represents the VDD or VSS power source.

**related_internal_pg_pin Attribute**

The `related_internal_pg_pin` string attribute is used to specify the name of the power and ground pin that represents the virtual VDD or virtual VSS power source.

**Pin-Level Attributes**

The following attributes are pin-level attributes for coarse-grain switch cells.

**switch_function Attribute**

The `switch_function` string attribute identifies the condition when the attached design partition is turned off by the input `switch_pin`.

For a coarse-grain switch cell, the `switch_function` attribute can be defined at both controlled power and ground pins (virtual VDD and virtual VSS for `pg_pin`) and the output pins. It identifies the signal pins that can turn the power pin on.

When the `switch_function` attribute is defined in the controlled power and ground pin, it is used to specify the Boolean condition under which the cell switches off (or drives an X to) the controlled design partitions, including the traditional signal input pins only with no related power pins to this output.

**switch_pin Attribute**

The `switch_pin` attribute is a pin-level Boolean attribute. When it is set to true, it is used to identify the pin as the switch pin of a coarse-grain switch cell.
function Attribute

The function attribute in a pin group defines the value of an output pin or inout pin in terms of the input pins or inout pins in the cell group or model group. The function attribute describes the Boolean function of only nonsleep input signal pins.

pg_function Attribute

The pg_function attribute specifies the Boolean function of a virtual PG pin as a function of the cell's actual input power and ground pins. Specify the pg_function attribute in the pg_pin group.

Note:

- Use the pg_function attribute to model:
  - The internal and virtual power output pins that propagate power to the coarse-grain and fine-grained switches (as a function of the actual input power and ground pins).
    
    For more information about modeling fine-grained switch cells, see “Fine-Grained Switch Support for Macro Cells” on page 9-102.
  
  - The PG pins of internal voltage converters of macro cells.
    
    Do not use the switch_cell_type and switch_function attributes to model macro cells with internal voltage converters.
    
    For more information about modeling macro cells with internal PG pins, see “Modeling Macro Cells With Internal PG Pins” on page 9-147.

power_down_function Attribute

The power_down_function string attribute is used to identify the condition under which the cell's signal output pin is switched off (when the cell is in off mode due to the external power pin states).

pg_pin Group

The cell-level pg_pin group is used to model the VDD and VSS pins and virtual VDD and VSS pins of a coarse-grain switch cell. The syntax is based on the Y-2006.06 power and ground pin syntax.

Fine-Grained Switch Support for Macro Cells

A macro cell with a fine-grained switch is a cell that contains a special switch transistor with a control pin that can turn off the power supply of the cell when it is idle. This significantly lowers the power consumption of a design.
With the growing popularity of low-power designs, macro cells with fine-grained switches play an important role. The syntax identifies a cell as a macro cell and specifies the correct power pin that supplies power to each signal pin.

**Macro Cell With Fine-Grained Switch Syntax**

```plaintext
cell(cell_name) {
  is_macro_cell : true;
  switch_cell_type : coarse_grain | fine_grain;

  pg_pin (power/ground pin name) {
    pg_type : primary_power | primary_ground | backup_power | backup_ground;
    direction: input | inout | output;
    ...
  }

  /* This is a special pg pin that uses "switch_function" to describe the logic to shut off the attached design partition */
  pg_pin (internal power/ground pin name) {
    direction: internal | input | output | inout;
    pg_type : internal_power | internal_ground;
    switch_function : "function_string";
    pg_function : "function_string";
    ...
  }

  pin (input_pin_name) {
    direction : input | inout;
    switch_pin : true | false;
    ...
  }

  ...

  pin(output_pin_name) {
    direction : output | inout;
    power_down_function : "function_string";
    ...
  } /* end pin group */
} /* end cell group */
```

**Cell-Level Attributes**

The following attributes are cell-level attributes for macro cells with fine-grained switches.

**is_macro_cell Attribute**

The `is_macro_cell` simple Boolean attribute identifies whether a cell is a macro cell. If the attribute is set to `true`, the cell is a macro cell. If it is set to `false`, the cell is not a macro cell.
**switch_cell_type Attribute**

The `switch_cell_type` attribute is enhanced to support macro cells with internal switches. The valid enumerated values for this attribute are `coarse_grain` and `fine_grain`.

**pg_pin Group**

The following attribute can be specified under the `pg_pin` group for macro cells with fine-grained switches.

**direction Attribute**

The `direction` attribute supports `internal` as a valid value for macros when the internal power and ground is not visible or accessible at the cell boundary.

---

**Switch-Cell Modeling Examples**

The following sections provide examples for a simple coarse-grain header switch cell and a complex coarse-grain header switch cell.

**Simple Coarse-Grain Header Switch Cell**

*Figure 9-20* and the example that follows it show a simple coarse-grain header switch cell.

*Figure 9-20  Simple Coarse-Grain Header Switch Cell*

```plaintext
library (simple_coarse_grain_lib) {
  ...
  current_unit : 1mA;
  ...
```
```liberty
voltage_map(VDD, 1.0);
voltage_map(VVDD, 0.8);
voltage_map(VSS, 0.0);

operating_conditions(XYZ) {
  process : 1.0;
  voltage : 1.0;
  temperature : 25.0;
}
default_operating_conditions : XYZ;

lu_table_template (ivt1) {
  variable_1 : input_voltage;
  variable_2 : output_voltage;
  index_1 ("0.1, 0.2, 0.4, 0.8, 1.0");
  index_2 ("0.1, 0.2, 0.4, 0.8, 1.0");
}

... cell (Simple_CG_Switch) {
...
  switch_cell_type : coarse_grain;

  pg_pin (VDD) {
    pg_type : primary_power;
    direction : input;
    voltage_name : VDD;
  }

  pg_pin (VVDD) {
    pg_type : internal_power;
    voltage_name : VVDD;
    direction : output;
    switch_function : "SLEEP";
    pg_function : "VDD";
  }

  pg_pin (VSS) {
    pg_type : primary_ground;
    direction : input;
    voltage_name : VSS;
  }

  /* I/V curve information */
  dc_current (ivt1) {
    related_switch_pin : SLEEP; /* control pin */
    related_pg_pin : VDD; /* source */
    related_internal_pg_pin : VVDD; /* drain */
    values("0.010, 0.020, 0.030, 0.030, 0.030", "0.011, 0.021, 0.031, 0.041, 0.051", "0.012, 0.022, 0.032, 0.042, 0.052", "0.013, 0.023, 0.033, 0.043, 0.053", "0.014, 0.024, 0.034, 0.044, 0.054");
```

Chapter 9: Advanced Low-Power Modeling
Switch Cell Modeling
Complex Coarse-Grain Header Switch Cell

Figure 9-21 and the example that follows it show a complex coarse-grain header switch cell.

Figure 9-21  Complex Coarse-Grain Header Switch Cell

```liberty
}
default_operating_conditions : XYZ;

lu_table_template ( ivt1 ) {
    variable_1 : input_voltage;
    variable_2 : output_voltage;
    index_1 ( "0.1, 0.2, 0.4, 0.8, 1.0" );
    index_2 ( "0.1, 0.2, 0.4, 0.8, 1.0" );
}

... cell ( Complex.CG_Switch ) {
    ... switch_cell_type : coarse_grain;
    pg_pin ( VDD ) {
        pg_type : primary_power;
        voltage_name : VDD;
        direction: input ;
    }
    pg_pin ( VVDD ) {
        pg_type : internal_power;
        direction : output ;
        voltage_name : VVDD;
        switch_function : "SLEEP";
        pg_function : "VDD" ;
    }
    pg_pin ( VSS ) {
        pg_type : primary_ground;
        voltage_name : VSS;
        direction : input ;
    }

    /* I/V curve information */
    dc_current ( ivt1 ) {
        related_switch_pin : SLEEP; /* control pin */
        related_pg_pin : VDD; /* source power pin */
        related_internal_pg_pin : VVDD; /* drain internal power pin*/
        values(     "0.010, 0.020, 0.030, 0.040, 0.050",
                    "0.011, 0.021, 0.031, 0.041, 0.051",
                    "0.012, 0.022, 0.032, 0.042, 0.052",
                    "0.013, 0.023, 0.033, 0.043, 0.053",
                    "0.014, 0.024, 0.034, 0.044, 0.054" );
    }

    pin ( SLEEP ) {
        direction : input;
        switch_pin : true;
        capacitance: 1.0;
        related_power_pin : VDD;
        related_ground_pin : VSS;
    }
    ... pin (Y) {
```

Chapter 9: Advanced Low-Power Modeling
Switch Cell Modeling
direction : output;
function : "SLEEP";
related_power_pin : VDD;
related_ground_pin : VSS;
power_down_function : "!VDD + VSS";
timing() {
  related_pin : SLEEP;
  ...
}
/* end pin group */
} /* end cell group*/
} /* end library group*/

Complex Coarse-Grain Switch Cell With an Internal Switch Pin

Figure 9-22 and the example that follows it show a complex coarse-grain switch cell with an internal switch pin.

Figure 9-22  Complex Coarse-Grain Switch Cell With an Internal Switch Pin

library (Complex_CG_lib) {
  ... 
  current_unit : 1mA;
  ...

  voltage_map(VDD, 1.0);
  voltage_map(VVDD, 0.8);
  voltage_map(VSS, 0.0);

  operating_conditions(XYZ) {

process : 1.0;
voltage : 1.0;
temperature : 25.0;
}
default_operating_conditions : XYZ;

lu_table_template ( ivt1 ) {
variable_1 : input_voltage;
variable_2 : output_voltage;
index_1 ("0.1, 0.2, 0.4, 0.8, 1.0");
index_2 ("0.1, 0.2, 0.4, 0.8, 1.0");
}

...  

cell (COMPLEX_HEADER_WITH_INTERNAL_SWITCH_PIN) {
cell_footprint : complex_mtpmos;
area : 1.0;
switch_cell_type : coarse_grain;
pg_pin(VDD) {
voltage_name : VDD;
p_type : primary_power;
direction : input;
}
pg_pin(VVDD) {
voltage_name : VVDD;
p_type : internal_power;
direction : output;
switch_function : "SLEEP";
}
pg_pin(VSS) {
voltage_name : VSS;
p_type : primary_ground;
direction : input;
}

dc_current(ivt1) {
related_switch_pin : internal;
related_pg_pin : VDD;
related_internal_pg_pin : VVDD;
values("0.010, 0.020, 0.030, 0.040, 0.050",
"0.011, 0.021, 0.031, 0.041, 0.051",
"0.012, 0.022, 0.032, 0.042, 0.052",
"0.013, 0.023, 0.033, 0.043, 0.053",
"0.014, 0.024, 0.034, 0.044, 0.054");
}

pin(SLEEP) {
switch_pin : true;
direction : input;
capacitance : 1.0;
related_power_pin : VDD;
related_ground_pin : VSS;
...
Complex Coarse-Grain Switch Cell With Parallel Switches

Figure 9-23 and the example that follows it show a complex coarse-grain switch cell with two parallel switches.
Figure 9-23 Complex Coarse-Grain Switch Cell With Two Parallel Switches

library (Complex_CG_lib) {
    ...
    current_unit : 1mA;
    ...

    voltage_map(VDD, 1.0);
    voltage_map(VVDD, 0.8);
    voltage_map(VSS, 0.0);

    operating_conditions(XYZ) {
        process : 1.0;
        voltage : 1.0;
        temperature : 25.0;
    }
    default_operating_conditions : XYZ;

    lu_table_template ( ivt1 ) {
        variable_1 : input_voltage;
        variable_2 : output_voltage;
        index_1 ("0.1, 0.2, 0.4, 0.8, 1.0");
        index_2 ("0.1, 0.2, 0.4, 0.8, 1.0");
    }
    ...

    cell (COMPLEX_HEADER_WITH_TWO_PARALLEL_SWITCHES) {
        cell_footprint : complex_mtpmos;
        area : 1.0;
        switch_cell_type : coarse_grain;
    }
}
pg_pin(VDD) {
    voltage_name : VDD;
    pg_type : primary_power;
    direction : input;
}
pg_pin(VVDD) {
    voltage_name : VVDD;
    pg_type : internal_power;
    direction : output;
    switch_function : "CTL1 & CTL2";
}
pg_pin(VSS) {
    voltage_name : VSS;
    pg_type : primary_ground;
    direction : input;
}
dc_current(ivt1) {
    related_switch_pin : CTL1;
    related_pg_pin : VDD;
    related_internal_pg_pin : VVDD;
    values(  "0.010, 0.020, 0.030, 0.040, 0.050",
            "0.011, 0.021, 0.031, 0.041, 0.051",
            "0.012, 0.022, 0.032, 0.042, 0.052",
            "0.013, 0.023, 0.033, 0.043, 0.053",
            "0.014, 0.024, 0.034, 0.044, 0.054")
}
dc_current(ivt1) {
    related_switch_pin : CTL2;
    related_pg_pin : VDD;
    related_internal_pg_pin : VVDD;
    values(  "0.010, 0.020, 0.030, 0.040, 0.050",
            "0.011, 0.021, 0.031, 0.041, 0.051",
            "0.012, 0.022, 0.032, 0.042, 0.052",
            "0.013, 0.023, 0.033, 0.043, 0.053",
            "0.014, 0.024, 0.034, 0.044, 0.054")
}
pin(CTL1) {
    switch_pin : true;
    direction : input;
    capacitance : 1.0;
    related_power_pin : VDD;
    related_ground_pin : VSS;
    ...
}
pin(CTL2) {
    switch_pin : true;
    direction : input;
    capacitance : 1.0;
    related_power_pin : VDD;
Retention Cell Modeling

Some elements of an electronic design can store the logic state of the design. This is required for the design to wake up in the same state in which it was shut down. The retention cell is one such design element.

Retention cells are sequential cells that hold their state when the power supply is shut down and restore this state when the power is brought up again. The retention cell or register consists of a main register and a shadow register that has a different power supply. The power supply to the shadow register is always powered on to maintain the memory of the state. Therefore, the shadow register has high threshold-voltage transistors to reduce the leakage power. The main register has low threshold-voltage transistors for performance during the normal operation of the retention cell.

Retention cells are broadly classified into the store-in-place and balloon structures. Figure 9-24 shows the two main retention cell structures. The store-in-place structure includes a master-slave latch where either the slave or the master latch stores the state of the cell when the master-slave latch is shut down. In this structure, the master-slave latch implements the main register while the latch that stores the state of the cell implements the
shadow register. The balloon structure includes a flip-flop or master-slave latch and a balloon latch with a control logic that stores the state of the cell when the master-slave latch is shut down. In this structure, the master-slave latch implements the main register and the balloon latch implements the shadow register. The control logic includes signals, such as save and restore signals that control the data storage in the shadow register and the data transfer between the shadow register and the main register.
Figure 9-24  Retention Cell Structures

Retention Cell: Data Stored “In Place”

- hold_state value: “L”
  - Master Latch
  - Slave Latch

- hold_state value: “N”
  - Master Latch
  - Slave Latch

- hold_state value: “H”
  - Master Latch
  - Slave Latch

Retention Cell With SAVE and RESTORE Pins: Data Stored In Balloon Latch

- Balloon Latch
  - RESTORE
  - SAVE
  - Master Latch
  - Slave Latch

Latch on primary power; it is powered down

Latch on standby power

Chapter 9: Advanced Low-Power Modeling
Retention Cell Modeling
Modes of Operation

A retention cell has two modes of operation: normal mode and retention mode. The retention mode has three stages: save event, sleep mode, and restore event. For example, for the balloon structure, the master-slave latch operates in normal mode and the balloon latch operates in retention mode. The master-slave latch is considered to be edge-triggered. The normal and retention modes are described as follows:

- normal mode
  In this mode, the retention cell is fully powered on and the master-slave latch operates normally. The balloon latch does not add to the load at the output of the retention cell.

- retention mode
  - save event
    During this event, the current state of the master-slave latch is saved before the power to the latch is shut down. The balloon latch is considered to be edge-triggered during the save event. The trailing edge of the save control signal is the triggering edge.

  - sleep mode
    In this mode, the master-slave latch is shut down.

  - restore event
    During the restore event, a wake-up signal activates the master-slave latch and the data stored in the balloon latch is fed back to the master-slave latch. The state of the master-slave latch is restored just after the power is restored and before the retention cell operation returns to normal mode. The balloon latch is also considered to be edge-triggered during the restore event. The two methods to restore the state are:
      - The leading edge of the restore signal is the triggering edge for the balloon latch. The master-slave latch becomes transparent, that is, it does not latch the data. However, it outputs the same state that was saved in the balloon latch.
      - The trailing edge is the triggering edge for the balloon latch. The data is fully restored from the balloon latch and the flip-flop starts operating normally. In this method, the restored data is available for a shorter time.

Retention Cell Modeling Syntax

The following syntax shows the modeling of retention cells. The reference_input attribute defines the connectivity information for the input pins based on the reference_pin_names variable. The reference_pin_names variable specifies the internal reference input nodes used within the ff, latch, ff_bank, and latch_bank groups.
The sequential components of a retention cell are defined by using the flip-flop and latch syntax. The syntax to model the scan retention cells is identical to the syntax to model the scan sequential components. All scan retention cell functional models have a regular cell function that includes the scan pins as part of the function, while the test_cell group models the nonscan functionality of the retention cells with the scan pins that have been specified with the signal_type attributes.

```liberty
cell(cell_name) {
    retention_cell : retention_cell_style;

    pg_pin (primary_power_name) {
        voltage_name : primary_power_name;
        pg_type : primary_power;
    }

    pg_pin (primary_ground_name) {
        voltage_name : primary_ground_name;
        pg_type : primary_ground;
    }

    pg_pin (backup_power_name) {
        voltage_name : backup_power_name;
        pg_type : backup_power;
    }

    pg_pin (backup_ground_name) {
        voltage_name : backup_ground_name;
        pg_type : backup_ground;
    }

    pin(pin_name) {
        retention_pin(pin_class, disable_value);
        related_ground_pin : backup_ground;
        related_power_pin : backup_power;
        save_action : L|H|R|F;
        restore_action : L|H|R|F;
        restore_edge_type : edge_trigger | leading | trailing;
        ... 
    }

    retention_condition() {
        power_down_function: "Boolean_function";
        required_condition: "Boolean_function";
    }

    clock_condition() {
        clocked_on : "Boolean_expression";
        required_condition : "Boolean_expression";
        hold_state : L|H|N;
        clocked_on_also : "Boolean_expression";
        required_condition_also : "Boolean_expression";
        hold_state_also : L|H|N;
    }

    preset_condition() {
        input : "Boolean_expression";
        required_condition : "Boolean_expression";
    }
}
```
clear_condition() {
    input : "Boolean_expression";
    required_condition : "Boolean_expression";
}

pin(pin_name) {
    direction : inout | output | internal;
    function : Boolean_equation_with_internal_node_name;
    reference_input : pin_names;
    ...
}

bus(bus_name) {
    bus_type : bus_type_name;
    direction : inout | output | internal;
    function : Boolean_equation_with_internal_node_name;
    reference_input : pin_names;
    ...
}

ff ("reference_pin_names",] variable1, variable2 ) {
    power_down_function : "Boolean_expression";
    ...
}

latch ("reference_pin_names",] variable1, variable2 ) {
    power_down_function : "Boolean_expression";
    ...
}

ff_bank ("reference_pin_names",] variable1, variable2, bits) {
    power_down_function : "Boolean_expression";
    ...
}

latch_bank ("reference_pin_names",] variable1, variable2, bits) {
    power_down_function : "Boolean_expression";
    ...
}

...
Cell-Level Attributes, Groups, and Variables

This section describes the cell-level attributes, groups, and variables for retention cell modeling.

retention_cell Simple Attribute

The retention_cell attribute identifies the type of the retention cell or register. For a given cell, there can be multiple types of retention cells that have the same function in normal mode but different sleep signals, wake signals, or clocking schemes. For example, if a D flip-flop supports two retention strategies, such as type1 (where the data is transferred when the clock is low) and type2 (where the data is transferred when the clock is high), the values of the retention_cell attribute are different, such as DFF_type1 and DFF_type2.

ff, latch, ff_bank, and latch_bank Groups

The ff, latch, ff_bank, and latch_bank groups define sequential blocks. Define these groups at the cell level. You can specify one or more of these groups within a cell group.

retention_condition Group

The retention_condition group is a group of attributes that specify the conditions for the retention cell to hold its state during the retention mode. The retention_condition group includes the power_down_function and required_condition attributes.

power_down_function Attribute

The power_down_function attribute specifies the Boolean condition for the retention cell to be powered down—that is, the primary power to the cell is shut down. When this Boolean condition evaluates to true, it triggers the evaluation of the control input conditions specified by the required_condition attribute.

required_condition Attribute

The required_condition attribute specifies the control input conditions during the retention mode. For example, in Figure 9-29 on page 9-129, the retention signal, RET, is low during the retention mode. These conditions are checked when the Boolean condition specified by the power_down_function attribute evaluates to true. If these conditions are not met, the cell is considered to be in an illegal state.

Note:

Within the retention_condition group, the power_down_function attribute by itself does not specify the retention mode of the cell. The conditions specified by the required_condition attribute ensure that the retention control pin is in the correct state when the primary power to the cell is shut down.
The `clock_condition` group is a group of attributes that specify the conditions for correct signal during clock-based events. The `clock_condition` group includes two classes of attributes: attributes without the _also suffix and attributes with the _also suffix. These are similar to the `clocked_on` and `clocked_on_also` attributes of the `ff` group.

### clocked_on and clocked_on_also Attributes

The `clocked_on` and `clocked_on_also` attributes specify the active edge of the clock signal. The Boolean expression of the `clocked_on` attribute must be identical to the one specified in the `clocked_on` attribute of the corresponding `ff` or `ff_bank` group.

For example, for a master-slave latch, use the `clocked_on` attribute on the clock to the master latch and the `clocked_on_also` attribute on the clock to the slave latch.

**Note:**

A single-stage flip-flop or latch does not use the `clocked_on_also` attribute.

### required_condition and required_condition_also Attributes

The `required_condition` and `required_condition_also` attributes specify the input conditions during the active edge of the clock signal. These conditions are checked, respectively, at the values specified by the `clocked_on` and `clocked_on_also` attributes. If any one of the conditions are not met, the cell is considered to be in an illegal state.

For example, when the `required_condition` attribute is checked at the rising edge of the clock signal specified by the `clocked_on` attribute, the `required_condition_also` attribute is also checked at the rising edge of the clock signal specified by the `clocked_on_also` attribute. If the `clocked_on_also` attribute is not specified, the `required_condition_also` attribute is checked at the falling edge of the clock signal specified by the `clocked_on` attribute. This condition is checked when the slave latch is in the hold state.

### hold_state and hold_state_also Attributes

The `hold_state` and `hold_state_also` attributes specify the values for the Boolean expressions of the `clocked_on` and `clocked_on_also` attributes during the retention mode. The valid values are L (low), H (high), or N (no change).

If the data is restored to both the master and slave latches, the value of the `hold_state` attribute is N. If the data is restored to the slave latch, the value of the `hold_state` attribute is L.
preset_condition and clear_condition Groups

The preset_condition and clear_condition groups contain attributes that specify the conditions, respectively, for the present and clear signals in the normal mode of the retention cell.

The asynchronous control signals, preset and clear, have higher priority over the clock signal. However, these signals do not control the balloon latch. Therefore, if the preset or clear signals are asserted during the restore event, these signals need to be active for a time longer than the restore event so that the master-slave latch content is successfully overwritten, as shown in Figure 9-25. Therefore, the preset and clear signals must be checked at the trailing edge.

Figure 9-25  Preset and Clear Signal Overlaps With Restore

input Attribute

The input attribute specifies how the preset or clear control signal is asserted. The Boolean expression of the input attribute must be identical to one specified in the input attribute of the ff group.

required_condition Attribute

The required_condition attribute specifies the input condition during the active edge of the preset or clear signal. The required_condition attribute is checked at the trailing edge of the preset or clear signal. When the input condition is not met, the cell is in an illegal state.

reference_pin_names Variable

The reference_pin_names variable specifies the input nodes that are used for internal reference within the ff, latch, ff_bank, or latch_bank groups. If you do not specify the reference_pin_names variable, the node names in the ff, latch, ff_bank, or latch_bank groups are considered to be the actual pin or bus names of the cell.
variable1 and variable2 Variables

The variable1 and variable2 variables define the output nodes for internal reference. The values of the variable1 and variable2 variables in the ff, latch, ff_bank, or latch_bank groups must be unique for a cell.

bits Variable

The bits variable defines the width of the ff_bank and latch_bank component.

Pin-Level Attributes

This section describes the pin-level attributes for retention cell modeling.

retention_pin Complex Attribute

The retention_pin attribute identifies the retention pins of a retention cell. In the normal mode, the retention pins are disabled.

The retention_pin attribute has the following arguments:

• Pin class
  The values are
  • restore
    Restores the state of the cell.
  • save
    Saves the state of the cell.
  • save_restore
    Saves and restores the state of the cell. When a single pin in a retention cell performs both the save and restore operations, specify the retention_pin attribute with the save_restore value. The retention pin is in save mode when it saves the data. The retention pin is in restore mode when it restores the data.

function Attribute

The function attribute maps an output, inout, or internal pin to a corresponding internal node or a value of the variable1 or variable2 variable in a ff, latch, ff_bank, or latch_bank group. The function attribute also accepts a Boolean equation containing the variable1 or variable2 variable and other input, inout, or internal pins. Define the function attribute in a pin or bus group.
reference_input Attribute

The `reference_input` attribute specifies the input pins that map to the reference pin names of the corresponding `ff`, `latch`, `ff_bank`, or `latch_bank` group. For each inout, output, or internal pin, the `variable1` or `variable2` value specified in the `function` statement determines the corresponding `ff`, `latch`, `ff_bank`, or `latch_bank` group. You can define the `reference_input` attribute in a `pin` or `bus` group.

save_action and restore_action Attributes

The `save_action` and `restore_action` attributes specify where the save and restore events occur with respect to the save and restore control signals, respectively. Valid values are `L` (low), `H` (high), `R` (rise), and `F` (fall). The `L` or `H` values indicate that the data is actually stored in the balloon latch at the trailing edge of the save signal. The `R` or `F` values indicate that this edge of the restore signal specifies when the data stored in the balloon latch is available at the output of the master-slave latch.

restore_edge_type Attribute

The `restore_edge_type` attribute specifies the type of the edge of the restore signal where the output of the master-slave latch is restored. The `restore_edge_type` attribute supports the following edge types: `edge_trigger`, `leading`, and `trailing`. The default edge type is `leading`. This is because the other control signals, such as clock, preset, and clear are of `leading` edge type, that is, they make the data available at the output of the master-slave latch when the latch is transparent.

The `edge_trigger` type indicates that the output of the master-slave latch is restored at the edge of the restore signal. The master-slave latch resumes normal operation immediately thereafter.

The `leading` edge type indicates that the output of the master-slave latch is restored at the leading edge of the restore signal. The master-slave latch resumes normal operation after the trailing edge of the restore signal.

The `trailing` edge type indicates that the output of the master-slave latch is restored at the trailing edge of the restore signal. The master-slave latch resumes normal operation after the trailing edge of the restore signal.

Figure 9-26 shows the valid data windows when the `restore_edge_type` attribute is set to the `leading` and `trailing` edge types.
The `save_condition` and `restore_condition` attributes specify the input conditions during the save and restore events respectively. These conditions are respectively checked at the values of the `save_action` and `restore_action` attributes. If any one of the conditions are not met, the cell is considered to be in an illegal state.

Retention Cell Model Examples

Figure 9-27 shows a retention cell structure that is defined using the `ff` or `latch` group. The cell has a balloon latch structure to store the data when the main flip-flop is shut down. The data is transferred to the output of the retention cell at the rising edge of the clock signal, CK. The SAVE and RESTORE pins save and restore the data.
Figure 9-27  Retention Cell Model Schematic With Balloon Latch

Example 9-18  Retention Cell With Balloon Latch

```liberty
library (BALLOON_RET_FLOP) {
  delay_model : table_lookup;
  input_threshold_pct_rise : 50 ;
  input_threshold_pct_fall : 50 ;
  output_threshold_pct_rise : 50 ;
  output_threshold_pct_fall : 50 ;
  slew_lower_threshold_pct_fall : 30.0 ;
  slew_lower_threshold_pct_rise : 30.0 ;
  slew_upper_threshold_pct_fall : 70.0 ;
  slew_upper_threshold_pct_rise : 70.0 ;
  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1uA";
  pulling_resistance_unit : "1kohm";
  capacitive_load_unit (0.1,ff);

  voltage_map (VDD, 1.0);
  voltage_map (VDDB, 0.9);
  voltage_map (VSS, 0.0);
}
```
nom_process : 1.0;
nom_temperature : 25.0;
nom_voltage : 1.1;

operating_conditions(typ) {
  process : 1.0;
  temperature : 25;
  voltage : 1.1;
  tree_type : "balanced_tree";
}
default_operating_conditions : typ;
wire_load("05*05") {
  resistance : 1.0;
  capacitance : 25;
  area : 1.1;
  slope : "balanced_tree";
  fanout_length(1,0.39);
}
cell (balloon_ret_cell) {
  retention_cell : retdiff;
  area : 1.0;
  pg_pin(VDD) {
    voltage_name : VDD;
    pg_type : primary_power;
  }
  pg_pin(VSS) {
    voltage_name : VSS;
    pg_type : primary_ground;
  }
  pg_pin(VDDB) {
    voltage_name : VDDB;
    pg_type : backup_power;
  }

  ff ( Q1, QN1 ) {
    clocked_on : " CK ";
    next_state : " D ";
    clear : " RESTORE * !Q2 ";
    preset : " RESTORE * Q2 ";
    clear_preset_var1 : "L";
    clear_preset_var2 : "H";
    power_down_function : "!VDD+VSS";
    /* Latch 1 is powered by primary power supply */
  }
  latch("Q2", "QN2") {
    enable : " SAVE ";
    data_in : "Q";
    power_down_function : "!VDDB+VSS";
    /* Latch 2 is powered by backup power supply */
  }
clock_condition() {
  clocked_on : "CK";
  required_condition : "RESTORE";
  hold_state : L;
}

pin(RESTORE) {
  direction : input;
  capacitance : 0.1;
  related_power_pin : VDDB;
  related_ground_pin : VSS;
  retention_pin(restore, "0");
  restore_action : "H";
  restore_condition : "!CK";
  restore_edge_type : "leading";
}

pin(SAVE) {
  direction : input;
  capacitance : 0.1;
  related_power_pin : VDDB;
  related_ground_pin : VSS;
  retention_pin(save, "0");
  save_action : "H";
  save_condition : "!CK";
}

pin(D) {
  direction : input;
  capacitance : 0.1;
  related_power_pin : VDD;
  related_ground_pin : VSS;
}

pin(CK) {
  direction : input;
  clock : true;
  capacitance : 0.1;
  related_power_pin : VDD;
  related_ground_pin : VSS;
}

pin(Q) {
  direction : output;
  function : "Q1";
  related_power_pin : VDD;
  related_ground_pin : VSS;
  timing() {
    related_pin : "CK";
    timing_type : rising_edge;
    cell_rise(scalar) { values ( "0.1");}
    rise_transition(scalar) { values ( "0.1");}
    cell_fall(scalar) { values ( "0.1");}
    fall_transition(scalar) { values ( "0.1");}
  }
}
retention_condition() {
    power_down_function : "!VDD+VSS";
    required_condition: "!SAVE";
}

/*
pin(q1) {
    direction : internal;
    function : "Q2";
}
*/

Figure 9-28 shows a schematic of a basic retention cell. The state of the cell is stored inside the slave latch that is powered by the backup power VDDB. Figure 9-29 shows the valid values of the input control signals when the cell is in retention mode.

In the figure, and in Example 9-19, the reference cells are defined by using the latch group syntax. The connectivity information for the input pins is specified in the reference_input attribute that is mapped to the reference pin names of the corresponding latch group.

Figure 9-28  Simple Retention Cell Model Schematic
Figure 9-29  Valid RET and CK Signal Values in the Retention Mode

Note:
To retain the stored data, the retention signal, RET, must be low during the retention mode. This condition is specified by the `required_condition` attribute of the `retention_condition` group.

Example 9-19  Retention Cell Model Example Using Multiple Latch Group

```plaintext
library (RET_FLOP) {
  delay_model : table_lookup;

  input_threshold_pct_rise : 50 ;
  input_threshold_pct_fall : 50 ;
  output_threshold_pct_rise : 50 ;
  output_threshold_pct_fall : 50 ;
  slew_lower_threshold_pct_fall : 30.0 ;
  slew_lower_threshold_pct_rise : 30.0 ;
  slew_upper_threshold_pct_fall : 70.0 ;
  slew_upper_threshold_pct_rise : 70.0 ;

  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1uA";
  pulling_resistance_unit : "1kohm";
}
capacitive_load_unit (0.1, ff);

voltage_map (VDD, 1.0);
voltage_map (VDDB, 0.9);
voltage_map (VSS, 0.0);

nom_process : 1.0;
nom_temperature : 25.0;
nom_voltage : 1.1;

operating_conditions(xyz) {
    process : 1.0;
    temperature : 25;
    voltage : 1.1;
    tree_type : "balanced_tree";
}
default_operating_conditions : xyz;
/* Other library level attributes and groups */

cell (retention_flip_flop) {
    retention_cell : retdiff;
    area : 1.0;

    pg_pin (VDD) {
        voltage_name : VDD;
        pg_type : primary_power;
    }
    pg_pin (VSS) {
        voltage_name : VSS;
        pg_type : primary_ground;
    }
    pg_pin (VDDB) {
        voltage_name : VDDB;
        pg_type : backup_power;
    }
    latch (Q1, QN1) {
        enable : "(RET * CK)";
        data_in : "D";
        power_down_function : "!VDD+VSS";
    } /* Latch1 is powered by primary power supply */
    latch("Q2", "QN2") {
        enable : "RET * CK";
        data_in : "Q1";
        power_down_function : "!VDDB+VSS";
    } /* Latch2 is powered by backup power supply */
    clock_condition() {
        clocked_on : "CK";
        required_condition : "RET";
        hold_state : "L";
    }
Figure 9-30 and Example 9-20 show a retention cell structure that is defined using the latch_bank group that has multibit parallel inputs and output buses in the datapath and the clock path.
Figure 9-30  Multibit (2-bit) Retention Cell Model Schematic

Example 9-20  Retention Cell Model Example Using Multiple latch_bank Groups

```liberty
library (RET_FLOP_BANK) {

    input_threshold_pct_rise : 50 ;
    input_threshold_pct_fall : 50 ;
    output_threshold_pct_rise : 50 ;
    output_threshold_pct_fall : 50 ;
    slew_lower_threshold_pct_fall : 30.0 ;
    slew_lower_threshold_pct_rise : 30.0 ;
    slew_upper_threshold_pct_fall : 70.0 ;
    slew_upper_threshold_pct_rise : 70.0 ;

    time_unit : "1ns";
    voltage_unit : "1V";
    current_unit : "1uA";
    pulling_resistance_unit : "1kohm";
    capacitive_load_unit (0.1,ff);

    voltage_map (VDD, 1.0);
    voltage_map (VDDB, 0.9);
    voltage_map (VSS, 0.0);

    nom_process : 1.0;
    nom_temperature : 25.0;
    nom_voltage : 1.1;

    operating_conditions(xyz) {
        process : 1.0 ;
        temperature : 25 ;
        voltage : 1.1 ;
        tree_type : "balanced_tree" ;
    }
    default_operating_conditions : xyz;
}
```
type (bus2) {
    base_type : array;
    data_type : bit;
    bit_width : 2;
    bit_from : 0;
    bit_to : 1;
    downto : false;
}

cell (retention_flip_bank) {
    retention_cell : retdiff_bank;
    area : 1.0;

    pg_pin(VDD) {
        voltage_name : VDD;
        pg_type : primary_power;
    }

    pg_pin(VSS) {
        voltage_name : VSS;
        pg_type : primary_ground;
    }

    pg_pin(VDDB) {
        voltage_name : VDDB;
        pg_type : backup_power;
    }

    latch_bank ("Q1", "QN1", 2) {
        enable : "((RET * CK))'";
        data_in : "SE*D + SE*SI";
        power_down_function : "!VDD+VSS";
    }

    latch_bank ("Q2", "QN2", 2) {
        enable : " RET * CK ";
        data_in : " q1 ";
        power_down_function : "!VDDB+VSS";
    }

    clock_condition() {
        clocked_on : " CK ";
        required_condition : " RET ";
        hold-state : " L ";
    }

    bus(D) {
        bus_type : bus2;
        direction : input;
        capacitance : 0.1;
        related_power_pin : VDD;
        related_ground_pin : VSS;
    }

    bus(q1) {
        bus_type : bus2;
    }
direction : internal;
function : "Q1";
related_power_pin : VDD;
related_ground_pin : VSS;
}
bus(Q) {
  bus_type : bus2;
direction : output;
function : "Q2";
related_power_pin : VDD;
related_ground_pin : VSS;
}
pin(RET) {
  direction : input;
capacitance : 0.1;
related_power_pin : VDDB;
related_ground_pin : VSS;
retention_pin("restore", 1);
save_action : "L";
restore_action : "H";
save_condition : "!CK";
restore_condition : "!CK";
restore_edge_type : "leading";
}

pin(CK) {
  direction : input;
clock : true;
capacitance : 0.1;
related_power_pin : VDD;
related_ground_pin : VSS;
}
retention_condition() {
power_down_function : "!VDD+VSS";
required_condition : "!RET";
}
bus(SI) {
  bus_type : bus2;
direction : input;
clock : true;
capacitance : 0.1;
related_power_pin : VDD;
related_ground_pin : VSS;
}
bus(SE) {
  bus_type : bus2;
direction : input;
clock : true;
capacitance : 0.1;
related_power_pin : VDD;
related_ground_pin : VSS;
}
cell_leakage_power : 1.000000;
Figure 9-31 shows a retention cell structure with two ff groups. The cell has a balloon latch or flip-flop to store the data when the main flip-flop is shut down. The data is transferred to the output of the main flip-flop, Flop1, on the rising edge of the clock signal, CK, when the asynchronous preset, SN, and clear, RN, are inactive. In retention mode, the retention pin, RET, saves the data into the balloon flip-flop and restores the data from the balloon flip-flop to the output pin of the retention cell. At the rising edge of the RET signal, the data is saved inside the balloon flip-flop, Flop2, and Flop1 is shut down. When the power to Flop1 is brought up and the retention pin, RET, becomes inactive, the data from Flop2 is restored to Flop1 at the falling edge of the RET signal.

Figure 9-31  Edge-Triggered Retention Cell Model Schematic

Example 9-21  Retention Cell With Edge-Triggered Balloon Logic

```liberty
library(edge_triggered_retention) {
  delay_model : table_lookup;
  time_unit : "1ns";
  voltage_unit : "1V";
}```
current_unit : "1uA";
capacitive_load_unit (0.1, ff);
default_fanout_load : 1.0;
default_inout_pin_cap : 1.0;
default_input_pin_cap : 1.0;
default_output_pin_cap : 1.0;
input_threshold_pct_rise : 50;
input_threshold_pct_fall : 50;
output_threshold_pct_rise : 50;
output_threshold_pct_fall : 50;
slew_lower_threshold_pct_fall : 30.0;
slew_lower_threshold_pct_rise : 30.0;
slew_upper_threshold_pct_fall : 70.0;
slew_upper_threshold_pct_rise : 70.0;
voltage_map (VDD, 1.0);
voltage_map (VDDB, 1.0);
voltage_map (VSS, 0.0);
nom_process : 1.0;
nom_temperature : 25.0;
nom_voltage : 1.1;
operating_conditions (xyz) {
  process : 1.0;
  temperature : 25;
  voltage : 1.1;
  tree_type : "balanced_tree";
}
default_operating_conditions : xyz;
cell (edge_trigger) {
  retention_cell : "edge_trigger";
  ... /* Other cell-level attributes and groups */
  pg_pin (VDD) {
    voltage_name : "VDD";
    pg_type : "primary_power";
  }
  pg_pin (VDDB) {
    voltage_name : "VDDB";
    pg_type : "backup_power";
  }
  pg_pin (VSS) {
    voltage_name : "VSS";
    pg_type : "primary_ground";
  }
  ff ("IQ1", "IQN1") {
    next_state : "D";
    clocked_on : "CK";
    clear : "RN + (RET * q1)";
    preset : "SN + (RET * q1)";
    clear_preset_var1 : L;
    clear_preset_var1 : H;
    power_down_function : "!VDD + VSS"; /* Flip-Flop "Flopl" is powered by Primary power supply */
  }
}
ff ("IQ2", "IQN2") {
    next_state : "Q";
    clocked_on : "RET";
    power_down_function : "!VDDS + VSS"; /* Flip-Flop “Flop2” is powered by Primary power supply */
}

clock_condition() {
    clocked_on : "CK"; /* clock must be Low to go into retention mode */
    hold_state : "N"; /* when clock switches (either direction), RET must be High */
    condition : "RET";
}
clear_condition() {
    input : "!RN"; /* When clear de-asserts, RET must be high to allow Low value to be transferred to Flop1 */
    required_condition : "RET";
}
preset_condition() {
    input : "!SN"; /* When clear de-asserts, RET must be high to allow High value to be transferred to Flop1 */
    required_condition : "RET";
}
retention_condition() {
    power_down_function : "!VDD+VSS";
    required_condition: "!RET";
}

pin (q1) {
    direction : "internal";
    function : "IQ2";
    related_power_pin : "VDD";
    related_ground_pin : "VSS";
    ... /* Other pin-level attributes and groups */
}

pin (CK) {
    direction : "input";
    clock : true;
    capacitance : 1.0;
    related_power_pin : "VDD";
    related_ground_pin : "VSS";
    ... /* Other pin-level attributes and groups */
}

pin (RET) {
    related_power_pin : "VDDB";
    related_ground_pin : "VSS";
    capacitance : 1.0;
    direction : "input";
    retention_pin("save_restore",0);
    save_action : "R";
    restore_action : "R";
    save_condition : "!CK";
    restore_condition : "!CK";
Figure 9-32 and Example 9-22 show a scan retention cell structure that is defined using the `latch` group. The cell is the scan version of the retention cell modeled in Example 9-19. Similar to Example 9-19, this retention cell structure is also a store in-place retention cell.
**Example 9-22 MUX-Scan Retention Cell Model**

```
library (Retention_cell_Example) {
  delay_model : table_lookup;
  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1uA";
  capacitive_load_unit (0.1,ff);
  default_fanout_load : 1.0;
  default_inout_pin_cap : 1.0;
  default_input_pin_cap : 1.0;
  default_output_pin_cap : 1.0;
  input_threshold_pct_rise : 50 ;
  input_threshold_pct_fall : 50 ;
  output_threshold_pct_rise : 50 ;
  output_threshold_pct_fall : 50 ;
  slew_lower_threshold_pct_fall : 30.0 ;
  slew_lower_threshold_pct_rise : 30.0 ;
  slew_upper_threshold_pct_fall : 70.0 ;
  slew_upper_threshold_pct_rise : 70.0 ;
  voltage_map (VDD, 1.0);
  voltage_map (VDDB, 0.9);
  voltage_map (VSS, 0.0);
  nom_process : 1.0;
  nom_temperature : 25.0;
  nom_voltage : 1.1;

  operating_conditions(xyz) {
    process : 1.0 ;
    temperature : 25 ;
    voltage : 1.1 ;
  }
```

tree_type : "balanced_tree";
}
default_operating_conditions : xyz;
... /* Other library-level attributes */
cell (scan_retention_cell) {
  retention_cell : my_scan_ret_cell;
  ... /* Other cell-level attributes and groups */
  pg_pin(VDD) {
    voltage_name : VDD;
    pg_type : primary_power;
  }
  pg_pin(VSS) {
    voltage_name : VSS;
    pg_type : primary_ground;
  }
  pg_pin(VDDB) {
    voltage_name : VDDB;
    pg_type : backup_power;
  }
  pin(RET) {
    direction : input;
    capacitance : 0.1;
    related_power_pin : VDDB;
    related_ground_pin : VSS;
    retention_pin(save_restore, "1");
    ... /* Other pin-level attributes and groups */
  }
  pin(D) {
    direction : input;
    capacitance : 0.1;
    related_power_pin : VDD;
    related_ground_pin : VSS;
    ... /* Other pin-level attributes and groups */
  }
  pin(CK) {
    direction : input;
    clock : true;
    capacitance : 0.1;
    related_power_pin : VDD;
    related_ground_pin : VSS;
    ... /* Other pin-level attributes and groups */
  }
  pin(Q) {
    direction : output;
    function : "Q2";
    related_power_pin : VDD;
    related_ground_pin : VSS;
    reference_input : "RET CK q1";
    ... /* Other pin-level attributes and groups */
  }
  pin(q1) {
    direction : internal;
function : "Q1";
related_power_pin : VDD;
related_ground_pin : VSS;
reference_input : "RET CK SE D SI"
... /* Other pin-level attributes and groups */
}
retention_condition() {
power_down_function : "!VDD+VSS";
required_condition: "!RET";
}
latch ("p1 p2,p3,p4,p5", "Q1", "QN1") {
enable : " p1' + p2' ";
data_in : " p3'*p4 + p3*p5 ";
power_down_function : "!VDD+VSS"; /* Latch 1 is powered by Primary
power supply */
}
latch ("p1 p2 p3", "Q2", "QN2") {
enable : " p1 * p2 ";
data_in : " p3 ";
power_down_function : "!VDDB+VSS"; /* Latch 2 is powered by Backup
power supply */
}
test_cell() {
  pin(SI) {
    direction : input;
    signal_type : "test_scan_in";
  }
  pin(RET) {
    direction : input;
  }
  pin(D) {
    direction : input;
  }
  pin(SE) {
    direction : input;
    signal_type : "test_scan_enable";
  }
  pin(CK) {
    direction : input;
  }
latch ("Q1", "QN1") {
enable : " RET' + CK' ";
data_in : " D ";
}
latch ("Q2", "QN2") {
enable : " RET * CK ";
data_in : " q1 ";
}
pin (q1) {
  direction : internal;
  function : "Q1";
Always-On Cell Modeling

In complex low-power designs, some signals need to be routed through blocks that have been shut down. As a result, a variety of cell categories require “always-on” signal pins. Always-on cells remain powered on by a backup power supply in the region where they are placed even when the main power supply is switched off. The cells also have a secondary backup power pin that supplies the current that is necessary when the main supply is not available.

For tools to recognize always-on cells in the reference library and use them during special always-on synthesis, library models need an attribute that can identify them. The `always_on` attribute identifies always-on cells and pins.

Always-On Cell Syntax

```plaintext
library (library_name) {
    ...
    cell (cell_name) {
        always_on : true;
        ...
    }
    pin (pin_name) {
        always_on : true;
        ...
    }
    ...
}
```

always_on Simple Attribute

The `always_on` simple attribute models always-on cells and signal pins. During compilation, the `always_on` attribute is automatically added to buffer or inverter cells that have input and output pins that are linked to backup power or backup ground PG pins. The `always_on` attribute is supported at the cell level and at the pin level.
Note:
Some macro cell input pins require that you specify the `always_on` attribute for always-on pins.

---

**Always-On Simple Buffer Example**

Figure 9-33 and the example that follows it show a simple always-on cell buffer. In the figure, the A signal pin and the Y signal pin are linked to the VDDBAK and VSS power and ground pin pair.

![Simple Always-On Cell Buffer](image)

### Figure 9-33 Simple Always-On Cell Buffer

```liberty
library (my_library) {
...
    voltage_map (VDD, 1.0);
    voltage_map (VDDBAK, 1.0);
    voltage_map (VSS, 0.0);
...

cell(buffer_type_AO) {  
    always_on : true;
    /* The always-on attribute is not required at the cell level if the cell is an always-on cell*/
    /* Other cell level information */
    pg_pin(VDD) {  
        voltage_name : VDD;
        pg_type : primary_power;
    }

    pg_pin(VDDBAK) {  
        voltage_name : VDDBAK;
        pg_type : backup_power;
    }

    pg_pin(VSS) {  
        voltage_name : VSS;
        pg_type : primary_ground;
    }
}
```
Macro Cell Modeling

Macro cells do not contain internal logic information. At the cell-level, macro cells are modeled using the `is_macro_cell` attribute. To model macro cells for power management, use the power management pin-level attributes described in the following sections.

Macro Cell Isolation Modeling

To indicate that a macro cell is internally isolated and does not require external isolation, set the `is_isolated` attribute. Figure 9-34 shows a macro cell with the `is_isolated` attribute. The macro cell is connected to a power-switching circuit. The macro cell pin, In1 is internally isolated, and In2 is connected to an external isolation cell. When the `is_isolated` attribute is set on the pin, In1, the pin is considered to be internally isolated.
Example 9-23 shows the modeling syntax of an internally isolated macro cell.

**Example 9-23 Macro Cell Isolation Modeling Syntax**

```plaintext
cell (cell_name) {
    ...
    is_macro_cell : true;
    pin (pin_name) {
        direction : input | inout | output;
        is_isolated : true | false;
        isolation_enable_condition : Boolean_expression;
        ...
    }
    bus (bus_name) {
        direction : input | inout | output;
        is_isolated : true | false;
        isolation_enable_condition : Boolean_expression;
        ...
    }
    bundle (bundle_name) {
        direction : input | inout | output;
        is_isolated : true | false;
        isolation_enable_condition : Boolean_expression;
        ...
    }
    ...
}
```
**Example 9-24** shows a typical model of an internally isolated macro cell.

**Example 9-24  Modeling an Internally Isolated Macro Cell by Using the is_isolated and isolation_enable_condition Attributes**

library (internal_isolated_pin_example) {
  ...
  type ( bus_type_name ) {
    base_type : array;
    data_type : bit
    bit_width : 2;
    bit_from : 1;
    bit_to : 0;
  }
  cell ( macro ) {
    is_macro_cell : true;
    pg_pin(GND) {
      voltage_name : VSS;
      pg_type : primary_ground;
    }
    pg_pin(VDDI) {
      voltage_name : VDDH;
      pg_type : primary_power;
    }
    ......
    pin (en) {
      direction : input;
      ...
    }
    pin (in_bus) {
      bus_type : bus_type_name;
      direction : input;
      ...
    }
    pin (sp) {
      direction : input|inout;
      is_isolated : true;
      isolation_enable_condition : "en’";
      related_power_pin : VDDI;
      related_ground_pin : GND;
      ......
    }
    pin (out) {
      direction : output;
      is_isolated : true;
      isolation_enable_condition : "en’";
      related_power_pin : VDDI;
      related_ground_pin : GND;
      ......
    }
  }
  bus (bus_a) {
    bus_type : bus_type_name;
    is_isolated : true;
  }
  ...
}
isolation_enable_condition : "en’ * in_bus * in_bundle";
related_power_pin : VDDI;
related_ground_pin : GND;
...}
...}
}

## Pin-Level Attributes

This section describes the pin-level attributes of isolated macro cells.

### is_isolated Attribute

The `is_isolated` attribute indicates that a pin, bus, or bundle of a macro cell is internally isolated and does not require the insertion of an external isolation cell. The default is `false`.

**Note:**

- The `is_isolated` attribute also supports the internal isolation of pad-cell pins.

### isolation_enable_condition Attribute

The `isolation_enable_condition` attribute specifies the condition of isolation for internally isolated pins, buses, or bundles of a macro cell. When this attribute is defined in a `pin` group, the corresponding Boolean expression can include only input and inout pins. Do not include the output pins of an internally isolated macro cell in the Boolean expression.

When the `isolation_enable_condition` attribute is defined in a `bus` or `bundle` group, the corresponding Boolean expression can include pins, and buses and bundles of the same bit-width. For example, when the Boolean expression includes a bus and a bundle, both of them must have the same bit-width.

All the pins, buses, and bundles specified in the Boolean expression of the `isolation_enable_condition` attribute must have the `always_on` attribute.

**Note:**

- The `isolation_enable_condition` attribute also supports the internal isolation of pad-cell pins.

---

### Modeling Macro Cells With Internal PG Pins

This capability is useful for modeling macro cells containing internal voltage converters that supply power to the internal subcells.

**Figure 9-35** shows the schematic of a macro cell with internal voltage converters, VC1 and VC2. VC1 and VC2 convert the external supply voltages, VDD and VCC, into internal supply...
voltages, IN1 and IN2, respectively. The pins, IN1 and IN2, are internal PG pins of the macro cell and are used to power the logic connected to the input ports, A1 and A2.

Example 9-25 shows a typical model of the macro cell. The pg_type attribute is set to internal_power, the direction attribute is set to internal, and the pg_function attribute is set to VDD and VCC, on the PG pins, IN1 and IN2, respectively.

Example 9-25 Macro Cell Model With Internal PG Pins

```plaintext
voltage_map (VDD, 1.4);
voltage_map (VCC, 2.8);
voltage_map (VDD, 1.4);
voltage_map (VCC, 2.8);
voltage_map (VSS, 0.0);
voltage_map (IN1, 0.7);
voltage_map (IN2, 6.5);
cell(new_macro_cell) {
  ...
```
pg_pin(VDD) {
  voltage_name : VDD;
  pg_type : primary_power;
}
pg_pin(VCC) {
  voltage_name : VCC;
  pg_type : primary_power;
}
pg_pin(VSS) {
  voltage_name : VSS;
  pg_type : primary_ground;
}
pg_pin(IN1) {
  voltage_name : IN1;
  pg_type : internal_power;
  direction : internal;
  pg_function : VDD;
}
pg_pin(IN2) {
  voltage_name : IN2;
  pg_type : internal_power;
  direction : internal;
  pg_function : VCC;
}
...
pin(CK) {
  related_power_pin : VDD;
  related_ground_pin : VSS;
  direction : input;
  ...
}
pin(A1) {
  related_power_pin : IN1;
  related_ground_pin : VSS;
  direction : input;
  ...
}
pin(A2) {
  related_power_pin : IN2;
  related_ground_pin : VSS;
  direction : input;
}
...

Note:
You do not need to specify the switch_cell_type and switch_function attributes for this macro cell because it does not include any built-in switches.
Macro Cell With Fine-Grained Internal Power Switches

Figure 9-36 and the example that follows it show a macro cell with fine-grained internal power switches. In the figure, the CTRL signal pin is linked to the PVDD and PVSS power and ground pin pair, and the MA signal pin and the A signal pin are linked to the PVVDD and PVSS power and ground pin pair.

Figure 9-36  Macro Cell With Fine-Grained Power Switch Schematics

library (macro_switch) {
  ...
  Voltage_map (PVDD, 1.0);
  Voltage_map (PVVDD, 1.0);
  Voltage_map (PVSS, 0.0);

  operating_conditions(XYZ) {
    process : 1.0;
    voltage : 1.0;
    temperature : 25.0;
  }
  default_operating_conditions : XYZ;

  cell(MACRO) {
    is_macro_cell : true;
    switch_cell_type : fine_grain;
    ...

    pg_pin(PVDD) {
      voltage_name : PVDD;
      pg_type : primary_power;
      direction : input;
    }
    pg_pin(PVSS) {

Macro Cell With an Always-On Pin Example

Figure 9-37 and the example that follows it show a macro cell with one always-on pin. In the figure, the A signal pin and Y1 signal pin are linked to the VDDBAK and VSS power and ground pin pair. The B, C, and Y2 signal pins are linked to the VDD and VSS power and ground pin pair.
Figure 9-37  Macro Cell With an Always-On Pin

```liberty
library (my_library) {
    ...
    voltage_map (VDD, 2.0) ;
    voltage_map (VSS, 0.1) ;
    voltage_map (VDDBAK, 1.0) ;
    ...

cell(Macro_cell_with_AO_pins) {
    /* other cell level information */

    pg_pin(VDD) {
        voltage_name : VDD;
        pg_type : primary_power;
    }

    pg_pin(VSS) {
        voltage_name : VSS;
        pg_type : primary_ground;
    }

    pg_pin(VDDBAK) {
        voltage_name : VDDBAK;
        pg_type : backup_power;
    }
    ...

    pin (A) {
        always_on : true;
        related_power_pin : VDDBAK;
        related_ground_pin : VSS;
        /* Other pin level information */
    }

    pin (B C) {
        related_power_pin : VDD;
        related_ground_pin : VSS;
        /* Other pin level information */
    }
}```
Modeling Antenna Diodes

Modeling antenna diodes includes modeling antenna-diode cells and cells with built-in antenna-diode ports.

Antenna-Diode Cell Modeling

An antenna-diode cell has only one input to a diode that discharges electrical charges. The cell is typically inserted at the boundary between two power domains and can be placed in any one of the power domains. Figure 9-38 shows the three types of antenna-diode cells.
In multivoltage designs, the power type antenna-diode cell is connected to VDD of a power domain where VSS is shut down. The ground type antenna-diode cell is connected to VSS of a power domain where VDD is shut down. The power-and-ground type antenna-diode cell is connected to both VDD and VSS. To eliminate leakage paths that can result in chip failure, the correct type of antenna-diode cell must be inserted.

The modeling syntax of the antenna-diode cell is as follows:

```liberty
cell (cell name) {
  antenna_diode_type : power | ground | power_and_ground;
  pin (pin name) {
    antenna_diode_related_power_pins : power pin name;
    antenna_diode_related_ground_pins : ground pin name;
    ...
  }
  ...
}
```

In a library, a cell that has a single pin with the `direction` attribute set to `input` or `inout` is considered to be an antenna-diode cell.

**Cell-Level Attribute**

**antenna_diode_type Attribute**

The `antenna_diode_type` attribute specifies the type of antenna-diode cell. Valid values are `power`, `ground`, and `power_and_ground`. 
Pin-Level Attributes

antenna_diode_related_power_pins Attribute
The *antenna_diode_related_power_pins* attribute specifies the related power pin of the antenna-diode cell. Apply the *antenna_diode_related_power_pins* attribute to the input pin of the cell.

antenna_diode_related_ground_pins Attribute
The *antenna_diode_related_ground_pins* attribute specifies the related ground pin of the antenna-diode cell. Apply the *antenna_diode_related_ground_pins* attribute to the input pin of the cell.

Antenna-Diode Cell Modeling Example

*Example 9-26* shows a typical model of the antenna-diode cell.

*Example 9-26  Antenna-Diode Cell Model*

```liberty
library (antenna_library)
...
cell (power_diode_cell) {
    antenna_diode_type : "power";
    pg_pin (VDD) {
        voltage_name : "VDD";
        pg_type : "primary_power";
    }
    pin (INP) {
        antenna_diode_related_power_pins : "VDD";
        direction : "input";
    }
}
cell (ground_diode_cell) {
    antenna_diode_type : "ground";
    pg_pin (VSS) {
        voltage_name : "VSS";
        pg_type : "primary_ground";
    }
    pin (INP) {
        antenna_diode_related_ground_pins : "VSS";
        direction : "input";
    }
}
cell (pg_diode_cell) {
    antenna_diode_type : "power_and_ground";
    pg_pin (VDD) {
        voltage_name : "VDD";
        pg_type : "primary_power";
    }
}```
pg_pin (VSS) {
    voltage_name : "VSS";
    pg_type : "primary_ground";
}

pin (INP) {
    antenna_diode_related_power_pins : "VDD";
    antenna_diode_related_ground_pins : "VSS";
    direction : "input";
}

---

Modeling Cells With Built-In Antenna-Diode Ports

To model a cell with a built-in antenna-diode pin, specify the `antenna_diode_type` attribute on the pin.

The modeling syntax of the cell with a built-in antenna-diode pin is as follows:

```markdown
cell (cell name) {
    ...
    pin (pin name) {
        antenna_diode_type : power | ground | power_and_ground;
        antenna_diode_related_power_pins : "power_pin1 power_pin2";
        antenna_diode_related_ground_pins : "ground_pin1 ground_pin2";
        ...
    }
    ...
    pin (pin name) {
        ...
    }
    ...
}
```

---

Pin-Level Attributes

### antenna_diode_type Attribute

The `antenna_diode_type` attribute specifies the type of antenna-diode pin. Valid values are `power`, `ground`, and `power_and_ground`.

### antenna_diode_related_power_pins Attribute

The `antenna_diode_related_power_pins` attribute specifies the related power pins for the antenna-diode pin. Apply the `antenna_diode_related_power_pins` attribute to the antenna-diode pin.
antenna_diode_related_ground_pins Attribute

The `antenna_diode_related_ground_pins` attribute specifies the related ground pins for the antenna-diode pin. Apply the `antenna_diode_related_ground_pins` attribute to the antenna-diode pin.

Antenna-Diode Pin Modeling Example

Example 9-27 shows a typical model of a cell with a built-in antenna-diode pin.

Example 9-27  A Cell Model With Built-In power_and_ground Antenna-Diode Pin Port

cell (cell_with_internal_diode_port)
  area : "1.0";
  pg_pin (VDD) {
    voltage_name : "VDD";
    pg_type : "primary_power";
  }
  pg_pin (VDD1) {
    voltage_name : "VDD1";
    pg_type : "primary_power";
  }
  pg_pin (VSS) {
    voltage_name : "VSS";
    pg_type : "primary_ground";
  }
  pg_pin (VSS1) {
    voltage_name : "VSS1";
    pg_type : "primary_ground";
  }
  pin (antenna_diode) {
    antenna_diode_type: power_and_ground;
    antenna_diode_related_power_pins : "VDD VDD1";
    antenna_diode_related_power_pins : "VSS VSS1";
    direction : "input";
    capacitance : "1.0";
  }
  pin (INP1) {
    direction : "input";
    capacitance : "1.0";
  }
  pin (INP2) {
    direction : "input";
    capacitance : "1.0";
  }
  pin (OUT1) {
    related_power_pin : "VDD";
    related_ground_pin : "VSS";
    direction : "output";
  }
  pin (OUT1) {
    related_power_pin : "VDD";
  }
related_ground_pin : "VSS";
direction : "output";
}
Composite Current Source Modeling

This chapter provides an overview of composite current source modeling (CCS). It covers the syntax for CCS modeling in the following sections:

- Modeling Cells With Composite Current Source Information
- Representing Composite Current Source Driver Information
- Representing Composite Current Source Receiver Information
- Comparison Between Two-Segment and Multisegment Receiver Models
- Two-Segment Receiver Capacitance Model
- Multisegment Receiver Capacitance Model
- CCS Retain Arc Support
Modeling Cells With Composite Current Source Information

Composite current source (CCS) models include driver and receiver models. A driver model can be used with or without the receiver model.

Nanometer transistor geometries with nonlinear waveforms, nonlinear interconnect delays, and high circuit speeds require accurate delay calculation. CCS models support driver complexity by using a time- and voltage-dependent current source with an infinite drive resistance. The driver model maps the arbitrary transistor behavior for lumped loads to that for an arbitrary detailed parasitic network, which results in high accuracy.

In the receiver model, the input capacitance of a receiver is dynamically adjusted during the voltage transitions.

Representing Composite Current Source Driver Information

Specify the nonlinear delay information based on input slew and output load at the pin level by defining a current lookup table in a timing group.

Composite Current Source Lookup Tables

To define the lookup tables, use the following groups and attributes:

- `output_current_template` group in the `library` group
- `output_current_rise` and `output_current_fall` groups in the `timing` group

Defining the `output_current_template` Group

Use this library-level group to create templates of common information that multiple current vectors can use. A table template specifies the composite current source driver model and the breakpoints for the axis. Specifying `index_1`, `index_2`, and `index_3` values at the library level is optional.

`output_current_template` Syntax

```
library(name_id) { ...
  output_current_template(template_name_id) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: time;
    ...
  }
...```

Template Variables for CCS Driver Models

The table template specifying composite current source driver models can have three variables: variable_1, variable_2, and variable_3. The valid values for variable_1 and variable_2 are input_net_transition and total_output_net_capacitance. The only valid value for variable_3 is time.

output_current_template Example

library (new_lib) {
...
output_current_template (CCT) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: time;
...
}...
}

Defining the Lookup Table Output Current Groups

To specify the output current for the nonlinear table model, use the output_current_rise and output_current_fall groups within the timing group.

Note:
The output_current_rise group can include negative current values and the output_current_fall group can include positive current values.

output_current_rise Syntax

timing() {
    output_current_rise () {
        vector (template_name_id) {
            reference_time : float;
            index_1 (float);
            index_2 (float);
            index_3 ("float,..., float");
            values("float,..., float");
        }
    }
}
vector Group

Define the vector group in the output_current_rise or output_current_fall group. This group stores current information for a particular input slew and output load.

reference_time Simple Attribute

Define the reference_time simple attribute in the vector group. The reference_time attribute represents the time at which the input waveform crosses the rising or falling input delay threshold.

Template Variables for CCS Driver Models

The table template specifying composite current source driver models can have three variables: variable_1, variable_2, and variable_3. The valid values for variable_1 and variable_2 are input_net_transition and total_output_net_capacitance. The only valid value for variable_3 is time.

The index value for input_net_transition or total_output_net_capacitance is a single floating-point number. The index values for time are a list of floating-point numbers.

The values attribute defines a list of floating-point numbers that represent the current values of the driver model.

vector Group Example

```liberty
library (new_lib) {
...
output_current_template (CCT) {
  variable_1: input_net_transition;
  variable_2: total_output_net_capacitance;
  variable_3: time;
}
...
timing() {
  output_current_rise() {
    vector(CCT) {
      reference_time : 0.05;
      index_1(0.1);
      index_2(2.1);
      index_3("1.0, 1.5, 2.0, 2.5, 3.0");
      values("1.1, 1.2, 1.4, 1.3, 1.5");
...}
  }
  output_current_fall() {
    vector(CCT) {
      reference_time : 0.05;
```
Representing Composite Current Source Receiver Information

Switching transistors have nonlinear capacitance. At an input or inout node, the collective capacitance of all the transistors connected to that node is modeled as the receiver capacitance. The Miller effect due to the output switching in the opposite direction adds to the nonlinearity. The capacitances are dependent on the input slew and output load.

CCS receiver models are of two types:

- **Two-segment receiver capacitance model**
  
  The voltage rise or fall at an input or inout node is divided into two segments and the corresponding capacitance values are stored as receiver_capacitance1 and receiver_capacitance2 lookup tables.

- **Multisegment capacitance model**
  
  The voltage rise or fall at an input or inout node is divided into multiple segments and the corresponding capacitance values are stored as lookup tables. The number of segments is represented by the letter N. This model better represents the nonlinearity of the net receiver capacitance.

Comparison Between Two-Segment and Multisegment Receiver Models

You might achieve different results with the two-segment and the multisegment receiver capacitance models.

This is because the number of segments are different. Also, the multisegment model uses the input slew value of the library-level `slew_derate_from_library` attribute, shown with the solid red line in Figure 10-1. This slew is used for the `input_net_transition (index_1)` values of all the N segments, regardless of the local waveform slopes.

The two-segment model uses the local slope of the two segments (s1 and s2 in Figure 10-1), shown with the solid blue lines. So even at N = 2, the multisegment and two-segment model values might not match.
Two-Segment Receiver Capacitance Model

Define the CCS receiver model

- At the pin level using the `receiver_capacitance` group.
  
  Specify the `receiver_capacitance1_rise, receiver_capacitance1_fall, receiver_capacitance2_rise, receiver_capacitance2_fall` groups in the `receiver_capacitance` group.

  The pin-level definition is not a function of the output capacitance and is useful when there are no forward timing arcs.

- At the timing level by specifying the `receiver_capacitance1_rise, receiver_capacitance1_fall, receiver_capacitance2_rise, receiver_capacitance2_fall` groups in the `timing` group.

Syntax

The following is the Liberty syntax of CCS timing receiver models.
Defining the receiver_capacitance Group at the Pin Level

To model the CCS receiver capacitance at the pin level, define the receiver_capacitance group in the pin group. In the receiver_capacitance group, specify the receiver_capacitance1_rise, receiver_capacitance1_fall,
receiver_capacitance2_rise, and receiver_capacitance2_fall groups. Each of these groups specify a lookup table. The lookup table can be one-dimensional, two-dimensional, or scalar.

The index values in the index_1 and index_1 attributes are a list of ascending floating-point numbers. The values attribute defines a list of floating-point numbers that represent the capacitances of the receiver model.

Define the template of the lookup table in the library-level lu_table_template group. To use the template, specify the name of the lu_table_template group as the receiver capacitance group name.

Defining the lu_table_template Group

The lu_table_template group defines the template for the following groups specified in the receiver_capacitance group:

- receiver_capacitance1_rise
- receiver_capacitance1_fall
- receiver_capacitance2_rise
- receiver_capacitance2_fall

The template definition includes the variable_1, variable_2, index_1, and index_2 attributes. The variable_1 and variable_2 attributes specify the index variables of the lookup tables of the receiver capacitance groups. The index_1 and index_2 attributes specify the numerical values of these variables.

The values of the variable_1 and variable_2 attributes are input_net_transition and total_output_net_capacitance.

You can specify the following types of lookup tables in receiver capacitance groups:

- One-dimensional tables with a single index, such as input_net_transition
- Two-dimensional tables with the indexes, input_net_transition and total_output_net_capacitance
- A scalar that has a single variation value irrespective of the input_net_transition and total_output_net_capacitance values

lu_table_template Group Syntax

library(name) {
...
  lu_table_template(template_name) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    index_1 ["float,..., float"];}
Conditional Data Modeling

Use the `mode`, `when`, and `char_when` attributes in the `receiver_capacitance` group for conditional data modeling in CCS receiver models.

`when` Attribute

The `when` attribute specifies the Boolean condition for the input state of the receiver capacitance timing arc.

`char_when` Attribute

The `char_when` attribute specifies the input state at which the receiver capacitance timing arc was characterized.

`mode` Attribute

The complex `mode` attribute specifies the current mode of the cell. When defined in the `receiver_capacitance` group, the `mode` attribute specifies the receiver capacitance for the given mode. If you specify the `mode` attribute, you must define the `mode_definition` group at the cell level.

Examples

Example 10-1 shows a receiver capacitance model with conditional data, that is, the `when` and `mode` attributes.

**Example 10-1  Modeling Receiver Capacitance With when and mode Attributes**

```plaintext
library(new_lib) {
  ...
  output_current_template(CCT) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: time;
    index_1("0.1, 0.2");
    index_2("1, 2");
    index_3("1, 2, 3, 4, 5");
  }
  lu_table_template(LTT1) {
    variable_1: input_net_transition;
    index_1("0.1, 0.2, 0.3, 0.4");
  }
  lu_table_template(LTT2) {
    variable_1: input_net_transition;
  }
  ...
}
```
variable_2: total_output_net_capacitance;
index_1("0.1, 0.2");
index_2("1, 2");
}
...
cell(my_cell) {
...
mode_definition(rw) {
  mode_value(read) {
    when : "I";
    sdf_cond : "I == 1";
  }
  mode_value(write) {
    when : "!I";
    sdf_cond : "I == 0";
  }
}

pin(I) { /* pin-based receiver model defined for pin 'A' */
  direction : input;
  /* receiver capacitance for condition 1 */
  receiver_capacitance() {
    when : "I"; /* or using mode as next commented line */
    /* mode (rw, read); */
    receiver_capacitance1_rise(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance1_fall(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance2_rise(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance2_fall(LTT1) {
      values("1, 2, 3, 4");
    }
  }
  /* receiver capacitance for condition 2 */
  receiver_capacitance() {
    when : "!I"; /* or using mode as next commented line */
    /* mode (rw, write); */
    receiver_capacitance1_rise(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance1_fall(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance2_rise(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance2_fall(LTT1) {
      values("1, 2, 3, 4");
    }
  }
}

pin (ZN) {
  direction : input;
  capacitance : 1.2;
  ...
Example 10-2 shows the use of the char_when attribute in the pin-level receiver capacitance model.

Example 10-2 Receiver Capacitance Model With the char_when Attribute

library (mylib) {
    /* library unit, slew, OC and other library level information */
    /* receiver model template */
    lu_table_template ("template_8x1") {
        variable_1 : "input_net_transition";
        index_1("0, 1, 2, 3, 4, 5, 6, 7");
    }
    ...
    cell (mycell) {
        pin (Y) {
            direction : "output";
            function : "A1 * A2 * A3";
            ...
            /* timing-based ccs receiver model and driver model */
            timing () {
                related_pin : "A1";
                ...
            }
            timing () {
                related_pin : "A2";
                ...
            }
            timing () {
                related_pin : "A3";
                ...
            }
            ...
        } /* end cell */
    } /* end library */
    ...
}

/* pin-based ccs receiver model */
receiver_capacitance () {
    /* default condition */
    /* characterization input-state */
    char_when : "!A2 * !A3";
    receiver_capacitance1_rise ("template_8x1") {
        values("0.11111, 0.22222, 0.33333, 0.44444, 0.55555, \
receiver_capacitance2_rise ("template_8x1") {
  values("0.11111, 0.22222, 0.33333, \$
    0.44444, 0.55555, \$
    0.66666, 0.77777, 0.88888")
}
receiver_capacitancel1_fall ("template_8x1") {
  values("0.11111, 0.22222, 0.33333, \$
    0.44444, 0.55555, \$
    0.66666, 0.77777, 0.88888")
}
receiver_capacitance2_fall ("template_8x1") {
  values("0.11111, 0.22222, 0.33333, \$
    0.44444, 0.55555, \$
    0.66666, 0.77777, 0.88888")
}
receiver_capacitance () {
  when : "!A1 * A2";
  receiver_capacitancel1_rise ("template_8x1") {
    values("0.11111, 0.22222, 0.33333, \$
      0.44444, 0.55555, \$
      0.66666, 0.77777, 0.88888")
  }
  receiver_capacitance2_rise ("template_8x1") {
    values("0.11111, 0.22222, 0.33333, \$
      0.44444, 0.55555, \$
      0.66666, 0.77777, 0.88888")
  }
  receiver_capacitancel1_fall ("template_8x1") {
    values("0.11111, 0.22222, 0.33333, \$
      0.44444, 0.55555, \$
      0.66666, 0.77777, 0.88888")
  }
  receiver_capacitance2_fall ("template_8x1") {
    values("0.11111, 0.22222, 0.33333, \$
      0.44444, 0.55555, \$
      0.66666, 0.77777, 0.88888")
  }
  receiver_capacitance () {
    when : "A1 * !A2";
Defining the Receiver Capacitance Groups at the Timing Level

At the timing level, you do not need to define the receiver_capacitance group. Define the receiver capacitance for the timing arcs by using the receiver_capacitancel1_rise, receiver_capacitancel1_fall, receiver_capacitancel2_rise, and receiver_capacitancel2_fall groups in the timing group.
Conditional Data Modeling

Use the \texttt{mode} and \texttt{when} attributes in timing arcs for conditional timing arcs and constraints.

Defining the \texttt{lu_table_template} Group

Use this library-level group to create templates of common information that multiple lookup tables can use. A table template specifies the table parameters and the breakpoints for each axis. Assign each template a name so that lookup tables can refer to it.

\texttt{lu_table_template} Group

Define your lookup table templates in the \texttt{library} group.

\texttt{lu_table_template} Group Syntax

Define your lookup table templates in the \texttt{library} group.

\begin{verbatim}
library(name_id) {
  ... 
  lu_table_template(template_name_id) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    index_1 ("float,..., float");
    index_2 ("float,..., float");
    ... 
  }
  ...
}
\end{verbatim}

Template Variables for CCS Receiver Models

The table template specifying composite current source receiver models can have only two variables: \texttt{variable_1} and \texttt{variable_2}. The parameters are the input transition time and the total output capacitance of a constrained pin.

The index values in the \texttt{index_1} and \texttt{index_2} attributes are a list of ascending floating-point numbers.

In the timing level, the table template specifying composite current source receiver models can have two variables: \texttt{variable_1} and \texttt{variable_2}. The valid values for either variable are \texttt{input_net_transition} and \texttt{total_output_net_capacitance}.

The index values in the \texttt{index_1} and \texttt{index_2} attributes are a list of ascending floating-point numbers.

The \texttt{values} attribute defines a list of floating-point numbers that represent the capacitance for the receiver model.
**lu_table_template Example**

```plaintext
... lu_table_template(LTT2) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    index_1("0.1, 0.2, 0.4, 0.3");
    index_2("1.0, 2.0");
}
```

**Timing-Level receiver_capacitance Example**

```plaintext
timing() { /* timing arc-based receiver model*/
    ...
    related_pin: "B"
    receiver_capacitance1_rise(LTT2) {
        values("1.1, 4., 2.0, 3.2");
    }
    receiver_capacitance1_fall(LTT2) {
        values("1.0, 3.2, 4.0, 2.1");
    }
    receiver_capacitance2_rise(LTT2) {
        values("1.1, 4., 2.0, 3.2");
    }
    receiver_capacitance2_fall(LTT2) {
        values("1.0, 3.2, 4.0, 2.1");
    }
    ...
}
```

**Multisegment Receiver Capacitance Model**

In the multisegment receiver capacitance model, the rise or fall voltage at an input or inout node is divided into N segments, where N is an integer greater than 2. For each segment, the capacitance values are stored in a lookup table using the `receiver_capacitance_rise` or `receiver_capacitance_fall` groups.

The modeling syntax is:

```plaintext
library (library_name) {
    receiver_capacitance_rise_threshold_pct ("float,...");
    receiver_capacitance_fall_threshold_pct ("float,...");
    ...
    cell(cell_name) {
        ...
        /* The receiver_capacitance group can be under a pin or timing group. */
    pin (pin_name) {
        direction : input;
        ...
        /* pin based receiver model */
```
receiver_capacitance () {
    when : boolean expression;
    mode (mode_name, mode_value);
    receiver_capacitance_rise (lu_template_name) {
        segment : "integer";
        ...
    }
    receiver_capacitance_fall (lu_template_name) {
        segment : "integer";
        ...
    }
} /* end receiver_capacitance group */

pin (pin_name) {
    direction : output;
    ...
    timing() {
        ...
        when : boolean_expression;
        mode (mode_name, mode_value);
        receiver_capacitance_rise (lu_template_name) {
            segment : "integer";
            ...
        }
        receiver_capacitance_fall (lu_template_name) {
            segment : "integer";
            ...
        }
    } /* end timing group */
} /* end pin group */
} /* end cell group */

---

Library-Level Groups and Attributes

This section describes the library-level groups and attributes for multisegment receiver capacitance modeling.

**lu_table_template Group**

The *lu_table_template* group defines the template for the *receiver_capacitance_rise* and *receiver_capacitance_fall* groups specified in the *receiver_capacitance* and *timing* groups:

The template definition includes the *variable_1, variable_2, index_1, and index_2* attributes. The *variable_1* and *variable_2* attributes specify the index variables for the lookup tables (LUTs) in these groups. The *index_1* and *index_2* attributes specify the numerical values of the variables.
The values of variable_1 and variable_2 attributes are input_net_transition and total_output_net_capacitance.

You can specify the following types of lookup tables within these groups:

- One-dimensional tables with a single index, such as input_net_transition
- Two-dimensional tables with the indexes, input_net_transition and total_output_net_capacitance
- A scalar that has a single variation value irrespective of the input_net_transition and total_output_net_capacitance values

receiver_capacitance_rise_threshold_pct and receiver_capacitance_fall_threshold_pct Attributes

The receiver_capacitance_rise_threshold_pct and receiver_capacitance_fall_threshold_pct attributes specify the points that separate the voltage rise and fall segments. Specify each point as a percentage of the rail voltage between 0.0 and 100.0.

Specify monotonically increasing values with the receiver_capacitance_rise_threshold_pct attribute, and monotonically decreasing values with the receiver_capacitance_fall_threshold_pct attribute. For example,

receiver_capacitance_rise_threshold_pct ("0, 50, 60, 70, 80, 90, 100") ;
receiver_capacitance_fall_threshold_pct ("100, 50, 40, 30, 20, 10, 0") ;

The number of segments, N, is one less than the number of points. So, N is 6. The following figure shows the segments for a voltage rise waveform.
Pin and Timing Level Groups

This section describes the pin and timing level groups and attributes for multisegment receiver capacitance modeling.

receiver_capacitance Group

At the pin level, define the receiver_capacitance_rise and receiver_capacitance_fall groups in the receiver_capacitance group.

Each of these groups specify a lookup table. Define the template of the lookup table in the library-level lu_table_template group. To use the template, specify the template name as the receiver_capacitance_rise or receiver_capacitance_fall group name.

Note:
At the timing level, define the receiver_capacitance_rise and receiver_capacitance_fall groups in the timing group.

receiver_capacitance_rise and receiver_capacitance_fall Groups

The receiver_capacitance_rise and receiver_capacitance_fall groups specify the receiver capacitance values of each of the N rise and fall voltage segments, in lookup tables. Specify N receiver_capacitance_rise groups, each for a unique segment. The receiver_capacitance_fall group has the same requirement.
The receiver_capacitance_rise and receiver_capacitance_fall groups are structurally similar to the receiver_capacitance1_rise, receiver_capacitance2_rise, receiver_capacitance1_fall, and receiver_capacitance2_fall groups and include all of their attributes and subgroups.

**segment Attribute**

The segment attribute specifies the segment that the receiver_capacitance_rise or the receiver_capacitance_fall group represents. The values vary from 1 to N.

**Conditional Data Modeling**

You can use the mode and when attributes in both the receiver_capacitance and timing groups.

**when Attribute**

The when attribute specifies the Boolean condition for the input state of the receiver capacitance timing arc.

**mode Attribute**

The complex mode attribute specifies the current mode of the cell. When defined in the receiver_capacitance group, the mode attribute specifies the receiver capacitance for the given mode. If you specify the mode attribute, you must define the mode_definition group at the cell level.

---

**Example**

The following example shows a typical multisegment receiver capacitance model.

```plaintext
library (mylib) {
    receiver_capacitance_rise_threshold_pct ("0, 50, 60, 70, 80, 90, 100");
    receiver_capacitance_fall_threshold_pct ("100, 50, 40, 30, 20, 10, 0");
    /* receiver model template */
    lu_table_template ("delay_template_8x1") {
        variable_1 : "input_net_transition";
        index_1("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8");
    }
    lu_table_template ("delay_template_8x8") {
        variable_1 : "input_net_transition";
        variable_2 : "total_output_net_capacitance";
        index_1("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8");
        index_2("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8");
    }
    ...
    cell (flop) {
        pin (D) {
```
direction : input;
/* pin-based CCS receiver model */
receiver_capacitance () {
/* default condition */
/* The old model groups are shown for comparison
receiver_capacitance1_rise(lu_template_name) {...}
receiver_capacitance2_rise(lu_template_name) {...}
receiver_capacitance1_fall(lu_template_name) {...}
receiver_capacitance2_fall(lu_template_name) {...}
*/
receiver_capacitance_rise ("delay_template_8x1") {
    segment : 1;
    values("0.11111, 0.22222, 0.33333, 0.44444, "
          0.55555, 0.66666, 0.77777, 0.88888");
}
...
receiver_capacitance_rise ("delay_template_8x1") {
    segment : 6;
    values("0.11111, 0.22222, 0.33333, 0.44444, "
          0.55555, 0.66666, 0.77777, 0.88888");
}
receiver_capacitance_fall ("delay_template_8x1") {
    segment : 1;
    values("0.11111, 0.22222, 0.33333, 0.44444, "
           0.55555, 0.66666, 0.77777, 0.88888");
}
...
receiver_capacitance_fall ("delay_template_8x1") {
    segment : 6;
    values("0.11111, 0.22222, 0.33333, 0.44444, "
           0.55555, 0.66666, 0.77777, 0.88888");
}
} /* end receiver_capacitance group */
} /* end pin D group */
pin (CLK) {
    direction : input;
    ...
}
pin (Y) {
    direction : output;
    function : "IQ";
    ...
    timing () {
        related_pin : "CLK";
        ...
    /* arc-based ccs receiver model */
    receiver_capacitance_rise ("delay_template_8x8") {
        segment : 1;
        values("0.11111, 0.22222, 0.33333, 0.44444, "
               0.55555, 0.66666, 0.77777, 0.88888, ...");
    }
    ...
    receiver_capacitance_rise ("delay_template_8x8") {

segment : 6;
values("0.11111, 0.22222, 0.33333, 0.44444, \n0.55555, 0.66666, 0.77777, 0.88888,...");
}
receiver_capacitance_fall ("delay_template_8x8") {
  segment : 1;
  values("0.11111, 0.22222, 0.33333, 0.44444, \n0.55555, 0.66666, 0.77777, 0.88888,...");
}
...
receiver_capacitance_fall ("delay_template_8x8") {
  segment : 6;
  values("0.11111, 0.22222, 0.33333, 0.44444, \n0.55555, 0.66666, 0.77777, 0.88888,...");
}
} /* end timing group */
} /* end pin Y group*/
} /* end cell flop group*/
} /* end library mylib */

## CCS Retain Arc Support

A **retain delay** is the shortest delay among all parallel arcs, from the input port to the output port. **Access time** is the longest delay from the input port to the output port. The output value is uncertain and unstable in the time interval between the retain delay and the access time. A retain arc, as shown in [Figure 10-3](#), ensures that the output does not change during this time interval.

**Figure 10-3  Retain Arc Example**

![Retain Arc Example](image)

Retain arcs:

- Guarantee that the output does not change for a certain time interval.
- Are usually defined for memory cells.
- Are not inferred as a timing check but are inferred as a delay arc.

Liberty syntax supports retain arcs in nonlinear delay models by providing the following timing groups:
• retaining_rise
• retaining_fall
• retain_rise_slew
• retain_fall_slew

---

**CCS Retain Arc Syntax**

The following syntax supports CCS timing models including the expanded and compact CCS timing models. Because retain arcs have no relation to CCS receiver models, only syntax for CCS driver models is described as follows:

**Syntax**

```plaintext
library (library_name) {
    delay_model : table_lookup;
    ...
    output_current_template(template_name) {
        variable_1: Input_net_transition;
        variable_2: total_output_net_capacitance;
        variable_3: time;
        index_1("float, ..., float"); /* optional at library level */
        index_2("float, ..., float"); /* optional at library level */
        index_3("float, ..., float"); /* optional at library level */
    }
    cell(name) {
        pin (name) {
            timing() {
                ccs_retain_rise() {
                    vector(template_name) {
                        reference_time : float;
                        index_1("float");
                        index_2("float");
                        index_3("float, ..., float");
                        values("float, ..., float");
                    }
                    vector(template_name) { . . . } ...
                }
                ccs_retain_fall() {
                    vector(template_name) {
                        reference_time : float;
                        index_1("float");
                        index_2("float");
                        index_3("float, ..., float");
                        values("float, ..., float");
                    }
                    vector(template_name) { . . . } ...
                }
            }
        }
    }
}
```
The format of the expanded CCS retain arc group is the same as the general CCS timing arcs that are defined by using the `output_current_rise` and `output_current_fall` groups.

**ccs_retain_rise and ccs_retain_fall Groups**

The `ccs_retain_rise` and `ccs_retain_fall` groups are provided in the timing group for expanded CCS retain arcs.

**vector Group**

The current vector group in the `ccs_retain_rise` and `ccs_retain_fall` groups uses the lookup table template defined by `output_current_template`. The vector group has the following parameters:

- `input_net_transition`
- `total_output_net_capacitance`
- `time`

For every value pair (such as `input_net_transition` and `total_output_net_capacitance`), there is a specified `current(time)` vector.

**reference_time Attribute**

The `reference_time` simple attribute specifies the time that the input signal waveform crosses the rising or falling input delay threshold.

**Compact CCS Retain Arc Syntax**

The compact CCS retain arc format is the same as a general compact CCS timing arc. The following retain arc syntax supports compact CCS timing.

Syntax:

```plaintext
library(my_lib) {
  ...
  base_curves (base_curves_name) {
```
compact_lut_template(template_name) {
    base_curves_group: base_curves_name;
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    variable_3 : curve_parameters;
    index_1 ("float..., float");
    index_2 ("float..., float");
    index_3 ("string..., string");
}

...
**base_curves_group Attribute**

The `base_curves_group` attribute is optional. The attribute is required when `base_curves_name` is different from that defined in the `compact_lut_template` template name.

**index_1, index_2, and index_3 Attributes**

The values for the `index_1` and `index_2` attributes are `input_net_transition` and `total_output_net_capacitance`.

The values for the `index_3` attribute must contain the following base curve parameters:

- `init_current`
- `peak_current`
- `peak_voltage`
- `peak_time`
- `left_id`
- `right_id`

**values Attribute**

The `values` attribute provides the compact CCS retain arc data values. The `left_id` and `right_id` values for compact CCS timing base curves should be integers, and they must be predefined in the `base_curves_group`.

---

**Composite Current Source Driver and Receiver Model Example**

*Example 10-3* is an example of composite current source driver and receiver model syntax.

**Example 10-3  Composite Current Source Driver and Receiver Model**

```plaintext
library(new_lib) {
  . . .
  output_current_template(CCT) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: time;
  }
  lu_table_template(LTT1) {
    variable_1: input_net_transition;
    index_1("0.1, 0.2, 0.3, 0.4");
  }
  lu_table_template(LTT2) {
```
variable_1: input_net_transition;
variable_2: total_output_net_capacitance;
index_1("1.1, 2.2");
index_2("1.0, 2.0");
}
...
cell(DFF) {
pin(D) { /* pin-based receiver model*/
direction : input;
receiver_capacitance() {
receiver_capacitance1_rise(LTT1) {
values("1.1, 0.2, 1.3, 0.4");
}
receiver_capacitance1_fall(LTT1) {
values("1.0, 2.1, 1.3, 1.2");
}
receiver_capacitance2_rise(LTT1) {
values("0.1, 1.2, 0.4, 1.3");
}
receiver_capacitance2_fall(LTT1) {
values("1.4, 2.3, 1.2, 1.1");
}
} /*end of pin (D)*/
} /*end of cell (DFF)*/
...
cell() {
...
pin (Y) {
direction : output;
capacitance : 1.2;
timing() { /* CCS and arc-based receiver model*/

... related_pin : "B";
receiver_capacitance1_rise(LTT2) {
values("0.1, 1.2" \ "3.0, 2.3");
}
receiver_capacitance1_fall(LTT2) {
values("1.1, 2.3" \ "1.3, 0.4");
}
receiver_capacitance2_rise(LTT2) {
values("1.3, 0.2" \ "1.3, 0.4");
}
receiver_capacitance2_fall(LTT2) {
values("1.3, 2.1" \ "0.4, 1.3");
}
output_current_rise() {
vector(CCT) {
reference_time : 0.05;
index_1(0.1);
index_2(1.0);
index_3("1.0, 1.5, 2.0, 2.5, 3.0");
values("1.1, 1.2, 1.5, 1.3, 0.5");
}

vector(CCT) {
  reference_time : 0.05;
  index_1(0.1);
  index_2(2.0);
  index_3("1.2, 2.2, 3.2, 4.2, 5.2");
  values("1.11, 1.31, 1.51, 1.41, 0.51");
}

vector(CCT) {
  reference_time : 0.06;
  index_1(0.2);
  index_2(1.0);
  index_3("1.2, 2.1, 3.2, 4.2, 5.2");
  values("1.0, 1.5, 2.0, 1.2, 0.4");
}

vector(CCT) {
  reference_time : 0.06;
  index_1(0.2);
  index_2(2.0);
  index_3("1.2, 2.2, 3.2, 4.2, 5.2");
  values("1.11, 1.21, 1.51, 1.41, 0.31");
}

output_current_fall() {
  vector(CCT) {
    reference_time : 0.05;
    index_1(0.1);
    index_2(1.0);
    index_3("0.1, 2.3, 3.3, 4.4, 5.0");
    values("-1.1, -1.3, -1.8, -1.4, -0.5");
  }

  vector(CCT) {
    reference_time : 0.05;
    index_1(0.1);
    index_2(2.0);
    index_3("1.2, 2.2, 3.2, 4.2, 5.2");
    values("1.11, -1.21, -1.41, -1.31, -0.51");
  }

  vector(CCT) {
    reference_time : 0.13;
    index_1(0.2);
    index_2(1.0);
    index_3("0.1, 1.3, 2.3, 3.4, 5.0");
    values("-1.1, -1.3, -1.8, -1.4, -0.5");
  }

  vector(CCT) {
    reference_time : 0.13;
    index_1(0.2);
  }
```plaintext
index_2(2.0);
index_3("1.2, 2.2, 3.2, 4.2, 5.2");
values("-1.11, -1.31, -1.81, -1.51, -0.41");
}
/*end of timing*/
. . .
} /* end of pin (Y) */
. . .
} /* end of cell */
. . .
} /* end of library */
```
Advanced Composite Current Source Modeling

This chapter provides an overview of advanced composite current source (CCS) modeling to support nanometer and very deep submicron IC development. The following composite current source modeling topics are covered:

- Modeling Cells With Advanced Composite Current Source Information
- Compact CCS Timing Model
- Variation-Aware Timing Modeling Support
Modeling Cells With Advanced Composite Current Source Information

Composite current source modeling supports additional driver model complexity by using a time- and voltage-dependent current source with essentially an infinite drive resistance. To achieve high accuracy, this driver model maps the transistor behavior for lumped loads to that for parasitic network instead of modeling the transistor behavior.

The composite current source model has better receiver model accuracy because the input capacitance of a receiver is dynamically adjusted during the transition by using two capacitance values. You can use the driver model with or without the receiver model.

Compact CCS Timing Model

Conventional CCS timing driver models require you to describe each CCS driver switching current waveform by sampling data points. As the number of timing arcs increase in a standard cell library, the CCS timing library size can become very large.

This section describes the syntax of a compact model that uses indirectly shared base curves to model the shape of switching curves. By allowing each base curve to model multiple switching curves with similar shapes, the modeling efficiency is improved and the library size is compressed.

The topics in the following sections include:

- Describing CCS timing base curves.
- Describing the syntax of base curves and the compact CCS driver modeling format.

Modeling With CCS Timing Base Curves

CCS driver switching curves in the I-V domain are smoother than those in the I(t) and V(t) domains. The I-V switching curves are usually convex, and they have no inflection point in the middle, a feature that facilitates compact modeling.

In Figure 11-1, the CCS segmentation process samples nine data points from the I(t) curve based on the tolerance. 18 floating-point numbers, that is, nine time points and nine current points are stored in the CCS library.
In Figure 11-2, the I-V curve is split into two halves at the peak, and an eleven point normalized base curve in the base curve database is selected to exactly match each half curve.

Only six parameters are required to model this inverter switching curve, reducing the storage cost. The six parameters are as follows:

- \( I_{\text{init}} \)
  - Switching current value at the starting point.

- \( I_{\text{peak}} \)
  - Peak switching current value.

- \( V_{\text{peak}} \)
  - Voltage value when current reaches peak value.

- \( T_{\text{peak}} \)
  - Time when current reaches peak value.
Left id

Reference id of the base curve that matches the left half.

Right id

Reference id of the base curve that matches the right half.

*Figure 11-2 Using Base Curves to Simplify I-V Curve Modeling*

The syntax for compact CCS timing model is as follows:

```liberty
library(my_compact_ccs_lib) {
...
```

**Compact CCS Timing Model Syntax**

In *Figure 11-2*, each switching I-V curve can be modeled by normalized base curves using the following parameters: `init_current, peak_current, peak_voltage, peak_time, left_id, and right_id`.

The `init_current, peak_current, peak_voltage, and peak_time` parameters are the critical characteristics of the curve. The `left_id` and `right_id` describe the two base curves that represent the two halves of the I-V curve.

The syntax for compact CCS timing model is as follows:

```liberty
library(my_compact_ccs_lib) {
...
```
The groups described in the following sections support compact CCS timing models.
base_curves Group

The base_curves library-level group contains the detailed description of normalized base curves. The base_curves group has the following attributes:

base_curve_type Complex Attribute

The base_curve_type attribute specifies the type of base curve. The only valid value for this attribute is ccs_timing_half_curve.

curve_x complex Attribute

The data array contains the x-axis values of the normalized base curve. Only one curve_x value is allowed for each base_curves group. See Figure 11-2 for more details.

For a ccs_timing_half_curve type base curve, the curve_x value must be between 0 and 1 and increase monotonically.

curve_y complex Attribute

Each base curve (as illustrated in Figure 11-2) is composed of one curve_x and one curve_y. You should define the curve_x base curve before curve_y for better clarity and easier implementation.

There are two data sections in the curve_y complex attribute:

- curve_id is the identifier of the base curve.
- Data array is the y-axis values of the normalized base curve.

compact_lut_template Group

The compact_lut_template group is used for compact CCS timing modeling. (The lu_table_template group is used for other timing models.) The compact_lut_template group has the following attributes:

base_curves_group Attribute

This is a required attribute in the compact_lut_template group. Its value should be a predefined base_curves group name. The base_curve_type of base_curves group determines the values in index_3. It also determines where you can use this template.

variable_1 and variable_2 Attributes

Only input_net_transition and total_output_net_capacitance are valid values for variable_1 and variable_2.
variable_3 Attribute

Only curve_parameters is a string value for this variable.

index_1 and index_2 Attributes

These are required attributes. They define the input_net_transition or total_output_net_capacitance values using the float notation.

index_3 Attribute

String values in index_3 are determined by the base_curve_type in base_curves group. When the base_curve_type is a ccs_timing_half_curve, at least six string parameters should be defined. They are init_current, peak_current, peak_voltage, peak_time, left_id, and right_id. If any of these six parameters are missing, a compilation error is issued.

compact_ccs_{rise|fall} Group

This is the compact CCS timing data in timing arc group, which has the following attributes:

base_curves_group Attribute

Defining this attribute is optional when the base_curves_name is the same as that defined in the compact_lut_template group. This group is referenced by the compact_ccs_{rise|fall} group.

values Attribute

Values of compact CCS timing data depend on how you define index_3 values.

For compact CCS timing, base curves data left_id and right_id values can only be integers.

Compact CCS Timing Library Example

library(my_lib) {

... /* normal lu table template for timing arcs */
lu_table_template (LTT2) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1  ("0.1, 0.2");
  index_2  ("1.0, 2.0");
}
base_curves (ctbct1) {
    base_curve_type : ccs_timing_half_curve;
    curve_x("0.2, 0.5, 0.8");
    curve_y(1, "0.8, 0.5, 0.2");
    curve_y(2, "0.75, 0.5, 0.35");
    ...
    curve_y(100, "0.85, 0.5, 0.15");
}

/* New lu table template for compact CCS timing model*/
compact_lut_template(LTT3) {
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    variable_3 : curve_parameters;
    index_1 ("0.1, 0.2");
    index_2 ("1.0, 2.0");
    index_3 ("init_current, peak_current, peak_voltage, peak_time, left_id, right_id");
    base_curves_group: "ctbct1";
}

...  
cell(cell_name) {
...
    pin(Y) {
        direction : output;
        capacitance : 1.2;
        timing() {
            /*compact CCS and arc-based receiver model*/
            compact_ccs_rise(LTT3) {
                base_curves_group : "ctbct1"; /* optional*/
                values("0.1, 0.5, 0.6, 0.8, 1, 3", \
                        "0.15, 0.55, 0.65, 0.85, 2, 4", \
                        "0.2, 0.6, 0.7, 0.9, 3, 2", \
                        "0.25, 0.65, 0.75, 0.95, 4, 1")
            } /*end of compact_ccs_rise() */
            compact_ccs_fall(LTT3) {
                ...
            } /*end of compact_ccs_fall() */
            ...
        } /*end of timing */
    } /*end of pin(Y) */
    } /*end of cell */
...
} /* end of library*/
Variation-Aware Timing Modeling Support

As process technologies scale to nanometer geometries, it is crucial to build variation-based cell models to solve uncertainties attributed to the variability in the device and interconnect. The CCS timing approach addresses the effects of nanometer processes by enabling advanced driver and receiver modeling.

This modeling capability supports variation parameters and is an extension of compact CCS timing driver modeling. You can even apply variation parameter models to CCS timing models.

These timing models employ a single current-based behavior that enables the concurrent analysis and optimization of timing issues. The result is a complete open-source current based modeling solution that reduces design margins and speeds design closure.

Process variation is modeled in static timing analysis tools to improve parametric yield and to control the design for corner-based analysis. This section describes the extension for variation-aware timing modeling using the existing CCS syntax.

The following sections include information about

- Variation-aware modeling for compact CCS timing driver models
- Variation-aware modeling for CCS timing receivers
- Variation-aware modeling for regular or interdependent timing constraints
- Conditional data modeling for variation-aware timing receiver models

The amount of data can be reduced by using the compact CCS timing syntax. Without compacting, variation parameter models require more library data storage. You should be familiar with the compact CCS syntax before reading this section.

Variation-Aware Compact CCS Timing Driver Model

This format supports variation parameters. It is an extension of a compact CCS timing driver modeling. The timing_based_variation groups specified in the timing group can represent variation-aware CCS driver information in a compact format. The syntax is as follows:

```
library (library_name) {
  ...
  base_curves (base_curves_name) {
    base_curve_type: enum (ccs_timing_half_curve);
    curve_x ("float,...");
    curve_y (integer, "float, ...");
    ...
  }
}
```
compact_lut_template(template_name) {
    base_curves_group : base_curves_name;
    variable_1 : input_net_transition |
    total_output_net_capacitance;
    variable_2 : input_net_transition |
    total_output_net_capacitance;
    variable_3 : curve_parameters;
    ...
}  
va_parameters(string, ...);
...

timing() {
    compact_ccs_rise(template_name) {...}
    compact_ccs_fall(template_name) {...}
    timing_based_variation() {
        va_parameters(string, ...);
        nominal_va_values(float, ...);
        va_compact_ccs_rise(template_name) {
            va_values(float, ...);
            values("..., float, ..., integer, ...");
        }
        ...
        va_compact_ccs_fall(template_name) { ...
        ...
    } /* end of timing based variation */
    } /* end of timing group */
    ...
} /* end of library group */

timing_based_variation Group
This group specifies rising and falling output transitions for variation parameters. The rising and falling output transitions are specified in va_compact_ccs_rise and va_compact_ccs_fall respectively.

- The va_compact_ccs_rise group is required only if a compact_ccs_rise group exists within a timing group.
- The va_compact_ccs_fall group is required only if a compact_ccs_fall group exists within a timing group.

va_parameters Attribute
The va_parameters attribute specifies a list of variation parameters with the following rules:

- One or more variation parameters are allowed.
- Variation parameters are represented by a string.
- Values in va_parameters must be unique.
• The `va_parameters` must be defined before being referenced by `nominal_va_values` and `va_values`.

The `va_parameters` attribute can be specified within a variation group or within a library level. `timing_based_variation` can be specified within a timing group only, and `pin_based_variation` can be specified within a pin group only. None of these can be specified within a library group.

• If the `va_parameters` attribute is specified at the library level, all cells under the library default to the same variation parameters.

• If the `va_parameters` attribute is defined in the `variation` group, all `va_values` and `nominal_va_values` under the same variation group refers to this `va_parameters`.

The attribute values can be user-defined or predefined parameters. For more information, see Example 11-1 on page 11-26.

The parameters defined in `default_operating_conditions` are process, temperature, and voltage. The voltage names are defined by using the `voltage_map` complex attribute. For more information see “`voltage_map Complex Attribute` on page 9-11.

You can use the following predefined parameters:

• For the parameters defined in `operating_conditions`, if `voltage_map` is defined, and you specify these attributes as values of `va_parameters`, these attributes are considered as user-defined variables.

• For the parameters defined in `operating_conditions`, if there is no `voltage_map` attribute at library level, and you specify these attributes as values of `va_parameters`, these attributes are taken as predefined parameters.

`nominal_va_values` Attribute

This attribute characterizes nominal values of all variation parameters.

• It is required for every `timing_based_variation` group.

• The value of this attribute has a one-to-one mapping to the corresponding `va_parameters`.

• If a nominal compact CCS driver model group and a variation-aware compact CCS driver model group are defined under the same timing group, the nominal values are applied to the nominal compact CCS driver model group.

`va_compact_ccs_rise` and `va_compact_ccs_fall` Groups

The `va_compact_ccs_rise` and `va_compact_ccs_fall` groups specify characterization corners with variation value parameters.
• These groups can be specified under different `timing_based_variation` groups if they cannot share the same `va_parameters`.

• You should characterize two corners at each side of the nominal value of all variation parameters as specified in `va_parameters`. When corners are characterized for one of the parameters, all other variations are assumed to be nominal values. Therefore, a `timing_based_variation` group with $N$ variation parameters requires exactly $2N$ characterization corners. For an example, see Example 11-2 on page 11-27.

• All variation-aware compact CCS driver model groups inside the `timing_based_variation` share the same `va_parameters`.

`va_values` Attribute

The `va_values` attribute specifies values of each variation parameter for all corners characterized in the variation-aware compact CCS driver model groups.

• Required for the variation-aware compact CCS driver model groups.

• The value of this attribute has a one-to-one mapping to the corresponding `va_parameters`.

For an example that shows how to specify `va_values` with three variation parameters, see Example 11-3 on page 11-27. In the example, the first variation parameter has a nominal value of 0.50, the second parameter has a nominal value of 1.0, and the third parameter has a nominal value of 2.0. All parameters have a variation range of -10% to +10%.

`values` Attribute

The `values` attribute follows the same rules as the nominal compact CCS driver model groups with the following exceptions:

• The `left_id` and `right_id` are optional.

• The `left_id` and `right_id` values must be used together. They can either be omitted or defined together in the `compact_lut_template`.

• If `left_id` and `right_id` are not defined in the variation-aware compact CCS driver model group, they default to the values defined in the nominal compact CCS driver model group.

`timing_based_variation` and `pin_based_variation` Groups

These groups represent variation-aware receiver capacitance information under the timing and pin groups.

• If `receiver_capacitance` group exists in pin group, variation-aware CCS receiver model groups are required in `pin_based_variation`.
• If nominal CCS receiver model groups exist in the timing group, variation-aware CCS receiver model groups are required in `timing_based_variation`.

va_parameters Attribute

The `va_parameters` attribute specifies a list of variation parameters within a `timing_based_variation` or `pin_based_variation` group. See “va_parameters Attribute” on page 11-10 for details.

nominal_va_values Attribute

The `nominal_va_values` attribute characterizes nominal values for all variation parameters. The following list describes the `nominal_va_values` attribute.

• The attribute is required in the `timing_based_variation` and `pin_based_variation` groups.
• The value of this attribute has one-to-one mapping to the corresponding `va_parameters` attribute.
• In pin-based models, if nominal CCS receiver model and variation-aware CCS receiver model groups are defined under the same pin, the nominal values are applied to the nominal CCS receiver model groups. For an example, see Example 11-5 on page 11-30.
• In timing-based models, if the nominal compact CCS driver model group and variation-aware CCS receiver model groups are defined under the same timing group, the nominal values are applied to the nominal compact CCS driver model groups.

va_receiver_capacitance1_rise, va_receiver_capacitance1_fall, va_receiver_capacitance2_rise, va_receiver_capacitance2_fall Groups

These groups specify characterization corners with variation values in `timing_based_variation` and `pin_based_variation` groups.

• You should characterize two corners at each side of the nominal value of all variation parameters specified in `va_parameters`.

  When corners are characterized for one of parameters, all other variations are assumed to be nominal value. Therefore, for a `timing_based_variation` group with N variation parameters, exactly 2N characterization corner groups are required. This same rule applies to `pin_based_variation`.

• All variation-aware CCS receiver model groups in `timing_based_variation` or `pin_based_variation` group share the same `va_parameters`.

va_values Attribute

Specifies values of each variation parameter for all corners characterized in variation-aware CCS receiver model groups.
• The attribute is required for variation-aware CCS receiver model groups.
• The value of this attribute has one-to-one mapping to the corresponding \texttt{va\_parameters} attribute.

---

**Variation-Aware CCS Timing Receiver Model**

The variation-aware CCS receiver model is expected to be used together with variation-aware compact CCS driver model. The \texttt{timing\_based\_variation} and \texttt{pin\_based\_variation} groups specify timing and pin groups respectively. In both groups, the variation-aware CCS receiver model groups are used to represent variation-aware CCS receiver information. They are defined in the following syntax and sections.

```
library() {
  ...
  lu_table_template(timing\_based\_template\_name) {
    variable\_1 : input\_net\_transition;
    variable\_2 : total\_output\_net\_capacitance;
    ...
  }
  lu_table_template(pin\_based\_template\_name) {
    variable\_1 : input\_net\_transition;
    ...
  }
  va\_parameters(string, ...);
  ...
  pin(pin\_name) {
    receiver\_capacitance() {
      receiver\_capacitance1\_rise(template\_name) {...}
      receiver\_capacitance2\_rise(template\_name) {...}
      receiver\_capacitance1\_fall(template\_name) {...}
      receiver\_capacitance2\_fall(template\_name) {...}
    }
  }
  pin_based\_variation() {
    va\_parameters(string, ...);
    nominal\_va\_values(float, ...);
    va\_receiver\_capacitance1\_rise(pin\_based\_template\_name) {
      va\_values(float, ...);
      values("float, ...", ...);
      ...
    }
    va\_receiver\_capacitance2\_rise(pin\_based\_template\_name) {...}
    va\_receiver\_capacitance1\_fall(pin\_based\_template\_name) {...}
    va\_receiver\_capacitance2\_fall(pin\_based\_template\_name) {...}
    ...
  } /* end of pin\_based\_variation */
  ...} /* end of pin */
  ...
  pin(pin\_name) {
    ...
```

---
timing() {
  receiver_capacitance1_rise(template_name) {...}
  receiver_capacitance2_rise(template_name) {...}
  receiver_capacitance1_fall(template_name) {...}
  receiver_capacitance2_fall(template_name) {...}
  timing_based_variation() {
    va_parameters(string, ...);
    nominal_va_values(float, ...);
    va_receiver_capacitance1_rise(timing_based_template_name) {
      va_values(float, ...);
      values("float, ", ...);
      ...
    }
    va_receiver_capacitance2_rise(timing_based_template_name) {
    }
    va_receiver_capacitance1_fall(timing_based_template_name) {
    }
    va_receiver_capacitance2_fall(timing_based_template_name) {
    }
  } /* end of timing_based_variation */
  ... /* end of timing */
  ... /* end of pin */
  ...
}

timing_based_variation and pin_based_variation Groups

These groups represent variation-aware receiver capacitance information under the pin or timing group level.

- If the receiver_capacitance group exists in the pin group, variation-aware CCS receiver model groups are required in the pin_based_variation group.
- If nominal CCS receiver model groups exist in the timing group, variation-aware CCS receiver model groups are required in the timing_based_variation group.

va_parameters Complex Attribute

This attribute specifies a list of variation parameters within the timing_based_variation or pin_based_variation groups. See “va_parameters Attribute” on page 11-10 for more details.

nominal_va_values Complex Attribute

This complex attribute specifies the nominal values of all variation parameters.

- The attribute is required for the timing_based_variation and pin_based_variation groups.
- The value of this attribute has one-to-one mapping to the corresponding va_parameters attribute.
• In a pin-based model, if nominal CCS receiver model and variation-aware CCS receiver model groups are defined under the same pin, the nominal values are applied to nominal CCS receiver model groups. For an example, see Example 11-5 on page 11-30.

• In a timing-based model, if nominal CCS receiver model and variation-aware CCS receiver model groups are defined under the same timing group, the nominal values are applied to nominal CCS receiver model groups.

va_receiver_capacitance1_rise, va_receiver_capacitance1_fall, va_receiver_capacitance2_rise, and va_receiver_capacitance2_fall Groups

These groups specify characterization corners with variation values in the timing_based_variation and pin_based_variation groups.

• You should characterize two corners at each side of the nominal value of all variation parameters specified in va_parameters attribute.

  When corners are characterized for one of the parameters, all other variations are assuming to be nominal value. Therefore, for a timing_based_variation group with N variation parameters, exactly 2N characterization corner groups are required. This rule also applies to pin_based_variation.

• All variation-aware CCS receiver model groups in timing_based_variation or pin_based_variation group share the same va_parameters.

va_values Attribute

Specifies values of each variation parameter for all corners characterized in variation-aware CCS receiver model groups.

• Required for variation-aware CCS receiver model groups.

• The value of this attribute has one-to-one mapping to the corresponding va_parameters attribute.

Variation-Aware Timing Constraint Modeling

This syntax supports variation parameters. It is an extension of the timing constraint modeling. It also applies to interdependent setup and hold. The timing_based_variation groups specified in the timing group represent variation-aware timing constraint sensitive information, which is defined in the following syntax:

```
library() {
 ...
  lu_table_template(template_name) {
```
variable_1 : variables;
variable_2 : variables;
variable_3 : variables;
...
}    
va_parameters(string, ...);
...
  timing () {
   ...
    interdependence_id : integer;
    rise_constraint(template_name) { ... }
    fall_constraint(template_name) { ... }
  timing_based_variation() { 
    va_parameters(string, ...);
    nominal_va_values(float, ...);
    va_rise_constraint(template_name) {
      va_values(float, ...);
      values("float, ...");
   ...  
  }    
  va_fall_constraint(template_name) { ... }
...  
}  /* end of timing_based_variation */
...
}  /* end of timing */
...
}  /* end of pin */
...
}  /* end of library */

timing_based_variation Group

The timing_based_variation group specifies the rise and fall timing constraints for variation parameters within a timing group. The rise and fall timing constraints are specified in the va_rise_constraint and va_fall_constraint groups respectively.

- The va_rise_constraint group is required only if rise_constraint group exists within a timing group.
- The va_fall_constraint group is required only if fall_constraint group exists within a timing group.

va_parameters Complex Attribute

This complex attribute specifies a list of variation parameters within timing_based_variation or pin_based_variation. See “va_parameters Attribute” on page 11-10 for details.
nominal_va_values Complex Attribute

This complex attribute is used to specify nominal values of all variation parameters. See "nominal_va_values Attribute" on page 11-11 for more information.

va_rise_constraint and va_fall_constraint Groups

The va_rise_constraint and va_fall_constraint groups specify characterization corners with variation values in timing_based_variation.

- The template name refers to the lu_table_template group.
- Both groups can be specified under different timing_based_variation groups if they cannot share the same va_parameters attribute.
- You are expected to characterize two corners at each side of the nominal value of all variation parameters as specified in va_parameters attribute.

When corners are characterized for one parameter, all other variations are assumed to be of nominal value. Therefore, for a timing_based_variation group with \( N \) variation parameters, exactly \( 2N \) characterization corners are required.

- All the va_rise_constraint and va_fall_constraint groups in the timing_based_variation group share the same va_parameters attribute.

va_values Attribute

Specifies values of each variation parameter for all corners characterized in the va_rise_constraint and va_fall_constraint groups.

- Required for the va_rise_constraint and va_fall_constraint groups.
- The value of this attribute has a one-to-one mapping to the corresponding va_parameters attribute.

Conditional Data Modeling for Variation-Aware Timing Receiver Models

Liberty provides the following syntax to support conditional data modeling for pin-based variation-aware timing receiver models:

```plaintext
library(library_name) {
  ...
  lu_table_template(timing_based_template_name) {
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    ...
  }
}  ```
lu_table_template(pin_based_template_name) {
  variable_1 : input_net_transition;
  ...
}
va_parameters(string, ...);

cell(cell_name) {
  mode_definition (mode_name) {
    mode_value(namestring) {
      when : "boolean expression";
      sdf_cond : "boolean expression";
    } ...
  } ...
}

pin(pin_name) {
  direction : input; /* or "inout" */
  receiver_cacapitance() {
    when : "boolean expression";
    mode (mode_name, mode_value);
    receiver_cacapitance1_rise (template_name) { ... }
    receiver_cacapitance1_fall (template_name) { ... }
    receiver_cacapitance2_rise (template_name) { ... }
    receiver_cacapitance2_fall (template_name) { ... }
  }
  pin_based_variation() {
    /* The "when" and "mode" attributes should be exactly the same as
    defined in the
    receiver_cacapitance group above */
    when : "boolean expression";
    mode (mode_name, mode_value);
    va_parameters(string, ...);
    nominal_va_values(float, ...);
    va_receiver_cacapitance1_rise (pin_based_template_name) { ... }
    va_receiver_cacapitance1_fall (pin_based_template_name) { ... }
    va_receiver_cacapitance2_rise (pin_based_template_name) { ... }
    va_receiver_cacapitance2_fall (pin_based_template_name) { ... }
    ...
  } /* end of pin_based_variation */
  ...
} /* end of pin group */

pin(pin_name) {
  direction : output; /* or "inout" */
  timing() {
    when : "boolean expression";
    mode (mode_name, mode_value);
    ...
    receiver_cacapitance() {
      receiver_cacapitance1_rise (template_name) { ... }
      receiver_cacapitance1_fall (template_name) { ... }
      receiver_cacapitance2_rise (template_name) { ... }
      receiver_cacapitance2_fall (template_name) { ... }
    }
    timing_based_variation() {

when Attribute

The `when` string attribute is provided in the `pin_based_variation` group to support conditional data modeling.

mode Attribute

The `mode` complex attribute is provided in the `pin_based_variation` group to support conditional data modeling. If the `mode` attribute is specified, `mode_name` and `mode_value` must be predefined in the `mode_definition` group at the cell level.

The following example shows conditional data modeling for pin-based variation-aware timing receiver models:

```liberty
library(new_lib) {
  ...
  output_current_template(CCT) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: time;
    index_1("0.1, 0.2");
    index_2("1, 2");
    index_3("1, 2, 3, 4, 5");
  }
  lu_table_template(LTT1) {
    variable_1: input_net_transition;
    index_1("0.1, 0.2, 0.3, 0.4");
  }
  lu_table_template(LTT2) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    index_1("0.1, 0.2");
    index_2("1, 2");
  }
  ...
  cell(my_cell) {
    ...
  }
}
```
... 

mode_definition(rw) {
  mode_value(read) {
    when : "I";
    sdf_cond : "I == 1";
  }
  mode_value(write) {
    when : "!I";
    sdf_cond : "I == 0";
  }
}

pin(I) { /* pin-based receiver model defined for pin 'A' */
  direction : input;
  /* receiver capacitance for condition 1 */
  receiver_capacitance() {
    when : "I"; /* or using mode as next commented line */
    /* receiver capacitance for condition 1 */
    receiver_capacitance1_rise(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance1_rise(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance1_rise(LTT1) {
      values("1, 2, 3, 4");
    }
    receiver_capacitance1_rise(LTT1) {
      values("1, 2, 3, 4");
    }
  }
  pin_based_variation ( ) {
    when : "I"; /* or using mode as next commented line */
    /* receiver capacitance for condition 1 */
    va_parameters(channel_length, threshold_voltage);
    nominal_va_values(0.5, 0.5) ;
    va_receiver_capacitance_rise(LTT1) {
      va_values(0.50, 0.45);
      values("1, 2, 3, 4");
    }
    va_receiver_capacitance_rise(LTT1) {
      va_values(0.50, 0.55);
      values("1, 2, 3, 4");
    }
    va_receiver_capacitance_rise(LTT1) {
      va_values(0.45, 0.5);
      values("1, 2, 3, 4");
    }
    va_receiver_capacitance_rise(LTT1) {
      va_values(0.55, 0.5);
      values("1, 2, 3, 4");
    }
  }
}
va_receiver_capacitance2_rise (LTT1) {
  va_values(0.50, 0.45);
  values("1, 2, 3, 4");
}

va_receiver_capacitance2_rise (LTT1) {
  va_values(0.50, 0.55);
  values("1, 2, 3, 4");
}

va_receiver_capacitance2_rise (LTT1) {
  va_values(0.45, 0.5);
  values("1, 2, 3, 4");
}

va_receiver_capacitance2_rise (LTT1) {
  va_values(0.55, 0.5);
  values("1, 2, 3, 4");
}

va_receiver_capacitance1_fall (LTT1) {
  va_values(0.50, 0.45);
  values("1, 2, 3, 4");
}

va_receiver_capacitance1_fall (LTT1) {
  va_values(0.50, 0.55);
  values("1, 2, 3, 4");
}

va_receiver_capacitance1_fall (LTT1) {
  va_values(0.45, 0.5);
  values("1, 2, 3, 4");
}

va_receiver_capacitance1_fall (LTT1) {
  va_values(0.55, 0.5);
  values("1, 2, 3, 4");
}

va_receiver_capacitance2_fall (LTT1) {
  va_values(0.50, 0.45);
  values("1, 2, 3, 4");
}

va_receiver_capacitance2_fall (LTT1) {
  va_values(0.50, 0.55);
  values("1, 2, 3, 4");
}

va_receiver_capacitance2_fall (LTT1) {
  va_values(0.45, 0.5);
  values("1, 2, 3, 4");
}

va_receiver_capacitance2_fall (LTT1) {
  va_values(0.55, 0.5);
  values("1, 2, 3, 4");
}
/* receiver capacitance for condition 2 */
receiver_capacitance() {
  when : "!I"; /* or using mode as next commented line */
  /* mode (rw, write); */
  receiver_capacitance1_rise(LTT1) {
    values("1, 2, 3, 4");
  }
  receiver_capacitance1_fall(LTT1) {
    values("1, 2, 3, 4");
  }
  receiver_capacitance2_rise(LTT1) {
    values("1, 2, 3, 4");
  }
  receiver_capacitance2_fall(LTT1) {
    values("1, 2, 3, 4");
  }
}

pin_based_variation ( ) {
  when : "!I"; /* or using mode as next commented line */
  /* mode (rw, write); */
  va_parameters(channel_length, threshold_voltage);
  nominal_va_values(0.5, 0.5);
  va_receiver_capacitance1_rise (LTT1) {
    va_values(0.50, 0.45);
    values("1, 2, 3, 4");
  }
  va_receiver_capacitance1_rise (LTT1) {
    va_values(0.50, 0.55);
    values("1, 2, 3, 4");
  }
  va_receiver_capacitance1_rise (LTT1) {
    va_values(0.45, 0.5);
    values("1, 2, 3, 4");
  }
  va_receiver_capacitance1_rise (LTT1) {
    va_values(0.55, 0.5);
    values("1, 2, 3, 4");
  }
  va_receiver_capacitance2_rise (LTT1) {
    va_values(0.50, 0.45);
    values("1, 2, 3, 4");
  }
  va_receiver_capacitance2_rise (LTT1) {
    va_values(0.50, 0.55);
    values("1, 2, 3, 4");
  }
  va_receiver_capacitance2_rise (LTT1) {
    va_values(0.45, 0.5);
    values("1, 2, 3, 4");
  }
  va_receiver_capacitance2_rise (LTT1) {
    va_values(0.55, 0.5);
    values("1, 2, 3, 4");
  }
va_receiver_capacitance2_rise (LTT1) {
  va_values(0.55, 0.5);
  values("1, 2, 3, 4");
}

va_receiver_capacitance1_fall (LTT1) {
  va_values(0.50, 0.45);
  values("1, 2, 3, 4");
}

va_receiver_capacitance1_fall (LTT1) {
  va_values(0.50, 0.55);
  values("1, 2, 3, 4");
}

va_receiver_capacitance1_fall (LTT1) {
  va_values(0.45, 0.5);
  values("1, 2, 3, 4");
}

va_receiver_capacitance2_fall (LTT1) {
  va_values(0.55, 0.45);
  values("1, 2, 3, 4");
}

pin (ZN) {
  direction : input;
  capacitance : 1.2;
  ...
  timing() {
    ...
  }
  ...
} /* end cell */

} /* end library */
Variation-Aware Compact CCS Retain Arcs

Variation-aware timing models include:

- Timing-based modeling for compact CCS timing drivers.
- Timing-based and pin-based modeling for CCS timing receivers.
- Timing-based modeling for regular or interdependent timing constraints.

Liberty provides the following syntax in the `timing_based_variation` group to support retain arcs for compact CCS driver models:

```liberty
library (library_name) {
  ...
  base_curves (base_curves_name) {
    base_curve_type: enum (ccs_timing_half_curve);
    curve_x ("float, ...");
    curve_y (integer, "float...");
    ...
  }
  compact_lut_template(template_name) {
    base_curves_group : base_curves_name;
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    variable_3 : curve_parameters;
    ...
  }
  va_parameters(string, ...);
  ...
  cell(cell_name) {
    ...
    pin(pin_name) {
      direction : string;
      capacitance : float;
      timing() {
        compact_ccs_rise(template_name) { ... }
        compact_ccs_fall(template_name) { ... }
        timing_based_variation() {
          va_parameters(string, ...);
          nominal_va_values(float, ...);
          va_compact_ccs_retain_rise(template_name) {
            va_values(float, ...);
            values ("..., float, ..., integer, "..., ...);
          }
          ...
          va_compact_ccs_retain_fall(template_name) {
            va_values(float, ...);
            values ("..., float, ..., integer, "..., ...);
          }
          ...
        }
        va_compact_ccs_rise(template_name) { ... }
      }
    }
  }
}
```
va_compact_ccs_fall(template_name) { ... } ... 
} /* end of timing_based_variation group */

... 
} /* end of pin group */

... 
} /* end of cell group */

... 
} /* end of library group*/

The format of variation-aware compact CCS retain arcs is the same as general variation-aware compact CCS timing arcs.

**va_compact_ccs_retain_rise and va_compact_ccs_retain_fall Groups**

The `va_compact_ccs_retain_rise` and `va_compact_ccs_retain_fall` groups in the `timing_based_variation` group specify characterization corners with variation value parameters for retain arcs.

**va_values Attribute**

The `va_values` attribute defines the values of each variation parameter for all corners characterized in variation-aware compact CCS retain arcs. The value of this attribute is mapped one-to-one to the corresponding `va_parameters`.

**values Attribute**

The `values` attribute follows the same rules as general variation-aware compact CCS timing models.

**Variation-Aware Syntax Examples**

*Example 11-1 va_parameters in Advanced CCS Modeling Usage*

```plaintext
library (example) {
...
operating_conditions (typical) {
  process : 1.5 ;
  temperature : 70 ;
  voltage : 2.75 ;
  ... 
}
default_operating_conditions: typical;
...
/* "temperature", "voltage" and "process" are predefined parameters, and "Vthr" is an user-defined parameter. */
```
```
va_parameters(temperature, voltage, process, Vthr, ...);
...
}
library (example) {
...
operating_conditions (typical) {
  process : 1.5 ;
  temperature : 70 ;
  voltage : 2.75 ;
...
}  
voltage_map(VDD1, 2.75);
  voltage_map(GND2, 0.2);
  default_operating_conditions: typical;
  ...  
/* "VDD1" and "GND2" are predefined parameters, and
  "voltage" is taken as an user-defined parameter. */
    va_parameters(VDD1, GND2, voltage,...);

For information about va_parameters, see “va_parameters Attribute” on page 11-10.

Example 11-2  va_compact_ccs_rise and va_compact_ccs_fall Groups

...  
timing() {
  ...
  compact_ccs_rise(temp) { /* nominal I-V waveform */
    ...
}  
timing_based_variation() {
  va_parameters(string, ... ); /* N variation parameters */
    ...
  va_compact_ccs_rise(temp) { /* 1st corner */
    ...
}  
va_compact_ccs_rise(temp) { /* last corner : total (2 * N) corners */
    ...
}  
   } /* end of timing_based_variation */
  ...
} /* end of timing */
...

For information about va_compact_ccs_rise and va_compact_ccs_fall, see
“va_compact_ccs_rise and va_compact_ccs_fall Groups” on page 11-11.

Example 11-3  va_values With Three Variation Parameters

...  
```
timing_based_variation ( ) {
    va_parameters(var1, var2, var3); /* assumed that three variation
parameters are var1, var2 and var3 */

    nominal_va_values(0.5, 1.0, 2.0);
    va(compact_ccs_rise ( ) {  // values
        va_values(0.50, 1.0, 1.8);
        ...
    }
    va(compact_ccs_rise ( ) {  // values
        va_values(0.50, 1.0, 2.2);
        ...
    }
    va(compact_ccs_rise ( ) {  // values
        va_values(0.50, 0.9, 2.0);
        ...
    }
    va(compact_ccs_rise ( ) {  // values
        va_values(0.50, 1.1, 2.0);
        ...
    }
    va(compact_ccs_rise ( ) {  // values
        va_values(0.45, 1.0, 2.0);
        ...
    }
    va(compact_ccs_rise ( ) {  // values
        va_values(0.55, 1.0, 2.0);
        ...
    }
}

For information about using va_values with the va(compact_ccs_rise and
va(compact_ccs_fall) groups, see "va(compact_ccs_rise and va(compact_ccs_fall
Groups" on page 11-11.

Example 11-4 peak_voltage in Values Attribute

library(va_ccs) {
    compact_lut_template(clt) {
        index_3 ("init_current, peak_current, peak_voltage, peak_time,\
                left_id, right_id");
        ...
    }
    voltage_map(VDD1, 3.0);
    voltage_map(VDD2, 3.5);
    voltage_map(GND1, 0.5);
    voltage_map(GND2, 0.2);
    ...
    cell(test) {
        pg_pin(v1) {
            voltage_name : VDD1;
...}
pg_pin(v2) {
  voltage_name : VDD2;
...
}
pg_pin(g1) {
  voltage_name : GND1;
...
}
pg_pin(g2) {
  voltage_name : GND2;
...
}
...
timing() {
timing_based_variation ( ) {
  va_parameters(Vthr);
  nominal_va_values(0.23);
  va_compact_ccs_rise (clt ) {
   璈 error : There are two power pins (v1 and v2)
and two ground pins (g1 and g2) in the cell "test".
The peak_voltage cannot be greater than the largest power
voltage, which is 3.5, and less than the smallest ground
voltage, which is 0.2. The value 4.0 is greater than 3.5 and 0.1
is less than 0.2. Both of them are wrong. */
  va_values(0.25);
  values("0.21, 0.54, 4.0, 0.36, 1, 2",
"0.15, 0.55, 0.1, 0.85, 2, 4", ...);
...
}
...}
timing_based_variation ( ) {
  va_parameters(Vthr, VDD2, GND2);
  nominal va_values(0.23, 3.5, 0.2);
  va_compact_ccs_rise ( clt) {
    /* In this group, the variation value of VDD2 is 4.1,
and GND2 is 0.0, so the largest power voltage is 4.1 and
the smallest ground voltage is 0.0. The peak_voltage 4.0. */
  va_values(0.18, 4.1, 0.0);
  values("0.21, 0.54, 4.0, 0.36, 1, 2",
"0.15, 0.55, 0.1, 0.85, 2, 4", ...);
...
}
...}

For information about peak_voltage in values attribute see “va_compact_ccs_rise and va_compact_ccs_fall Groups” on page 11-11.
Example 11-5  pin_based_variation Group

...  
  pin(pin_name) {  
    receiver_capacitance() {  
      receiver_capacitance1_rise(template_name) {  
        /* nominal input capacitance table */  
        /* nominal input capacitance table */  
        ...  
      }  
    }  
    pin_based_variation() {  
      nominal_va_values(2.0, 4.54, 0.23);  
      /* These nominal values apply to nominal input capacitance tables. */  
      va_receiver_capacitance1_rise(template_name) {  
        /* variational input capacitance table */  
        ...  
      }  
      ...  
    }  
  }  
  ...  
  }/* end of pin */  
  ...  

For information about peak_voltage in values attribute see “timing_based_variation and pin_based_variation Groups” on page 11-15.

Example 11-6  pin-based Model With nominal CCS receiver model and Variation-aware CCS Receiver Model Groups

/* Assume that there is no va_parameters defined */  
library(lib_name) {  
  ...  
  timing() {  
    timing_based_variation() {  
      va_compact_ccs_rise(cltdf) {  
        base_curves_group : base_name;  
        va_values(2.4)  
        /* error : can't find a corresponding va_parameters */  
        ...  
      }  
      ...  
    }  
    ...  
  }/* end of timing_based_variation */  
  ...  
}/* end of library */

/* Assume that va_parameters is defined at the end of timing_based_variation group and no default va_parameters is defined at library level */  
...  
  timing_based_variation() {  
    nominal_va_values(2.0);  
    /* error : can't find a corresponding va_parameters */  
    ...  
  } /* end of timing_based_variation */  
  ...  
  va_parameters(Vthr);  
  /* within a timing_based_variation, this is defined before
all nominal va_values and va_values attributes */
} /* end of timing_based_variation */
...

/* Assume that va_parameters is defined only at library level */
... 
library(lib_name) {
... 
timing_based_variation() {
  nominal_va_values(2.0);
  /* error : can't find a corresponding va_parameters */
... 
} 

va_parameters(Vthr);
  /* within a library, this is defined before all cell groups (or all nominal_va_values and va_values attributes) */
} /* end of library */

For information about peak_voltage in values attribute see “timing_based_variation Group” on page 11-10.

**Example 11-7 nominal_va_values in Advanced CCS Modeling Usage**

/* ASSUME that there is no voltage_map defined in library */

library (example) {
  operating_conditions (typical) {
    process : 1.5 ;
    temperature : 70 ;
    voltage : 2.75 ;
    ...
  }
  default_operating_conditions: typical;
  ...
  timing_based_variation() {
    va_parameters(voltage, temperature, process);
    nominal_va_values(2.00, 70, 1.5);
    /* error : The nominal voltage defined in default_operating_conditions is 2.75. The value 2.00 is wrong. */
  }

  /* There is voltage_map defined at library level. */
  library (example) {
    operating_conditions (typical) {
      process : 1.5 ;
      temperature : 70 ;
      voltage : 2.75 ;
      ...
    }
    voltage_map(VDD1, 2.75);
    default_operating_conditions: typical;
Example 11-8 Variational Values in Advanced CCS Modeling Usage

/* When var2 has a nominal value (1.0), var1 has three variational values (0.45, 0.55 and 0.50). However, only two values are allowed.
When var1 has a nominal value (0.50), var2 has two variational values (0.8 and 1.0). The value 0.8 is less than the nominal value (1.0).
However, the value 1.0 is not greater than the nominal value, and this is incorrect. */

/* When var2 has a nominal value (1.0), var1 has three variational values (0.45, 0.55 and 0.50). However, only two values are allowed.
When var1 has a nominal value (0.50), var2 has two variational values (0.8 and 1.0). The value 0.8 is less than the nominal value (1.0).
However, the value 1.0 is not greater than the nominal value, and this is incorrect. */

Example 11-9 Variation-Aware CCS Driver or Receiver with Timing Constraints

library(my_lib) {
    base_curves (ctbct1){
        base_curve_type : ccs_timing_half_curve;
        curve_x("0.2, 0.5, 0.8");
        curve_y(1, "0.8, 0.5, 0.2");
        curve_y(2, "0.75, 0.5, 0.35");
        curve_y(3, "0.7, 0.5, 0.45");
        ...
        curve_y(37, "0.23, 1.4, 6.23");
    }
    compact_lut_template(LUT4x4) (}
variable_1 : input_net_transition;
variable_2 : total_output_net_capacitance;
variable_3 : curve_parameters;
index_1("0.1, 0.2, 0.3, 0.4");
index_2("1.0, 2.0, 3.0, 4.0");
index_3 ("init_current, peak_current, peak_voltage, peak_time, \left_id, right_id");

base_curves_group: "ctbct1";
}
lu_table_template(LUT3) {
  variable_1 : input_net_transition;
  index_1("0.1, 0.3, 0.5");
}
lu_table_template(LUT3x3) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1("0.1, 0.3, 0.5");
  index_2("1.0, 3.0, 5.0");
}
lu_table_template(LUT5x5) {
  variable_1 : constrained_pin_transition;
  variable_2 : related_pin_transition;
  index_1("0.01, 0.05, 0.1, 0.5, 1");
  index_2("0.01, 0.05, 0.1, 0.5, 1");
}

... cell(INV1) {
  ...
  pin (A) {
    direction: input;
    capacitance: 0.3;
    receiver_capacitance ( ) {
      ...
    }
    pin_based_variation ( ) {
      va_parameters(channel_length, threshold_voltage);
      nominal_va_values(0.5, 0.5);
      va_receiver_capacitance1_rise (LUT3) {
        va_values(0.50, 0.45);
        values("0.29, 0.30, 0.31");
      }
      va_receiver_capacitance1_rise (LUT3) {
        va_values(0.50, 0.55);
        ...
      }
      va_receiver_capacitance1_rise (LUT3) {
        va_values(0.45, 0.50);
        ...
      }
      va_receiver_capacitance1_rise (LUT3) {
        va_values(0.55, 0.50);
        ...
      }
    }
  }
...
va_receiver_capacitance2_rise (LUT3) {
  va_values(0.50, 0.45);
  values("0.19, 8.60, 5.41");
}
va_receiver_capacitance2_rise (LUT3) {
  va_values(0.50, 0.55);
  ...
}
va_receiver_capacitance2_rise (LUT3) {
  va_values(0.45, 0.50);
  ...
}
va_receiver_capacitance2_rise (LUT3) {
  va_values(0.55, 0.50);
  ...
}
va_receiver_capacitance1_fall (LUT3) {
  va_values(0.50, 0.45);
  values("0.53, 2.16, 9.18");
}
va_receiver_capacitance1_fall (LUT3) {
  va_values(0.50, 0.55);
  ...
}
va_receiver_capacitance1_fall (LUT3) {
  va_values(0.45, 0.50);
  ...
}
va_receiver_capacitance1_fall (LUT3) {
  va_values(0.55, 0.50);
  ...
}
va_receiver_capacitance2_fall (LUT3) {
  va_values(0.50, 0.45);
  values("0.39, 0.98, 5.15");
}
va_receiver_capacitance2_fall (LUT3) {
  va_values(0.50, 0.55);
  ...
}
va_receiver_capacitance2_fall (LUT3) {
  va_values(0.45, 0.50);
  ...
}
va_receiver_capacitance2_fall (LUT3) {
  va_values(0.55, 0.50);
  ...
}

}  /* end of pin */

pin (Y) {
  direction: output;
timing () {
    related_pin: "A";
    compact_ccs_rise (LUT4x4) {
        ...
    }
    compact_ccs_fall (LUT4x4) {
        ...
    }
    timing_based_variation() {
        va_parameters(channel_length, threshold_voltage);
        nominal_va_values(0.50, 0.50);
        va_compact_ccs_rise (LUT4x4) { /* without optional fields */
            va_values(0.50, 0.55);
            values("0.1, 0.5, 0.6, 0.8, 1, 3", \  
                "0.15, 0.55, 0.65, 0.85, 2, 4", \  
                "0.2, 0.6, 0.7, 0.9, 3, 2", \  
                "0.1, 0.2, 0.3, 0.4, 1, 3", \  
                "0.2, 0.3, 0.4, 0.5, 4, 5", \  
                "0.3, 0.4, 0.5, 0.6, 2, 4", \  
                "0.4, 0.5, 0.6, 0.7, 7, 8", \  
                "0.5, 0.6, 0.7, 0.8, 10, 4", \  
                "0.5, 0.6, 0.8, 0.9, 11, 2", \  
                "0.25, 0.55, 1.65, 1.85, 3, 4", \  
                "1.2, 1.6, 1.7, 1.9, 5, 2", \  
                "1.1, 2.2, 2.3, 0.4, 1.30", \  
                "1.2, 2.3, 1.4, 0.5, 17, 5", \  
                "1.3, 2.4, 1.5, 0.6, 22, 24", \  
                "1.4, 2.5, 1.6, 1.7, 17,18", \  
                "1.5, 2.6, 0.7, 0.8, 10,33");
        }
        va_compact_ccs_rise (LUT4x4) {
            va_values(0.50, 0.55);
            ...
        }
        va_compact_ccs_rise (LUT4x4) {
            va_values(0.45, 0.50);
            ...
        }
        va_compact_ccs_rise (LUT4x4) {
            va_values(0.55, 0.50);
            ...
        }
        va_compact_ccs_fall (LUT4x4) { /* without optional fields */
            ...
        }
    }
    /* end of timing_based_variation */
    ...
} /* end of timing */
    ...
} /* end of pin */
    ...
} /* end of cell */
...  

cell(INV4) {
...  
  pin (Y) {
    direction: output;
    timing ( ) {
      related_pin: "A";
      receiver_capacitance1_rise (LUT3x3) {
        ...
      }
      receiver_capacitance2_rise (LUT3x3) {
        ...
      }
      receiver_capacitance1_fall (LUT3x3) {
        ...
      }
      receiver_capacitance2_fall (LUT3x3) {
        ...
      }
      rise_constraint(LUT5x5) {
        ...
      }
      fall_constraint(LUT5x5) {
        ...
      }
      timing_based_variation ( ) {
        va_parameters(channel_length,threshold_voltage);
        nominal_va_values(0.50, 0.50);
        va_receiver_capacitance1_rise (LUT3x3) {
          va_values(0.50, 0.45);
          values("1.10, 1.20, 1.30", \
              "1.11, 1.21, 1.31", \
              "1.12, 1.22, 1.32");
        }
        va_receiver_capacitance2_rise (LUT3x3) {
          va_values(0.50, 0.45);
          values("1.20, 1.30, 1.40", \
              "1.21, 1.31, 1.41", \
              "1.22, 1.32, 1.42");
        }
        va_receiver_capacitance1_fall (LUT3x3) {
          va_values(0.50, 0.45);
          values("1.10, 1.20, 1.30", \
              "1.11, 1.21, 1.31", \
              "1.12, 1.22, 1.32");
        }
        va_receiver_capacitance2_fall (LUT3x3) {
          va_values(0.50, 0.45);
          values("1.20, 1.30, 1.40", \
              "1.21, 1.31, 1.41", \
              "1.22, 1.32, 1.42");
        }
      }
    }
  }
}
va_receiver_capacitance1_rise (LUT3x3) {
    va_values(0.50, 0.55);
    ...
}
...
va_receiver_capacitance1_rise (LUT3x3) {
    va_values(0.45, 0.50);
    ...
}
va_receiver_capacitance1_rise (LUT3x3) {
    va_values(0.55, 0.50);
    ...
}
...
va_rise_constraint(LUT5x5) {
    va_values(0.50, 0.55);
    values( "-0.1452, -0.1452, -0.1452, -0.1452, 0.3329", \
            "-0.1452, -0.1452, -0.1452, -0.1452, 0.3952", \
            "-0.1245, -0.1452, -0.1452, -0.1358, 0.5142", \
            "0.05829, 0.0216, 0.01068, 0.06927, 0.723", \
            "1.263, 1.227, 1.223, 1.283, 1.963");
}
va_rise_constraint(LUT5x5) {
    va_values(0.50, 0.55);
    ...
}
va_rise_constraint(LUT5x5) {
    va_values(0.55, 0.50);
    ...
}
va_rise_constraint(LUT5x5) {
    va_values(0.45, 0.50);
    ...
}
va_fall_constraint(LUT5x5) {
    va_values(0.50, 0.55);
    ...
}
va_fall_constraint(LUT5x5) {
    va_values(0.55, 0.50);
    ...
}
va_fall_constraint(LUT5x5) {
    va_values(0.45, 0.50);
    ...
}
va_fall_constraint(LUT5x5) {
    va_values(0.50, 0.45);
    ...
}
va_fall_constraint(LUT5x5) {
    va_values(0.55, 0.45);
    ...
}
va_fall_constraint(LUT5x5) {
    va_values(0.45, 0.45);
    ...
}
va_fall_constraint(LUT5x5) {
    va_values(0.50, 0.45);
    ...
}
/* end of timing_based_variation */
} /* end of timing */
...
} /* end of pin */
...
} /* end of cell */
...
} /* end of library */
This chapter provides an overview of modeling noise to support gate-level static noise analysis. It covers various topics on modeling noise for calculation, detection, and propagation, in the following sections:

- Modeling Noise Terminology
- Modeling Cells for Noise
- Representing Noise Calculation Information
- Representing Noise Immunity Information
- Representing Propagated Noise Information
- Examples of Modeling Noise
**Modeling Noise Terminology**

A net can be either an aggressor or a victim:

- An aggressor net is a net that injects noise onto a victim net.
- A victim net is a net onto which noise is injected by one or more neighboring nets through the cross-coupling capacitors between the nets.

Noise effect can be categorized in two ways:

- Delay noise
- Functional noise

Delay noise occurs when victim and aggressor nets switch simultaneously. This activity alters the delay and slew of the victim net.

Functional noise occurs when a victim net is intended to be at a stable value and the noise injected onto this net causes it to glitch. The glitch might propagate to a state element, such as a latch, altering the circuit state and causing a functional failure.

To compute and detect any delay or functional noise failure, the following are calculated:

- Noise calculation
- Noise immunity
- Noise propagation

---

**Noise Calculation**

Coupled noise is the noise voltage induced at the output of nonswitching gates when coupled adjacent drivers to the output (aggressor drivers) are switching.

---

**Noise Immunity**

The main concept of noise immunity is that for most cells, a glitch on the input pin has to be greater than a certain fixed voltage to cause a failure. However, a glitch with a tall height might still not cause any failure if the glitch width is very small. This is mainly because noise failure is related to input noise glitch energy and this energy is proportional to the area under the glitch waveform.

For example, if a large voltage glitch in terms of height and width occurs on the clock pin of a flip-flop, the glitch can cause a change in the data and therefore the flip-flop output might change.
**Noise Propagation**

Propagated noise is the noise waveform created at the output of nonswitching gates due to the propagation of noise from the inputs of the same gate.

**Modeling Cells for Noise**

Library information for noise can be characterized in the following ways:

- **I-V Characteristics and Drive Resistance**
- **Noise Immunity**
- **Noise Propagation**

**I-V Characteristics and Drive Resistance**

To calculate the coupled noise glitch on a victim net, you need to know the effective steady-state drive resistance of the net. Figure 12-1 on page 12-4 shows the four different types of noise glitch:

- **Vh+**: Input is high, and the noise is over the high voltage rail.
- **Vh-**: Input is high, and the noise is less than the high voltage rail.
- **Vl+**: Input is low, and the noise is over the low voltage rail.
- **Vl-**: Input is low, and the noise is less than the low voltage rail.

Because the current is a nonlinear function of the voltage, you need to characterize the steady-state I-V characteristics curve, which provides a more accurate view of the behavior of a cell in its steady state. This information is specified for every timing arc of the cell that can propagate a transition. If an I-V curve cannot be obtained for a specific arc, the steady-state drive resistance single value can be used, but it is less accurate than the I-V curve.
Figure 12-1  Noise Glitch and Steady-State Drive Resistance

Figure 12-2 on page 12-5 is an example of two I-V curves and the steady-state resistance value.
Noise Immunity

Circuits can tolerate large glitches at their inputs and still work correctly if the glitches deliver only a small energy. Given this concept, each cell input can be characterized by application of a wide range of coupling voltage waveform stimuli on it. Figure 12-3 shows a glitch noise model.
One method of modeling the noise immunity curve involves applying coupling voltage waveform stimuli with various heights (in library voltage units) and widths (in library time units) to the cell input, and then observing the output voltage waveform. The exact set of input stimuli (in terms of height and width) that produces an output noise voltage height equal to a predefined voltage is on the noise immunity curve. This predefined voltage is known as the cell failure voltage. Any input stimulus that has a height and width above the noise immunity curve causes a noise voltage higher than the cell failure voltage at the output and produces a functional failure in the cell.

Figure 12-4 shows an example of a noise immunity curve.

As shown in Figure 12-4, any noise width and height combination that falls above the noise rejection curve causes functional failure. The selection of cell failure voltage is important for noise immunity curve characterization.
There are many ways to select a failure voltage for a cell that produces usable noise immunity curves, including the following:

- $V_{\text{failure}}$ equal to the output DC noise margin
- $V_{\text{failure}}$ equal to the next cell’s $V_{\text{IL}}$ or $(V_{\text{cc}} - V_{\text{IH}})$
- $V_{\text{failure}}$ corresponding to the point on the DC transfer curve where $\frac{dV_{\text{out}}}{dV_{\text{in}}}$ is 1.0 or $-1.0$
- $V_{\text{failure}}$ corresponding to the point on the DC transfer curve where $\frac{V_{\text{out}}}{V_{\text{in}}}$ is less than 1.0 or $-1.0$
- $V_{\text{failure}}$ corresponding to the point on the DC transfer curve where $\frac{V_{\text{out}}}{V_{\text{in}}}$ is 1.0 or $-1.0$

Figure 12-5  Different Failure Voltage Criteria for Noise Immunity Curve

The noise immunity curve can also be a function of output loads, where cells with larger output loads can tolerate greater input noise.

The noise immunity curve is also state-dependent. For example, the noise on the A-to-Z arc of an XOR gate when $B = 0$ might be different from the B-to-Z arc when $B = 1$, because the arcs might go through different sets of transistors.
Using the Hyperbolic Model

The noise immunity curve resembles a hyperbola, because the area of different noise along the hyperbola is constant. Therefore noise immunity can be defined as a hyperbolic function with only three coefficients for every input on an I/O library pin. The formula for the height based on these three coefficients is as follows:

\[ \text{height} = \text{height}_\text{coefficient} + \frac{\text{area}_\text{coefficient}}{(\text{width} - \text{width}_\text{coefficient})}; \]

Your tool gets these coefficients from the library and applies the calculated height and width to determine whether the noise can cause functional failure. Any point above the hyperbolic curve signifies a functional failure.

Noise Propagation

Propagated noise from the input to the output of a cell is modeled by

- Output glitch height
- Output glitch width
- Output glitch peak time ratio
- Output load

Figure 12-6 illustrates basic noise characteristics.

Figure 12-6  Basic Noise Characteristics

The output noise width, height, and peak-time ratio depend on the input noise width, height, and peak-time ratio as well as on the output load. However, in some cases, the dependency on peak-time ratio can be negligible; therefore, to reduce the amount of data, the lookup table does not have a peak-time-ratio dependency.
Table 12-1 shows a summary of the syntax used to model cases when the cell is not switching.

### Table 12-1  Summary of Library Requirements for Noise Model

<table>
<thead>
<tr>
<th>Category</th>
<th>Model type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noise detection</td>
<td>Voltage ranges (DC noise margin)</td>
<td><code>input_voltage</code> or <code>output_voltage</code> defined for all library pins</td>
</tr>
<tr>
<td></td>
<td>Hyperbolic noise immunity curves</td>
<td>Four hyperbolic curves; each has three coefficients, defined for input or bidirectional library pins</td>
</tr>
<tr>
<td></td>
<td>Noise immunity tables</td>
<td>Four tables indexed by noise width and output load defined for timing arcs</td>
</tr>
<tr>
<td></td>
<td>Noise immunity polynomials</td>
<td>Four polynomials as a function of noise width and output load defined for timing arcs</td>
</tr>
<tr>
<td>Noise calculation</td>
<td>Steady-state resistances</td>
<td>Four floating-point values defined for timing arcs</td>
</tr>
<tr>
<td></td>
<td>I-V characteristics tables</td>
<td>Two tables indexed by output steady-state voltage for non-three-state arcs and one table for three-state arcs</td>
</tr>
<tr>
<td></td>
<td>I-V characteristics polynomials</td>
<td>Two polynomials as a function of output steady-state voltage for non-three-state arcs and one table for three-state arcs</td>
</tr>
<tr>
<td>Noise propagation</td>
<td>Noise propagation tables</td>
<td>Four pairs of noise width and height tables, each indexed by noise width, height, and load</td>
</tr>
<tr>
<td></td>
<td>Noise propagation polynomials</td>
<td>Four sets of three polynomials (width, height, and peak-time ratio), each a function of width, height, peak-time ratio, and load</td>
</tr>
</tbody>
</table>
Representing Noise Calculation Information

You can represent coupled noise information with an I-V characteristics lookup table model or polynomial model at the timing level or four simple attributes defined at the timing level:

- steady_state_resistance_above_high
- steady_state_resistance_below_low
- steady_state_resistance_high
- steady_state_resistance_low

I-V Characteristics Lookup Table Model

You can describe I-V characteristics in your libraries by using lookup tables. To define your lookup tables, use the following groups and attributes:

- The iv_lut_template group in the library group
- The steady_state_current_high, steady_state_current_low, and steady_state_current_tristate groups in the timing group

iv_lut_template Group

Use this library-level group to create templates of common information that multiple lookup tables can use. A table template specifies the I-V output voltage and the breakpoints for the axis. Assign each template a name. Make the template name the group name of a steady_state_current_low, steady_state_current_high, or steady_state_current_tristate group.

Syntax

```plaintext
library(namestring) {
  ...
  iv_lut_template(template_namestring) {
    variable_1: iv_output_voltage;
    index_1 ("float,..., float");
  }
  ...
}
```
Template Variables
To specify I-V characteristics, define the following variable and index:

variable_1
The only valid value is iv_output_voltage, which specifies the I-V voltage of the output pin specified in the pin group. The voltage is measured from the pin to the ground.

index_1
The index values are a list of floating-point numbers that can be negative or positive. The values in the list must be in increasing order. The number of floating-point numbers in the index_1 variable determines the dimension.

Example
iv_lut_template(my_current_low) {
variable_1 : iv_output_voltage;
index_1 ("-1, -0.1, 0, 0.1 0.8, 1.6, 2");
}
iv_lut_template(my_current_high) {
variable_1 : iv_output_voltage;
index_1("-1, 0, 0.3, 0.5, 0.8, 1.5, 1.6, 1.7, 2");
}

Defining the Lookup Table Steady-State Current Groups
To specify the I-V characteristics curve for the nonlinear table model, use the steady_state_current_high, steady_state_current_low, or steady_state_current_tristate groups within the timing group.

Syntax for Table Model
timing() { /* for non-three-state arcs */
steady_state_current_high(template_namestring) {
values("float,..., float");
}
steady_state_current_low(template_namestring) {
values("float,..., float");
}
...
}
timing() { /* for three-state arcs */
steady_state_current_tristate(template_namestring) {
values("float,..., float");
}
}
...
The values are floating-point numbers indicating values for current.

The following rules apply to lookup table groups:

- Each table must have an associated name for the `iv_lut_template` it uses. The name of the template must be identical to the name defined in a library `iv_lut_template` group.
- You can overwrite `index_1` in a lookup table, but the overwrite must come before the definition of values.
- The current values of the table are stored in a `values` attribute. The values can be negative.

Example

```plaintext
timing() {
  ...
  steady_state_current_low(my_current_low) {
    values("-0.1, -0.05, 0, 0.1, 0.25, 1, 1.8");
  }
  steady_state_current_high(my_current_high) {
    values("-2, -1.8, -1.7, -1.4, -1, -0.5, 0, 0.1, 0.8");
  }
}
```

I-V Characteristics Curve Polynomial Model

As with the lookup table model, you can describe an I-V characteristics curve in your libraries by using the polynomial representation. To define your polynomial, use the following groups and attributes:

- The `poly_template` group in the `library` group
- The `steady_state_current_high`, `steady_state_current_low`, and `steady_state_current_tristate` groups within the `timing` group

poly_template Group

You can define a `poly_template` group at the library level to specify the equation variables, the variable ranges, the voltages mapping, and the piecewise data. The valid values for the variables are extended to include `iv_output_voltage`, `voltage`, `voltage_i`, and `temperature`.
Syntax

```
library(name_string) {
...
  poly_template(template_name_string) { 
    variables(variable_1_enum, ..., variable_n_enum);
    variable_i_range: (float, float);
    ... 
    variable_n_range: (float, float);
    mapping(voltage_enum, power_rail_id);
    domain(domain_name_string) {
      variable_i_range: (float, float);
      ... 
      variable_n_range: (float, float);
    }
    ... 
    } 
  } 
...
```

The syntax of the `poly_template` group is the same as that used for the delay model, except that the variables used in the format are

- `iv_output Voltage` for the output voltage of the pin
- `voltage`, `voltage1`, `temperature`

The piecewise model through the `domain` group is also supported.

Example

```
poly_template (my_current_low) {
  variables (iv_output_voltage, voltage, voltage1, temperature);
  mapping(voltage1, VDD2);
  variable_1_range (-1, 2);
  variable_2_range (1.4, 1.8);
  variable_3_range (1.1, 1.5);
  variable_4_range (-40, 125);
}
```

---

**Defining Polynomial Steady-State Current Groups**

To specify the I-V characteristics curve to define the polynomial, use the `steady_state_current_high`, `steady_state_current_low`, and `steady_state_current_tristate` groups within the `timing` group.
Syntax for Polynomial Model

timing { /* for non-three-state arcs */
  steady_state_current_high(template_namestring) {
    orders("integer,..., integer");
    coefs("float,..., float");
    domain(domain_namestring) {
      orders("integer,..., integer");
      coefs("float,..., float");
    }
    ...
  }
  steady_state_current_low(template_namestring) {
    ...
  }
  steady_state_current_tristate(template_namestring) {
   ...
  }
}

timing() { /* for three-state arcs */
  steady_state_current_tristate(template_namestring) {
    ...
  }
  ...
}

The orders, coefs, and variable_range attributes represent the polynomial for the current for high, low, and three-state.

The output voltage, temperature, and any power rail of the cell are allowed as variables for steady_state_current groups.

Example

timing() {
  steady_state_current_low(my_current_low) {
    orders (*3, 3, 0, 0*);
    coefs (*8.4165, 0.3198, -0.0004, 0.0000, \ 1133.8274, 8.7287, -0.0054, 0.0000, \ 139.8645, -60.3898, 0.0589, -0.0000, \ -167.4473, 95.7112, -0.1018, 0.0000*);
  }
  steady_state_current_high(my_current_high) {
    orders (*3, 3, 0, 0*);
    coefs (*10.9165, 0.2198, -0.0003, 0.0000, \ 1433.8274, 8.7287, -0.0054, 0.0000, \ 128.8645, -60.3898, 0.0589, -0.0000, \ -167.4473, 95.7112, -0.1018, 0.0000*);
  }
  ...
}

Using Steady-State Resistance Simple Attributes

To represent steady-state drive resistance values, use the following attributes to define the four regions:

- `steady_state_resistance_above_high`
- `steady_state_resistance_below_low`
- `steady_state_resistance_high`
- `steady_state_resistance_low`

These attributes are defined within the `timing` group to represent the steady-state drive resistance. If one of these attributes is missing, the model becomes inaccurate.

Syntax

```plaintext
pin(name) {
  ...
  timing() {
    ...
    steady_state_resistance_above_high : float;
    steady_state_resistance_below_low   : float;
    steady_state_resistance_high      : float;
    steady_state_resistance_low       : float;
    ...
  }
  ...
}
```

*float*

The value of steady-state resistance for the four different noise regions in the I-V curve.

Example

```plaintext
steady_state_resistance_above_high : 200.0;
steady_state_resistance_below_low  : 100.0;
steady_state_resistance_high      : 100.0;
steady_state_resistance_low       : 1100.0;
```

Using I-V Curves and Steady-State Resistance for tied_off Cells

In tied-off cells, the output pins are tied to either high or low and there is no need to define timing information for related pins. The tied-off cells have been enhanced to accept I-V curve and steady-state resistance in the `timing` group. To specify only the noise data (I-V curves and steady-state resistance) in the `timing` group, you must specify a new Boolean attribute, `tied_off`, and set it to true.
Defining tied_off Attribute Usage

You can specify the I-V characteristics and steady-state drive resistance values on tied-off cells by using the `tied_off` attribute in the `timing` group.

**Syntax**

```
pin(name) {
    ...
    timing() {
        ...
        tied_off : boolean;
        /* timing type is not defined */
        /* steady-state resistance */
    }
}
```

The following rules apply to `tied_off` cells:

- Steady-state resistance and I-V curves can coexist in the same timing arc of a `tied_off` output pin.
- If the output pin is tied to low (function: "0") and its timing arc specifies the `steady_state_current_high` group, an error is generated. Similarly if the output pin is tied to high (function: "1") and its timing arc specifies the `steady_state_current_low` group, an error is generated.
- If noise immunity and noise propagation are specified in the timing arcs of a `tied_off` pin, an error is generated.
- If the `related_pin` attribute is specified on a `tied_off` output pin, an error is generated.

**Example**

```
pin (high) {
    direction : output;
    capacitance : 0;
    function : "1";
    
    /* noise information */
    timing() {
        tied_off : true;
        steady_state_resistance_high : 1.22;
        steady_state_resistance_above_high : 1.00;
        steady_state_current_high(iv1x5) {
            index_1("0.3,0.75,1.0,1.2,2");
            values("-513.2,-447.9,-359.3,-245.7,497.3");
        }
    }
}
```
Representing Noise Immunity Information

In the Liberty syntax, you can represent noise immunity information with a
• Lookup table or a polynomial model at the timing level
• Input noise width range at the pin level
• Hyperbolic model at the pin level

Noise Immunity Lookup Table Model

You can represent noise immunity in your libraries by using lookup tables. To define your lookup tables, use the following groups and attributes:
• noise_lut_template group in the library group
• noise_immunity_above_high, noise_immunity_above_low, noise_immunity_below_low, and noise_immunity_high groups in the timing group

noise_lut_template Group

Use this library-level group to create templates of common information that multiple noise immunity lookup tables can use.

A table template specifies the input noise width, the output load, and their corresponding breakpoints for the axis. Assign each template a name, and make the name the group name of a noise immunity group.

Syntax

library(name_string) {
  ...
  noise_lut_template(template_name_string) {
    variable_1: value;
    variable_2: value;
    index_1 ("float,..., float");
    index_2 ("float,..., float");
  }
  ...
}

Template Variables

The library-level table template specifying noise immunity can have two variables (variable_1 and variable_2). The variables indicate the parameters used to index the lookup table along the first and second table axes. The parameters are input_noise_width and total_output_net_capacitance.
The index values in \texttt{index\_1} and \texttt{index\_2} are a list of positive floating-point numbers. The values in the list must be in increasing order.

The unit for the input noise width is the library time unit.

**Example**

\begin{verbatim}
 noise_lut_template(my_noise_reject) {
    variable\_1 : input_noise_width;
    variable\_2 : total_output_net_capacitance;
    index\_1("0, 0.1, 0.3, 1, 2");
    index\_2("1, 2, 3, 4, 5");
}
\end{verbatim}

**Defining the Noise Immunity Table Groups**

To represent noise immunity, use the \texttt{noise\_immunity\_above\_high}, \texttt{noise\_immunity\_below\_low}, \texttt{noise\_immunity\_high}, and \texttt{noise\_immunity\_low} groups within the \texttt{timing} group.

**Syntax for Noise Immunity Table Model**

\begin{verbatim}
 timing() {
    noise\_immunity\_above\_high(template\_name\_string) {
        index\_1 ("float,..., float");
        index\_2 ("float,..., float");
        values("float,...,float"..."float,...,float");
    }
    noise\_immunity\_below\_low(template\_name\_string) {
        ...
    }
    noise\_immunity\_high(template\_name\_string) {
        ...
    }
    noise\_immunity\_low(template\_name\_string) {
        ...
    }
}
\end{verbatim}

The following rules apply to the noise immunity groups:

- These tables are optional, and each of them can exist separately on the library timing arcs.
- Each noise immunity table has an associated name for the \texttt{noise\_lut\_template} it uses. The name of the table must be identical to the name defined in a library \texttt{noise\_lut\_template} group.
• Each table is two-dimensional. The indexes are `input_noise_width` and `total_output_net_capacitance`. The values in the table are the noise heights (that is, height as a function of width and output load).

• You can overwrite any or both indexes in a noise template. However, the overwrite must occur before the actual definition of the values.

• The height values of the table are stored in the `values` attribute. Each height value is the absolute difference of the noise bump height voltage and the related rail voltage and is, therefore, a positive number. Any point over this curve describes a height and width combination that causes functional failure.

• The unit for the height is the library voltage unit.

• For points outside table ranges, extrapolation can be used.

**Example**

```plaintext
pin ( Y ) {
  ....
  timing () {
    noise_immunity_below_low (my_noise1) {
      values ("1, 0.8, 0.5", "1, 0.8, 0.5", "1, 0.8, 0.5");
    }
    noise_immunity_above_high (my_noise1){
      values ("1, 0.8, 0.5", "1, 0.8, 0.5", "1, 0.8, 0.5");
    }
  }
}
```

**Noise Immunity Polynomial Model**

As with the lookup table model, you can represent noise immunity in your libraries by using the polynomial representation. To define your polynomial, use the following groups and attributes:

• The `poly_template` group in the library group

• The `noise_immunity_above_high`, `noise_immunity_below_low`, `noise_immunity_high`, and `noise_immunity_low` groups in the `timing` group
**poly_template Group**

You can define a `poly_template` group at the library level to specify the polynomial equation variables, the variable ranges, the voltage mapping, and the piecewise data. The valid values for the variables include `total_output_net_capacitance`, `input_noise_width`, `voltage`, `voltage1`, and `temperature`.

**Syntax**

```plaintext
library(namestring) {
    ...
    poly_template(template_namestring) {
        variables(variable_1, ..., variable_n);
        variable_i_range: (float, float);
        ...  
        variable_n_range: (float, float);
        mapping(voltage_enum, power_rail_id);
        domain(domain_namestring); {
            variable_i_range: (float, float);
            ...  
            variable_n_range: (float, float);
        }
    }
    ...
}
```

**Template Variables**

The syntax of the `poly_template` group is the same as that used for the delay model, except that the variables used in the format are

- `input_noise_width`
- `total_output_net_capacitance`
- `voltage, voltage1, temperature`

The piecewise model through the `domain` group is also supported.

**Example**

```plaintext
poly_template (my_noise_reject) { /* existing syntax */
    variables (input_noise_width, voltage, voltage1, temperature, \
        total_output_net_capacitance);
    mapping(voltage1, VDD2);
    variable_1_range (0, 2);
    variable_2_range (1.4, 1.8);
    variable_3_range (1.1, 1.5);
    variable_4_range (-40, 125);
    variable_5_range (0.01, 1.0);
    domain (typ) {
        variables (input_noise_width, voltage, voltage1, \
```
Defining the Noise Immunity Polynomial Groups

To represent noise immunity, use the `noise_immunity_above_high`, `noise_immunity_below_low`, `noise_immunity_high`, and `noise_immunity_low` groups within the `timing` group.

Syntax

```plaintext
... timing() { 
    noise_immunity_above_high(template_name_string) { 
        orders("integer,..., integer"); 
        coefs("float,..., float"); 
        ... 
        domain(domain_name_string) { 
            orders("integer,..., integer"); 
            coefs("float,..., float"); 
        } 
        ... 
    } 
    noise_immunity_below_low(template_name_string) { 
        ... 
    } 
    noise_immunity_high(template_name_string) { 
        ... 
    } 
    noise_immunity_low(template_name_string) { 
        ... 
    } 
    ... 
}
```

Because the polynomial model is a superset of the lookup table model, all syntax supported in the lookup table is also supported in the polynomial model. For example, you can have a polynomial `noise_immunity_high` and a table `noise_immunity_low` defined in the same group in a scalable polynomial delay model library.
Example

```liberty
noise_immunity_low (my_noise_reject) {
    domain (typ) {
        orders (*1, 1, 1, 1, 1*)
        coefs (*1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, \ 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, \ 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0* );
    } domain (min) {
        orders("1 3 1 1");
        coefs("-0.01, 0.02, 1.41, -0.54, \ 1.85, 1.83, -5.58, -2.96, -0.0001, \ 0.0001, -0.002, -0.0019, 0.002, \ 0.0012, -0.010, -0.0061, 0.034, \ 0.015, 2.08, -0.22, 4.13, 2.44, \ -14.02, -7.83, 7.09e-05, -1.98e-05, \ -0.0019, 0.0009, 0.0065, -0.0004, \ -0.027, -0.016");
    }
}
```

Input Noise Width Ranges at the Pin Level

To specify whether a noise immunity or propagation table is referenced within the noise range indexes, the Liberty syntax allows you to specify the minimum and maximum values of the input noise width.

Defining the input_noise_width Range Limits

You can specify two float attributes, `min_input_noise_width` and `max_input_noise_width`, at the pin level. These attributes are optional and specify the minimum and maximum values of the input noise width.

Syntax

```liberty
pin(name_string) {
    ... /* used for noise immunity or propagation */
    min_input_noise_width : float;
    max_input_noise_width: float;
    ... 
}
```

float

The values of `min_input_noise_width` and `max_input_noise_width` are the minimum and maximum input noise width, in library time units.
The following rules apply to \texttt{input\_noise\_width} range limits:

- The \texttt{min\_input\_noise\_width} and \texttt{max\_input\_noise\_width} attributes can be defined only on input or inout pins.
- The \texttt{min\_input\_noise\_width} and \texttt{max\_input\_noise\_width} attributes must both be defined.
- \texttt{min\_input\_noise\_width} \leq \texttt{max\_input\_noise\_width}

**Example**

```plaintext
pin (O) {
    direction : output ; /* existing syntax */
    capacitance : 1 ; /* existing syntax */
    fanout_load : 1 ; /* existing syntax */

    /* Noise range */
    min_input_noise_width : 0.0;
    max_input_noise_width : 2.0;

    /* Timing group defines what is acceptable noise on input pins. */
    timing () {
        /* Noise immunity.
        * Defines maximum allowed noise height for given pulse width.
        * Pulse height is absolute value from the signal level.
        * Any of the following four tables are optional. */

        noise_immunity_low (my\_noise\_reject) {
            values ("1.5, 0.9, 0.8, 0.65, 0.6");
        }
        noise_immunity_high (my\_noise\_reject) {
            values ("1.3, 0.8, 0.7, 0.6, 0.55");
        }
        noise_immunity_below_low (my\_noise\_reject\_outside\_rail) {
            values ("1, 0.8, 0.5");
        }
        noise_immunity_above_high (my\_noise\_reject\_outside\_rail) {
            values ("1, 0.8, 0.5");
        }
    } /* end of timing group */
} /* end of pin group */
```

**Defining the Hyperbolic Noise Groups**

To specify hyperbolic noise immunity information, use the \texttt{hyperbolic\_noise\_above\_high}, \texttt{hyperbolic\_noise\_below\_low}, \texttt{hyperbolic\_noise\_high}, \texttt{and} \texttt{hyperbolic\_noise\_low} groups within the \texttt{pin} group.
Syntax

pin(name_string) {
    ...
    hyperbolic_noise_above_high() {
        height_coefficient : float;
        area_coefficient : float;
        width_coefficient : float;
    }
    hyperbolic_noise_below_low() {
        ...
    }
    hyperbolic_noise_high() {
        ...
    }
    hyperbolic_noise_low() {
        ...
    }
    ...
}

float

The coefficient values for height, width, and area must be 0 or a positive number.

The following rules apply to noise immunity groups:

- The hyperbolic noise groups are optional, and each can be defined separately from the other three.

- For the same region (above-high, below-low, high, or low), the hyperbolic noise groups can coexist with normal noise immunity tables.

- For different regions (above-high, below-low, high, or low), a combination of tables and hyperbolic functions is allowed. For example, you might have a hyperbolic function for below and above the rails and have tables for high and low tables on the same pin.

- When no table or hyperbolic function is defined for a given pin, the application checks other measures for noise immunity, such as DC noise margins.

- The unit for height and height_coefficient is the library unit of voltage. The unit for width and width_coefficient is the library unit of time. The unit for area_coefficient is the library unit of voltage multiplied by the library unit of time.

Example

hyperbolic_noise_low() {
    height_coefficient : 0.4;
    area_coefficient : 1.1;
    width_coefficient : 0.1;
}

hyperbolic_noise_high() {
    height_coefficient : 0.3;
area_coefficient : 0.9;
width_coefficient : 0.1;
}

Representing Propagated Noise Information

You can represent propagated noise information at the timing level by using a

- Propagated Noise Lookup Table Model
- Propagated Noise Polynomial Model

Propagated Noise Lookup Table Model

You can represent propagated noise in your libraries by using lookup tables. To define your lookup tables, use the `propagation_lut_template` group in the `library` group. In the `timing` group, use the following groups:

- `propagated_noise_height_above_high`
- `propagated_noise_height_below_low`
- `propagated_noise_height_high`
- `propagated_noise_height_low`
- `propagated_noise_width_above_high`
- `propagated_noise_width_below_low`
- `propagated_noise_width_high`
- `propagated_noise_width_low`

`propagation_lut_template` Group

Use this library-level group to create templates of common information that multiple propagation lookup tables can use.

A table template specifies the propagated noise width, height, and output load and their corresponding breakpoints for the axis. Assign each template a name. Make the template name the group name of a propagated noise group.

**Syntax**

```plaintext
library(name_string) {
    ...
```
propagation_lut_template(template_namesstring) {
    variable_1: value;
    variable_2: value;
    variable_3: value;
    index_1 ("float,..., float");
    index_2 ("float,..., float");
    index_3 ("float,..., float");
}

Template Variables
The table template specifying propagated noise can have three variables (variable_1, variable_2, and variable_3). The variables indicate the parameters used to index the lookup table along the first, second, and third table axes. The parameters are input_noise_width, input_noise_height, and total_output_net_capacitance.

The index values in the index_1, index_2, and index_3 attributes are a list of positive floating-point numbers. The values in the list must be in increasing order.

The unit for input_noise_width and input_noise_height is the library time unit.

Example
propagation_lut_template(my_propagated_noise) {
    variable_1 : input_noise_width;
    variable_2 : input_noise_height;
    variable_3 : total_output_net_capacitance;
    index_1("0.01, 0.2, 2");
    index_2("0.2, 0.8");
    index_3("0, 2");
}

Defining the Propagated Noise Table Groups
To represent propagated noise, use the following groups within the timing group: propagated_noise_height_above_high, propagated_noise_height_below_low, propagated_noise_height_high, propagated_noise_height_low, and propagated_noise_width_above_high.
Syntax for Table Model

timing() {
...
  propagated_noise_height_above_high (temp_namestring) {
    index_1 ("float,..., float");
    index_2 ("float,..., float");
    index_3 ("float,..., float");
    values("float,..., float","..."float,..., float");
  }
  propagated_noise_height_below_low (temp_namestring) {
    ...
  }
  propagated_noise_width_above_high (temp_namestring) {
    ...
  }
  propagated_noise_width_below_low (temp_namestring) {
    ...
  }
  propagated_noise_height_high(temp_namestring) {
    ...
  }
  propagated_noise_height_low(temp_namestring) {
    ...
  }
  propagated_noise_width_high(temp_namestring) {
    ...
  }
  propagated_noise_width_low(temp_namestring) {
    ...
  }
...

Propagation Noise Group Rules

The following rules apply to the propagation noise groups:

- Each of the three pairs of tables is optional; the assumption is that if one pair is missing, the corresponding region does not propagate any noise.

- If a pair of tables for a particular region (high, low, above-high, or below-low) is specified, both width and height must be specified.

- Each propagated noise table has an associated name for the propagation_lut_template it uses. The name of the table must be identical to the name defined in a library propagated_noise_template group.
Each table can be two-dimensional or three-dimensional. The indexes are
input_noise_width, input_noise_height, and total_output_net_capacitance. The values are coefficients of height and width. The coefficient values for height and width must be 0 or a positive number.

You can overwrite any or all indexes in a propagated noise template. However, the overwrite must occur before the actual definition of the values.

The width and height values of the table are stored in the values attribute. Each height value is the absolute difference of the noise bump height voltage and the related rail voltage and is, therefore, a positive number. Any point over this curve describes a height/width combination that causes functional failure.

The unit for all propagated height is the library voltage unit. The unit for all propagated width is the library unit of time.

For points outside table ranges, extrapolation can be used.

Example

```plaintext
class propagated_noise_width_high {
  values ("0.01, 0.10, 0.15", "0.04, 0.14, 0.18","0.05, 0.15, 0.24","0.07, 0.17, 0.32");
}
class propagated_noise_height_high {
  values ("0.01, 0.20, 0.25", "0.04, 0.24, 0.28","0.05, 0.25, 0.28","0.07, 0.27, 0.35");
}
```

Propagated Noise Polynomial Model

As with the lookup table model, you can describe propagated noise in your libraries by using polynomial representation. To define your polynomial, use the poly_template group in the library group.

In the timing group, use the following groups:

- propagated_noise_height_above_high
- propagated_noise_height_below_low
- propagated_noise_height_high
- propagated_noise_height_low
- propagated_noise_width_above_high
- propagated_noise_width_below_low
- propagated_noise_width_high
You can define a poly_template group at the library level to specify the equation variables, the variable ranges, the voltage mapping, and the piecewise data. The valid values for the variables are extended to include input_noise_width, input_noise_height, input_peak_time_ratio, total_output_net_capacitance, temperature, and the related rail voltages.

**Syntax**

```plaintext
library(namestring) {
...
poly_template(template_namestring) {
  variables(variable_i_enum,..., variable_n_enum);
  variable_i_range: (float, float);
  ...
  variable_n_range: (float, float);
  mapping(voltage_enum, power_rail_id);
  domain(domain_namestring) {
    variable_i_range: (float, float);
    ...
    variable_n_range: (float, float);
  }
}
...
}
```

**Template Variables**

The syntax of the poly_template group is the same as that of the delay model, except that the variables used in the format are

- input_noise_width, input_noise_height, input_peak_time_ratio
- total_output_net_capacitance
- voltage, voltagei, temperature

The piecewise model through the domain group is also supported.
The `input_peak_time_ratio` is always specified as a ratio of width, so it is a value between 0.0 and 1.0.

Example

```plaintext
poly_template(my_propagated_noise) {
    variables (input_noise_width, input_noise_height, input_peak_time_ratio, total_output_net_capacitance);
    variable_1_range (0.01, 2);
    variable_2_range (0, 0.8);
    variable_3_range (0.0, 1.0);
    variable_4_range (0, 2);
} /* poly_template(propagated_noise) */
```

Defining Propagated Noise Groups for Polynomial Representation

To specify polynomial representation, use the `propagated_noise_height_above_high`, `propagated_noise_height_below_low`, `propagated_noise_height_high`, `propagated_noise_height_low`, `propagated_noise_width_above_high`, `propagated_noise_width_below_low`, `propagated_noise_width_high`, `propagated_noise_width_low`, `propagated_noise_peak_time_ratio_above_high`, `propagated_noise_peak_time_ratio_below_low`, `propagated_noise_peak_time_ratio_high`, and `propagated_noise_peak_time_ratio_low` groups within the timing group to define the polynomial.

The `peak_time_ratio` groups are supported only in the polynomial model.
Syntax for Polynomial

timing() {
...
propagated_noise_height_above_high (temp_nameString) {
    variable_i_range: (float, float);
    orders("integer, ..., integer");
    coefs("float, ..., float");
    domain(domain_nameString) {
        variable_i_range: (float, float);
        orders("integer, ..., integer");
        coefs("float, ..., float");
    }
...
}
propagated_noise_width_above_high (temp_nameString) {
...
}
propagated_noise_height_below_low (temp_nameString) {
...
}
propagated_noise_width_below_low (temp_nameString) {
...
}
propagated_noise_height_high(temp_nameString) {
...
}
propagated_noise_width_high(temp_nameString) {
...
}
propagated_noise_height_low(temp_nameString) {
...
}
propagated_noise_width_low(temp_nameString) {
...
}
propagated_noise_peak_time_ratio_above_high (temp_nameString) {
...
}
propagated_noise_peak_time_ratio_below_low(temp_nameString) {
...
}
propagated_noise_peak_time_ratio_high (temp_nameString) {
...
}
propagated_noise_peak_time_ratio_low (temp_nameString) {
...
}
...
}
Because the polynomial model is a superset of the lookup table model, all syntax supported in the lookup table is also supported in the polynomial model. For example, you can have a `propagated_noise_width_high` polynomial and a `propagated_noise_width_low` table defined in the same group in a scalable polynomial delay model library.

**Example**

```
propagated_noise_width_high(my_propagated_noise) {
    orders("1, 1, 1, 1");
    coefs("1, 2, 3, 4, ",
          1, 2, 3, 4, ",
          1, 2, 3, 4, ",
          1, 2, 3, 4");
}
propagated_noise_height_high(my_propagated_noise) {
    orders("1, 1, 1, 1");
    coefs("1, 2, 3, 4, ",
          1, 2, 3, 4, ",
          1, 2, 3, 4, ",
          1, 2, 3, 4");
}
```

### Examples of Modeling Noise

The examples in this section model libraries for noise extension for scalable polynomials and nonlinear lookup table model libraries.

#### Scalable Polynomial Model Noise Example

A scalable polynomial delay library allows you to describe how noise parameters vary with rail voltage and temperature.

```
library (my_noise_lib) {
    delay_model : "polynomial";
    time_unit : "1ns";
    voltage_unit : "1V";
    current_unit : "1mA";
    capacitive_load_unit (1, pf);
    pulling_resistance_unit : 1kohm;
    power_supply() {
        default_power_rail : VDD1;
        power_rail(VDD1, 1.6);
        power_rail(VDD2, 1.3);
    }
    nom_voltage : 1.0;
    nom_temperature : 40;
    nom_process : 1.0;
    /* Templates of DC noise margins and output levels */
    input_voltage(MY_CMOS_IN) {
        vil : 0.3;
    }
```
vih : 1.1;
vinmin : -0.3;
vimax : VDD + 0.3;
}

output_voltage(MY_CMOS_OUT) {
  vol : 0.1;
  voh : 1.4;
  vinmin : -0.3;
  vomax : VDD + 0.3;
}

/* Template definitions for noise immunity. Variable: * input_noise_width */
poly_template ( my_noise_reject ) {
  temperature, total_output_net_capacitance);
  variables (input_noise_width, voltage, voltage1, \
      temperature, total_output_net_capacitance);
  mapping(voltage, VDD1);
  mapping(voltage1, VDD2);
  variable_1_range (0, 2);
  variable_2_range (1.4, 1.8);
  variable_3_range (1.1, 1.5);
  variable_4_range (-40, 125);
  variable_5_range (0.0, 1.0);
  domain (typ) {
    variables (input_noise_width, voltage, voltage1, \
      temperature, total_output_net_capacitance);
    variable_1_range (0, 2);
    variable_2_range (1.5, 1.7);
    variable_3_range (1.2, 1.4);
    variable_4_range (25, 25);
    variable_5_range (0.0, 1.0);
    mapping(voltage, VDD1);
    mapping(voltage1, VDD2);
  }
  domain (min) {
    variables (input_noise_width, voltage, voltage1, \
      temperature);
    variable_1_range (0, 2);
    variable_2_range (1.7, 1.8);
    variable_3_range (1.4, 1.5);
    variable_4_range (-40, -40);
    mapping(voltage, VDD1);
    mapping(voltage1, VDD2);
  }
  domain (max) {
    variables (input_noise_width, voltage, voltage1, \
      temperature);
    variable_1_range (0, 2);
    variable_2_range (1.6, 1.7);
    variable_3_range (1.1, 1.2);
    variable_4_range (125, 125);
    mapping(voltage, VDD1);
    mapping(voltage1, VDD2);
  }
}

/* end poly_template (my_noise_reject) */
poly_template ( my_noise_reject_outside_rail ) {
  variables (input_noise_width, voltage, voltage1, \
      temperature);
mapping(voltage, VDD1);
mapping(voltage1, VDD2);
variable_1_range (0, 2);
variable_2_range (1.4, 1.8);
variable_3_range (1.1, 1.5);
variable_4_range (-40, 125);
} /* end poly_template ( my_noise_reject_outside_rail ) */
/* Template definitions for I-V characteristics. Variable: *
  * iv_output_voltage */
poly_template ( my_current_low ) {
  variables ( iv_output_voltage, voltage, voltage1, 
           temperature );
mapping(voltage, VDD1);
mapping(voltage1, VDD2);
variable_1_range (-1, 2);
variable_2_range (1.4, 1.8);
variable_3_range (1.1, 1.5);
variable_4_range (-40, 125);
} /* end poly_template ( my_current_low ) */
poly_template ( my_current_high ) {
  variables ( iv_output_voltage, voltage, voltage1, 
           temperature );
mapping(voltage, VDD1);
mapping(voltage1, VDD2);
variable_1_range (-1, 2);
variable_2_range (1.4, 1.8);
variable_3_range (1.1, 1.5);
variable_4_range (-40, 125);
} /* end poly_template ( my_current_high ) */
/* Template definitions for propagated noise. Variables:
  * input_noise_width
  * input_noise_height
  * input_peak_time_ratio
  * total_output_net_capacitance */
poly_template(my_propagated_noise) {
  variables ( input_noise_width, input_noise_height, 
            input_peak_time_ratio, 
            total_output_net_capacitance, voltage, 
            voltage1, temperature );
mapping(voltage, VDD1);
mapping(voltage1, VDD2);
variable_1_range (0.01, 2);
variable_2_range (0, 0.8);
variable_3_range (0.0, 1.0);
variable_4_range (0, 2);
variable_5_range (1.4, 1.8);
variable_6_range (1.1, 1.5);
variable_7_range (-40, 125);
} /* end poly_template (my_propagated_noise) */
/* INVERTER */
cell ( INV ) {
  area : 1 ;
  pin { A } {
    direction : input ;
capacitance : 1 ;
fanout_load : 1 ;
  } /* DC noise margins.
  * These are used for compatibility of level shifters.
* In noise analysis, they are the least accurate way
* to define noise margins.
* Compatible: can coexist in the pin group with any
* other noise margin definition. */

```c
input_voltage : MY_CMOS_IN ;
/* Noise group defines what is acceptable noise on input
* pins. */
/* Hyperbolic noise immunity.
* Another way to specify noise immunity.
* Mutually exclusive: noise_immunity_low cannot be
* together with
*hyperbolic_noise_immunity_low, and so on.
* Defines pulse_height = height_coefficient +
* area_coefficient / (width - width_coefficient)
* Characterization recommendation: Use
* hyperbolic_noise_immunity_*
* if it can fit the curve, otherwise use
* table noise_immunity_* */
hyperbolic_noise_low() {
    height_coefficient : 0.4;
    area_coefficient : 1.1;
    width_coefficient : 0.1;
}
hyperbolic_noise_high() {
    height_coefficient : 0.3;
    area_coefficient : 0.9;
    width_coefficient : 0.1;
}
hyperbolic_noise_below_low() {
    height_coefficient : 0.1;
    area_coefficient : 0.3;
    width_coefficient : 0.01;
}
hyperbolic_noise_above_high() {
    height_coefficient : 0.1;
    area_coefficient : 0.3;
    width_coefficient : 0.01;
}
}

/* end pin (A) */

pin ( Y ) {
    direction : output ;
    max_fanout : 10 ;
    function : "!A ";
    output_voltage : MY_CMOS_OUT ;
    timing () {
        related_pin : A ;
        /* Steady state drive resistance */
        steady_state_resistance_high : 1500;
        steady_state_resistance_low : 1100;
        steady_state_resistance_above_high : 200;
        steady_state_resistance_below_low : 100;
        /* I-V curve.
* Describes how much current the pin can deliver in a given state for
* a given voltage on the pin.
* Voltage is measured from the pin to ground, current is measured
* flowing into the cell (both can be either positive or negative). */
        steady_state_current_low(my_current_low) {
            orders ("3, 3, 0, 0");
```

Chapter 12: Nonlinear Signal Integrity Modeling
Examples of Modeling Noise 12-35
coefs (*8.4165, 0.3198, -0.0004, 0.0000, \ 1133.8274, 8.7287, -0.0054, 0.0000, \ 139.8645, -60.3898, 0.0589, -0.0000, \ -167.4473, 95.7112, -0.1018, 0.0000*);
}

steady_state_current_high(my_current_high) {
  orders (*3, 3, 0, 0*);
  coefs (*10.9165, 0.2198, -0.0003, 0.0000, \ 1433.8274, 8.7287, -0.0054, 0.0000, \ 128.8645, -60.3898, 0.0589, -0.0000, \ -167.4473, 95.7112, -0.1018, 0.0000*);
}

/* Noise immunity.
* Defines maximum allowed noise height for given pulse width.
* Pulse height is absolute value from the signal level.
* Any of the 4 tables below are optional. */
noise_immunity_low (my_noise_reject) {
  domain (typ) {
    orders (*3, 3, 0, 0, 0*);
    coefs (*11.4165, 0.2198, -0.0003, 0.0000, \ 1353.8274, 8.7287, -0.0054, 0.0000, \ 149.8645, -60.3898, 0.0589, -0.0000, \ -167.4473, 95.7112, -0.1018, 0.0000*);
  }
  domain (min) {
    orders (*3, 3, 0, 0*);
    coefs (*6.964065, 0.134078, -0.000183, 0.0000, \ 825.834714, 5.324507, -0.003294, 0.0000, \ 91.417345, -36.837778, .035929, -0.0000, \ -102.142853, 58.383832, -.062098, 0.0000*);
  }
  domain (max) {
    orders (*3, 3, 0, 0*);
    coefs (*19.065555, 0.367066, -0.000501, 0.0000, \ 2260.891758, 14.576929, -0.009018, 0.0000, \ 250.273715, -100.850966, 0.098363, -0.0000, \ -279.636991, 159.837704, -0.170006, 0.0000*);
  }
}

noise_immunity_high (my_noise_reject) {
  domain (typ) {
    orders (*3, 3, 0, 0*);
    coefs (*12.4165, 0.2198, -0.0003, 0.0000, \ 1353.8274, 8.7287, -0.0054, 0.0000, \ 129.8645, -60.3898, 0.0589, -0.0000, \ -147.4473, 95.7112, -0.1018, 0.0000*);
  }
  domain (min) {
    orders (*3, 3, 0, 0*);
    coefs (*6.364065, 0.134078, -0.000183, 0.0000, \ 845.834714, 5.324507, -0.003294, 0.0000, \ 91.417345, -36.837778, .035929, -0.0000, \ -103.142853, 58.383832, -.062098, 0.0000*);
  }
  domain (max) {
    orders (*3, 3, 0, 0*);
    coefs (*19.265555, 0.367066, -0.000601, 0.0000, \ 2460.891758, 14.576929, -0.009018, 0.0000, \
Examples of Modeling Noise 12-37

Chapter 12: Nonlinear Signal Integrity Modeling

Noise Immunity:

```c
noise_immunity_below_low (my_noise_reject_outside_rail) {
    orders (*3, 3, 0, 0*);
    coefs (*10.4165, 0.1198, -0.0003, 0.0000, \n         1333.8274, 8.7287, -0.0054, 0.0000, \n         149.8645, -60.3898, 0.0589, -0.0000, \n         -167.4473, 95.7112, -0.1018, 0.0000*);
}

noise_immunity_above_high (my_noise_reject_outside_rail) {
    orders (*3, 3, 0, 0*);
    coefs (*12.4165, 0.2298, -0.0003, 0.0000, \n         1253.8274, 8.7287, -0.0054, 0.0000, \n         149.8645, -60.3898, 0.0589, -0.0000, \n         -167.4473, 95.7112, -0.1018, 0.0000*);
}
```

/* Propagated noise. */

* It is a function of input noise width and height and output capacitance. Width and height are in separate tables. */

```c
propagated_noise_width_high(my_propagated_noise) {
    orders (*1, 2, 1, 0, 0, 0, 0*);
    coefs (*8.4165, 0.3198, -0.0004, 0.2000, \n          1.8645, -6.3898, 0.0589, -0.0300, \n          -1.4473, 9.7112, -0.1018, 0.3500, *);
}

propagated_noise_height_high(my_propagated_noise) {
    orders (*1, 2, 1, 0, 0, 0, 0*);
    coefs (*0.4165, 0.3198, -0.0014, 0.2000, \n          0.8645, -6.3898, 0.0589, -0.0300, \n          -0.4473, 0.7112, -0.1018, 0.3500, *);
}

propagated_noise_peak_time_ratio_high(my_propagated_noise) {
    orders (*1, 2, 1, 0, 0, 0, 0*);
    coefs (*0.4165, 0.3198, 0.0014, 0.2000, \n          0.8645, 0.3898, 0.0589, 0.0300, \n          0.4473, 0.7112, 0.1018, 0.3500, *);
}

propagated_noise_width_low(my_propagated_noise) {
    orders (*1, 2, 1, 0, 0, 0, 0*);
    coefs (*8.4165, 0.3198, -0.0004, 0.2000, \n          1.8645, -6.3898, 0.0589, -0.0300, \n          -1.4473, 9.7112, -0.1018, 0.3500, *);
}

propagated_noise_height_low(my_propagated_noise) {
    orders (*1, 2, 1, 0, 0, 0, 0*);
    coefs (*0.4165, 0.3198, -0.0014, 0.2000, \n          0.8645, -6.3898, 0.0589, -0.0300, \n          -0.4473, 0.7112, -0.1018, 0.3500, *);
}

propagated_noise_peak_time_ratio_low(my_propagated_noise) {
    orders (*1, 2, 1, 0, 0, 0, 0*);
    coefs (*0.4165, 0.3198, 0.0014, 0.2000, \n          0.8645, 0.3898, 0.0589, 0.0300, \n          0.4473, 0.7112, 0.1018, 0.3500, *);
}

propagated_noise_width_above_high(my_propagated_noise) {
```
orders ("1, 2, 1, 0, 0, 0, 0*");
coefs ("8.4165, 0.3198, 0.0004, 0.2000, \
1.8645, -6.3898, 0.0589, -0.0300, \
-1.4473, 9.7112, -0.1018, 0.3500, *");
}

propagated_noise_height_above_high(my_propagated_noise) {
    orders ("1, 2, 1, 0, 0, 0, 0*");
    coefs ("0.4165, 0.3198, 0.0014, 0.2000, \
0.8645, -6.3898, 0.0589, 0.03000, \
-0.4473, 0.7112, 0.1018, 0.3500, *");
}

propagated_noise_peak_time_ratio_above_high(my_propagated_noise) {
    orders ("1, 2, 1, 0, 0, 0, 0*");
    coefs ("0.4165, 0.3198, 0.0014, 0.2000, \
0.8645, 0.3898, 0.0589, 0.03000, \
0.4473, 0.7112, 0.1018, 0.3500, *");
}

propagated_noise_width_below_low(my_propagated_noise) {
    orders ("1, 2, 1, 0, 0, 0, 0*");
    coefs ("8.4165, 0.3198, 0.0004, 0.2000, \
1.8645, -6.3898, 0.0589, -0.0300, \
-1.4473, 9.7112, -0.1018, 0.3500, *");
}

propagated_noise_height_below_low(my_propagated_noise) {
    orders ("1, 2, 1, 0, 0, 0, 0*");
    coefs ("0.4165, 0.3198, 0.0014, 0.2000, \
0.8645, -6.3898, 0.0589, -0.0300, \
-0.4473, 0.7112, -0.1018, 0.3500, *");
}

propagated_noise_peak_time_ratio_below_low(my_propagated_noise) {
    orders ("1, 2, 1, 0, 0, 0, 0*");
    coefs ("0.4165, 0.3198, 0.0014, 0.2000, \
0.8645, 0.3898, 0.0589, 0.03000, \
0.4473, 0.7112, 0.1018, 0.3500, *");
}

cell_rise(scalar) { values("0");}
rise_transition(scalar) { values("0");}
cell_fall(scalar) { values("0");}
fall_transition(scalar) { values("0");}
} /* end of timing group */
} /* end of pin (Y) */
} /* end of cell (INV) */
} /* end of library (my_noise_lib)
Nonlinear Delay Model Library With Noise Information

A nonlinear delay model noise library is limited to a fixed voltage.

library (my_noise_lib) {
    delay_model : "table_lookup";
    time_unit : "1ns";
    voltage_unit : "1V";
    current_unit : "1mA";
    capacitive_load_unit (1, pf);
    pulling_resistance_unit : 1kohm;
    nom_voltage : 1.6;
    nom_temperature : 40.0;
    nom_process : 1.0;
    /* Templates of input and output levels (used for DC noise margin) */
    input_voltage(MY_CMOS_IN) {
        vil : 0.3;
        vih : 1.1;
        vimin : -0.3;
        vimax : VDD + 0.3;
    }
    output_voltage(MY_CMOS_OUT) {
        vol : 0.1;
        voh : 1.4;
        vomin : -0.3;
        vomax : VDD + 0.3;
    }
    /* Template definitions for noise immunity. Variable:
     * input_noise_width */
    noise_lut_template(my_noise_reject) {
        variable_1 : input_noise_width;
        variable_2 : total_output_net_capacitance;
        index_1("0, 0.1, 0.3, 1, 2");
        index_2("0, 0.1, 0.3, 1, 2");
    }
    noise_lut_template(my_noise_reject_outside_rail) {
        variable_1 : input_noise_width;
        variable_2 : total_output_net_capacitance;
        index_1("0, 0.1, 2");
        index_2("0, 0.1, 2");
    }
    /* Template definitions for I-V characteristics. Variable:
     * iv_output_voltage */
    iv_lut_template(my_current_low) {
        variable_1 : iv_output_voltage
        index_1("-1, -0.1, 0, 0.1, 0.8, 1.6, 2");
    }
    iv_lut_template(my_current_high) {
        variable_1 : iv_output_voltage
        index_1("-1, 0, 0.3, 0.5, 0.8, 1.5, 1.6, 1.7, 2");
    }
    /* Template definitions for propagated noise. Variables:
     * input_noise_width
     * input_noise_height
     * total_output_net_capacitance */
    propagation_lut_template(my_propagated_noise) {
        variable_1 : input_noise_width;
variable_2 : input_noise_height;
variable_3 : total_output_net_capacitance;
index_1("0.01, 0.2, 2");
index_2("0.2, 0.8");
index_3("0, 2");
}
cell (tieoff_30_esd) {
  pin (high) {
    direction : output;
capacitance : 0;
function : "1";
/* noise information */
timing() {
  tied_off : true;
  steady_state_resistance_high : 1.22;
  steady_state_resistance_above_high : 1.00;
  steady_state_current_high(ivl1x5)(
    index_1("0.3,0.75,1.0,1.2,2");
    values("-513.2,-447.9,-359.3,-245.7,497.3");
  )
}
}
pin (low) {
  direction : output;
capacitance : 0;
function : "0";
/* noise information */
timing() {
  tied_off : true;
  steady_state_resistance_low : 0.1;
  steady_state_resistance_below_low : 0.4;
  steady_state_current_low(ivl1x5)(
    index_1("-0.25,0.3,0.5,1.0,1.8");
    values("-595.4,555.4,690.5,774.75,822.5");
  )
}
}
/* INVERTER */
cell (INV) {
  area : 1;
  pin (A) {
    direction : input;
capacitance : 1;
fanout_load : 1;
/* DC noise margins.
* These are used for compatibility of level shifters. In noise
* analysis they are the least accurate way to define noise margins.
* Compatible: can coexist in the pin group with any other noise margin
* definition. */
input_voltage : MY_CMOS_IN;
/* Timing group defines what is acceptable noise on input pins. */
/* Hyperbolic noise immunity.
* Another way to specify noise immunity.
* Mutually exclusive: noise_immunity_low cannot be together with
* hyperbolic_noise_immunity_low, etc.
* Defines pulse_height = height_coefficient +
* area_coefficient / (width - width_coefficient)
* Characterization recommendation: use hyperbolic_noise_immunity_*
* if can fit the curve, otherwise table noise_immunity_* */
hyperbolic_noise_low() {
    height_coefficient : 0.4;
    area_coefficient : 1.1;
    width_coefficient : 0.1;
}
hyperbolic_noise_high() {
    height_coefficient : 0.3;
    area_coefficient : 0.9;
    width_coefficient : 0.1;
}
hyperbolic_noise_below_low() {
    height_coefficient : 0.1;
    area_coefficient : 0.3;
    width_coefficient : 0.01;
}
hyperbolic_noise_above_high() {
    height_coefficient : 0.1;
    area_coefficient : 0.3;
    width_coefficient : 0.01;
}
/* end of pin A */
pin ( Y ) {
    direction : output;
    max_fanout : 10;
    function : "!A ";
    output_voltage : MY_CMOS_OUT;
    min_input_noise_width : 0.0;
    max_input_noise_width : 2.0;
    timing () {
        related_pin : A;
        /* Steady-state drive resistance */
        steady_state_resistance_high : 1500;
        steady_state_resistance_low : 1100;
        steady_state_resistance_above_high : 200;
        steady_state_resistance_below_low : 100;
        /* I-V curve. */
        * Describes how much current the pin can deliver in a given state for
        * a given voltage on the pin. The steady_state_resistance* max is the
        * highest resistance in the I-V curve, the
        * steady_state_resistance* min is the lowest.
        * Mutually exclusive: If steady_state_resistance_low* or
        * steady_state_resistance_max or steady_state_resistance_min is
        * specified, the I-V curve cannot be specified.
        * Characterization recommendation: Use steady_state_resistance* if
        * an I-V curve cannot be generated.
        * Voltage is measured from the pin to ground, current measured
        * flowing into the cell (both can be either positive or negative). */
        steady_state_current_low(my_current_low) {
            values("0.1, 0.05, 0, -0.1, -0.25, -1, -1.8");
        }
        steady_state_current_high(my_current_high) {
            values("2, 1.8, 1.7, 1.4, 1, 0.5, 0, -0.1, -0.8");
        }
        cell_rise(scalar) { values("0");}
        rise_transition(scalar) { values("0");}
cell_fall(scalar) { values("0");}
fall_transition(scalar) { values("0");}
/* Noise immunity.
* Defines the maximum allowed noise height for given pulse width.
* Pulse height is the absolute value from the signal level.
* Any of the following four tables are optional. */
noise_immunity_low (my_noise_reject) {
    values ("1.5, 0.9, 0.8, 0.65, 0.6", 
            "1.5, 0.9, 0.8, 0.65, 0.6", 
            "1.5, 0.9, 0.8, 0.65, 0.6", 
            "1.5, 0.9, 0.8, 0.65, 0.6");
}
noise_immunity_high (my_noise_reject) {
    values ("1.3, 0.8, 0.7, 0.6, 0.55", 
            "1.5, 0.9, 0.8, 0.65, 0.6", 
            "1.5, 0.9, 0.8, 0.65, 0.6", 
            "1.5, 0.9, 0.8, 0.65, 0.6");
}
noise_immunity_below_low (my_noise_reject_outside_rail) {
    values ("1, 0.8, 0.5", 
            "1, 0.8, 0.5");
}
noise_immunity_above_high (my_noise_reject_outside_rail) {
    values ("1, 0.8, 0.5", 
            "1, 0.8, 0.5");
}
/* Propagated noise.
* A function of input noise width, height, and output
* capacitance. Width and height are in separate tables. */
propagated_noise_width_high(my_propagated_noise) {
    values ("0.01, 0.10", "0.15, 0.04", "0.14, 0.18", 
            "0.05, 0.15", "0.24, 0.07", "0.17, 0.32");
}
propagated_noise_height_high(my_propagated_noise) {
    values ("0.01, 0.10", "0.15, 0.04", "0.14, 0.18", 
            "0.05, 0.15", "0.24, 0.07", "0.17, 0.32");
}
propagated_noise_width_low(my_propagated_noise) {
    values ("0.01, 0.10", "0.15, 0.04", "0.14, 0.18", 
            "0.05, 0.15", "0.24, 0.07", "0.17, 0.32");
}
propagated_noise_height_low(my_propagated_noise) {
    values ("0.01, 0.10", "0.15, 0.04", "0.14, 0.18", 
            "0.05, 0.15", "0.24, 0.07", "0.17, 0.32");
}
propagated_noise_width_above_high(my_propagated_noise) {
    values ("0.01, 0.10", "0.15, 0.04", "0.14, 0.18", 
            "0.05, 0.15", "0.24, 0.07", "0.17, 0.32");
}
propagated_noise_height_above_high(my_propagated_noise) {
    values ("0.01, 0.10", "0.15, 0.04", "0.14, 0.18", 
            "0.05, 0.15", "0.24, 0.07", "0.17, 0.32");
}
propagated_noise_width_below_low(my_propagated_noise) {
values ("0.01, 0.10", "0.15, 0.04", "0.14, 0.18", \n*0.05, 0.15", "0.24, 0.07", "0.17, 0.32");
}
propagated_noise_height_below_low(my_propagated_noise) {
    values ("0.01, 0.10", "0.15, 0.04", "0.14, 0.18", \n*0.05, 0.15", "0.24, 0.07", "0.17, 0.32");
} /*end propagated noise groups */
} /* end of timing group */
} /* end of pin (Y){ */
} /* end of cell (INV) */
} /* end of library (my_noise_lib)
Composite Current Source Signal Integrity Modeling

This chapter provides an overview of composite current source (CCS) modeling to support noise (signal integrity) modeling for advanced technologies. This chapter includes the following sections:

- CCS Signal Integrity Modeling Overview
- CCS Noise Modeling for Unbuffered Cells With a Pass Gate
- CCS Noise Modeling for Multivoltage Designs
- Referenced CCS Noise Modeling
CCS Signal Integrity Modeling Overview

CCS noise modeling can capture essential noise properties of digital circuits using a compact library representation. It enables fast and accurate gate-level noise analysis while maintaining a relatively simple library characterization. CCS noise modeling supports noise combination and driver weakening.

CCS noise is characterization data that provides information for noise failure detection on cell inputs, calculation of noise bumps on cell outputs, and noise propagation through the cell. For the best accuracy, you must add CCS timing data to the library in addition to the CCS noise data. The CCS noise data includes the following:

- Channel-connected block parameters
- DC current tables
- Timing tables for rising and falling transitions
- Timing tables for low and high propagated noise

CCS Signal Integrity Modeling Syntax

```
library (name) {
  ...
  lu_table_template(dc_template_name) {
    variable_1 : input_voltage;
    variable_2 : output Voltage;
  }
  lu_table_template(output_voltage_template_name) {
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    variable_3 : time;
  }
  lu_table_template(propagated_noise_template_name) {
    variable_1 : input_noise_height;
    variable_2 : input_noise_width;
    variable_3 : total_output_net_capacitance;
    variable_4 : time;
  }
  cell (name) {
    pin (name) {
      ccsn_first_stage () {
        is_needed : true | false;
        is_inverting : Boolean;
        stage_type : stage_type_value;
        miller_cap_rise : float;
        miller_cap_fall : float;
        output_signal_level : power_supply_name;
      }
    }
  }
```
dc_current (dc_current_template) {
    index_1("float, ...")
    index_2("float, ...")
    values("float, ...")
}
output_voltage_rise () {
    vector (output_voltage_template_name) {
        index_1(float)
        index_2(float)
        index_3("float, ...")
        values("float, ...")
    }
    ...
}
output_voltage_fall () {
    vector (output_voltage_template_name) {
        index_1(float)
        index_2(float)
        index_3("float, ...")
        values("float, ...")
    }
    ...
}
propagated_noise_low () {
    vector (propagated_noise_template_name) {
        index_1(float)
        index_2(float)
        index_3(float)
        index_4("float, ...")
        values("float, ...")
    }
    ...
}
propagated_noise_high () {
    vector (propagated_noise_template_name) {
        index_1(float)
        index_2(float)
        index_3(float)
        index_4("float, ...")
        values("float, ...")
    }
    ...
}
when : "Boolean expression"
} /* ccsn_first_stage */
ccsn_last_stage () {
    is_needed : true | false;
    is_inverting : Boolean;
    stage_type : stage_type_value;
    miller_cap_rise : float;
    miller_cap_fall : float;
    input_signal_level : power_supply_name;
dc_current (dc_current_template) {
  index_1("float, ...");
  index_2("float, ...");
  values("float, ...");
}
output_voltage_rise ( ) {
  vector (output_voltage_template_name) {
    index_1(float);
    index_2(float);
    index_3("float, ...");
    values("float, ...");
  }
  ...
}
output_voltage_fall ( ) {
  vector (output_voltage_template_name) {
    index_1(float);
    index_2(float);
    index_3("float, ...");
    values("float, ...");
  }
  ...
}
propagated_noise_low ( ) {
  vector (propagated_noise_template_name) {
    index_1(float);
    index_2(float);
    index_3(float);
    index_4("float, ...");
    values("float, ...");
  }
  ...
}
propagated_noise_high ( ) {
  vector (propagated_noise_template_name) {
    index_1(float);
    index_2(float);
    index_3(float);
    index_4("float, ...");
    values("float, ...");
  }
  ...
}
when : "Boolean expression";
} /* ccsn_last_stage */

/* ccsn_first_stage */
...timing() {
  ...
  ccsn_first_stage () {
    is_needed : true | false;
    is_inverting : Boolean;
    stage_type : stage_type_value;
    miller_cap_rise : float;
miller_cap_fall : float;
output_signal_level : power_supply_name;
dc_current (dc_current_template) {
    index_1("float, ...");
    index_2("float, ...");
    values("float, ...");
}

output_voltage_rise ( ) {
    vector (output_voltage_template_name) {
        index_1(float);
        index_2(float);
        index_3("float, ...");
        values("float, ...");
    }
    ...
}

output_voltage_fall ( ) {
    vector (output_voltage_template_name) {
        index_1(float);
        index_2(float);
        index_3("float, ...");
        values("float, ...");
    }
    ...
}

propagated_noise_low ( ) {
    vector (propagated_noise_template_name) {
        index_1(float);
        index_2(float);
        index_3(float);
        index_4("float, ...");
        values("float, ...");
    }
    ...
}

propagated_noise_high ( ) {
    vector (propagated_noise_template_name) {
        index_1(float);
        index_2(float);
        index_3(float);
        index_4("float, ...");
        values("float, ...");
    }
    ...
}

when : "Boolean expression"
} /* ccsn_first_stage */

ccsn_last_stage () { 
is_needed : true | false;
is_inverting : Boolean;
stage_type : stage_type_value;
miller_cap_rise : float;
miller_cap_fall : float;
input_signal_level : power_supply_name;
dc_current (dc_current_template)
  index_1("float, ...");
  index_2("float, ...");
  values("float, ...");
}

output_voltage_rise ( )
  vector (output_voltage_template_name) {
    index_1(float);
    index_2(float);
    index_3("float, ...");
    values("float, ...");
  }
...

output_voltage_fall ( ) {
  vector (output_voltage_template_name) {
    index_1(float);
    index_2(float);
    index_3("float, ...");
    values("float, ...");
  }
...

propagated_noise_low ( ) {
  vector (propagated_noise_template_name) {
    index_1(float);
    index_2(float);
    index_3(float);
    index_4("float, ...");
    values("float, ...");
  }
...

propagated_noise_high ( ) {
  vector (propagated_noise_template_name) {
    index_1(float);
    index_2(float);
    index_3(float);
    index_4("float, ...");
    values("float, ...");
  }
...

  when : "Boolean expression";
} /* ccsn_last_stage */
} /* end timing/pin/cell/library group */
Library-Level Groups and Attributes
This section describes the library-level groups and attributes used for CCS noise modeling.

lu_table_template Group
The lu_table_template group creates the lookup-table template for the dc_current group and vectors for the output_voltage_rise, output_voltage_fall, propagated_noise_high, and propagated_noise_low groups.

variable_1, variable_2, variable_3, and variable_4 Attributes
Set the variable_1, variable_2, variable_3, and variable_4 attributes inside the lu_table_template group.

You can specify the template used for the following tables and vectors by using a combination of these attributes:

- The output_current_rise and output_current_fall group vectors
  Valid values for variable_1, variable_2, and variable_3 are input_net_transition, total_output_net_capacitance, and time, respectively.

- The propagated_noise_low and propagated_noise_high group vectors
  Valid values for variable_1, variable_2, variable_3, and variable_4 are input_noise_height, input_noise_width, total_output_net_capacitance, and time, respectively.

- The template used for the dc_current tables
  Valid values for variable_1 and variable_2 are input_voltage and output_voltage, respectively.

Pin-Level Groups and Attributes
This section describes the pin-level groups and attributes used for CCS noise modeling.

ccsn_first_stage and ccsn_last_stage Groups
The ccsn_first_stage and ccsn_last_stage groups specify CCS noise data for the first stage or the last stage of channel-connected blocks. The ccsn_first_stage and ccsn_last_stage groups can be defined inside timing or pin groups.

The ccsn_first_stage and ccsn_last_stage groups contain the following:
• The is_needed, is_inverting, stage_type, miller_cap_rise and miller_cap_fall, and output_signal_level or input_signal_level channel-connected block attributes

• The dc_current group, which contains a two-dimensional DC current table

• The output_current_rise and output_current_fall groups, which contain two timing tables for rising and falling transitions.

• The propagated_noise_low and propagated_noise_high groups, which contain two noise tables for low and high propagated noise.

Note:
If the ccsn_first_stage and ccsn_last_stage groups are defined at the pin level, the ccsn_first_stage group can be defined only in an input pin or inout pin, and the ccsn_last_stage group can be defined only in an output pin or inout pin.

is_needed Attribute

The is_needed Boolean attribute determines whether the dc_current, output_current_rise, output_current_fall, propagated_noise_low, and propagated_noise_high channel-connected block attributes should be specified to include CCS noise data for a cell. The is_needed attribute is defined inside the ccsn_first_stage and ccsn_last_stage groups.

By default, the is_needed attribute is set to true, which means that CCS noise data is included in the ccsn_first_stage and ccsn_last_stage groups for the cell. The is_needed attribute should be set to false for cells that do not need a current-based driver model, such as diodes, antennas, and cload cells. When the attribute is set to false, CCS noise data, enabled by the channel-connected block attributes, is not included in the ccsn_first_stage and ccsn_last_stage groups.

is_inverting Attribute

The is_inverting attribute specifies whether the channel-connected block is inverting. If the channel-connected block is inverting, set the is_inverting attribute to true. Otherwise, set the attribute to false. This attribute is mandatory if the is_needed attribute is set to true. Note that the is_inverting attribute is different from the "invertness" or timing_sense of the timing arc, which might consist of multiple channel-connected blocks.
stage_type Attribute

The stage_type attribute specifies the channel-connected block’s output voltage stage type. The valid values are pull_up, which causes the channel-connected block’s output voltage to be pulled up or to rise; pull_down, which causes the channel-connected block’s output voltage to be pulled down or to fall; and both, which causes the channel-connected block’s output voltage to be pulled up or down.

miller_cap_rise and miller_cap_fall Attributes

The miller_cap_rise and miller_cap_fall float attributes specify the Miller capacitance value for a rising and falling channel-connected block output transition. The value must be greater than or equal to zero. The attributes are defined inside the ccsn_first_stage and ccsn_last_stage groups.

output_signal_level and input_signal_level Attributes

The output_signal_level and input_signal_level attributes specify the power supply voltage names for a channel-connected block output and input, respectively. The output_signal_level attribute is defined inside the ccsn_first_stage group and the input_signal_level attribute is defined inside the ccsn_last_stage group.

Note:

- The output_signal_level and input_signal_level attribute specifications within the ccsn_first_stage and ccsn_last_stage groups override the output_signal_level and input_signal_level attribute specifications at the pin level.
- For a timing arc, the output_signal_level attribute specification within the ccsn_first_stage group overrides the output_signal_level attribute specification for the related pin (defined by the related_pin attribute).

dc_current Group

The dc_current group specifies the input and output voltage values of a two-dimensional current table for a channel-connected block. Use the index_1 and index_2 attributes, respectively, to list the input and output voltage values in library voltage units. Specify the values attribute in the dc_current group to list the relative channel-connected block DC current values, in library current units, that are measured at the channel-connected block output node.
output_voltage_rise and output_voltage_fall Groups

The output_voltage_rise and output_voltage_fall groups specify vector groups that describe three-dimensional output_voltage tables for a channel-connected block whose output node’s voltage values are rising or falling. The groups are defined inside the ccsn_first_stage and ccsn_last_stage groups.

Specify the following attributes in the vector group: The index_1 attribute lists the input_net_transition (slew) values in library time units. The index_2 attribute lists the total_output_net_capacitance (load) values in library capacitance units. The index_3 attribute lists the sampling time values in library time units. The values attribute lists the voltage values, in library voltage units, that are measured at the channel-connected block output node.

propagated_noise_high and propagated_noise_low Groups

The propagated_noise_low and propagated_noise_high groups use vector groups to specify the three-dimensional output_voltage tables of the channel-connected block whose output node’s voltage values are rising or falling. The groups are defined inside the ccsn_first_stage and ccsn_last_stage groups.

Specify the following attributes in the vector group: The index_1 attribute lists the input_noise_height values in library voltage units. The index_2 attribute lists the input_noise_width values in library time units. The index_3 attribute lists the total_output_net_capacitance values in library capacitance units. The index_4 attribute lists the sampling time values in library time units. The values attribute lists the voltage values, in library voltage units, that are measured at the channel-connected block output node.

when Attribute

The when attribute specifies the condition under which the channel-connected block data is applied. The attribute is defined in the ccsn_first_stage and ccsn_last_stage groups both at the pin level and the timing level.

CCS Noise Library Example

Example 13-1  CCS Noise Library Example

library (CCS_noise) {

technology ( cmos ) ;
delay_model : table_lookup;
time_unit : "1ps" ;
leakage_power_unit : "1pW" ;
voltage_unit : "1V" ;
current_unit : "1uA" ;
pulling_resistance_unit : "1kohm";
capacitive_load_unit(1000.000, ff);

nom_voltage : 1.200;
nom_temperature : 25.000;
nom_process : 1.000;

operating_conditions("OC1") {
  process : 1.000;
  temperature : 25.000;
  voltage : 1.200;
  tree_type : "balanced_tree";
}

default_operating_conditions:OC1;

lu_table_template(del_0_5_7_t) {
  variable_1 : input_net_transition;
  index_1("10.000, 175.000, 455.000, 980.000, 2100.000");
  variable_2 : total_output_net_capacitance;
  index_2("0.000000, 0.004000, 0.007000, 0.019000, 0.040000, 0.075000, 0.175000");
}

lu_table_template(ccsn_dc_29x29) {
  variable_1 : input_voltage;
  variable_2 : output_voltage;
}

lu_table_template(ccsn_timing_lut_5) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  variable_3 : time;
}

lu_table_template(ccsn_prop_lut_5) {
  variable_1 : input_noise_height;
  variable_2 : input_noise_width;
  variable_3 : total_output_net_capacitance;
  variable_4 : time;
}

lu_table_template(lu_table_template7x9) {
  variable_1 : input_net_transition;
  variable_2 : voltage;
}

cell(inv) {
  area : 0.75;
  pin(I) {
    direction : input;
    max_transition : 2100.0;
    capacitance : 0.002000;
    fanout_load : 1;
  }
  pin(Z) {
    direction : output;
    max_capacitance : 0.175000;
    max_fanout : 58;
    max_transition : 1400.0;
  }
function : "(I)’";

timing() {
    related_pin : "I";
    timing_sense : negative_unate;
    ...
}

ccsn_first_stage ( ) {
    is_needed : true;
    is_inverting : true;
    stage_type : both;
    miller_cap_rise : 0.00055;
    miller_cap_fall : 0.00084;

dc_current (ccsn_dc_29x29) {
    index_1 ("-1.200, -0.600, -0.240, -0.120, 0.000, 0.060, 0.120, 0.180, \
    0.240, 0.300, 0.360, 0.420, 0.480, 0.540, 0.600, 0.660, \ 
    0.720, 0.780, 0.840, 0.900, 0.960, 1.020, 1.080, 1.140, \ 
    1.200, 1.320, 1.440, 1.800, 2.400");
    index_2 ("-1.200, -0.600, -0.240, -0.120, 0.000, 0.060, 0.120, 0.180, \ 
    0.240, 0.300, 0.360, 0.420, 0.480, 0.540, 0.600, 0.660, \ 
    0.720, 0.780, 0.840, 0.900, 0.960, 1.020, 1.080, 1.140, \ 
    1.200, 1.320, 1.440, 1.800, 2.400");
    values ("619.332000, 0.548416, 0.510134, 0.491965, 0.470368, \ 
    ...
    -0.390604, -0.394495, -0.403571, -579.968000");
}

output_voltage_rise ( ) {
    vector (ccsn_timing_lut_5) {
        index_1(175.000);
        index_2(0.004000);
        index_3 ("104.222, 127.996, 144.729, 159.367, 176.983");
        values ("1.080, 0.840, 0.600, 0.360, 0.120");
    }
    ...
}

output_voltage_fall ( ) {
    vector (ccsn_timing_lut_5) {
        index_1(175.000);
        index_2(0.004000);
        index_3 ("104.222, 127.996, 144.729, 159.367, 176.983");
        values ("1.080, 0.840, 0.600, 0.360, 0.120");
    }
    ...
}

propagated_noise_low ( ) {
    vector (ccsn_prop_lut_5) {
        index_1(0.6000);
        index_2(1365.00);
        index_3 ("640.90, 679.55, 711.76, 755.45, 793.68");
        values ("0.0553, 0.0884, 0.1105, 0.0884, 0.0553");
    }
    ...
    vector (ccsn_prop_lut_5) {
        index_1(0.6500);
        index_2(2730.00);
        ...
    }
    ...
Conditional Data Modeling in CCS Noise Models

The following attributes support conditional data modeling in pin-based CCS noise models:

- The `when` attribute in the `ccsn_first_stage` and `ccsn_last_stage` groups.
- The `mode` attribute.

The following syntax shows the `when` and `mode` support for pin-based CCS noise models:

```plaintext
cell(cell_name) {
    mode_definition (mode_name) {
        mode_value(namestring) {
            when : "Boolean expression";
            sdf_cond : "Boolean expression";
        } ...
    }
    ...
}
pin(pin_name) {
    direction : input;
    /* The following syntax supports pin-based ccs noise */
    /* ccs noise first stage for Condition 1 */
    ccsn_first_stage() { 
        is_needed :true | false;
```
when : "Boolean expression";
    mode (mode_name, mode_value);
...
}

/* ccs noise first stage for Condition n */
ccsn_first_stage() {
    is_needed : true | false;
    when : "Boolean expression";
    mode (mode_name, mode_value);
...
}
}

pin(pin_name) {
     direction : output;
/* ccs noise last stage for Condition 1 */
ccsn_last_stage() {
    is_needed : true | false;
    when : "Boolean expression";
    mode (mode_name, mode_value);
...
}

/* ccs noise last stage for Condition n */
ccsn_last_stage() {
    is_needed : true | false;
    when : "Boolean expression";
    mode (mode_name, mode_value);
...
}

/* following are arc-based ccs noise */
timing() {
...
/* following are arc-based ccs noise */
ccsn_first_stage() {
    is_needed : true | false;
...
}

ccsn_last_stage() {
    is_needed : true | false;
...
}
}

when Attribute

The when attribute is a conditional attribute that is supported in pin-based CCS noise models in the ccsn_first_stage and ccsn_last_stage groups.
mode Attribute

The pin-based mode attribute is provided in the ccsn_first_stage and ccsn_last_stage groups for conditional data modeling. If the mode attribute is specified, mode_name and mode_value must be predefined in the mode_definition group at the cell level.

Example

```liberty
library (csm13os120_typ) {
  technology ( cmos ) ;
  delay_model : table_lookup;
  lu_table_template(ccsn_dc_29x29) {
    variable_1 : input_voltage;
    variable_2 : output_voltage;
  }
  cell(inv0d0) {
    area : 0.75;
    mode_definition(rw) {
      mode_value(read) {
        when : "I";
        sdf_cond : "I == 1";
      }
      mode_value(write) {
        when : "!I";
        sdf_cond : "I == 0";
      }
    }
    pin(I) {
      direction : input;
      max_transition : 2100.0;
      capacitance : 0.002000;
      fanout_load : 1;
      ...
    }
    pin(ZN) {
      direction : output;
      max_capacitance : 0.175000;
      max_fanout : 58;
      max_transition : 1400.0;
      function : "(I)";
      /* pin-based CCS noise first stage for Condition 1 */
      ccsn_first_stage () {
        ...
        when : "I"; /* or using mode as next commented line */
        /* mode(rw, read); */
        dc_current (ccsn_dc_29x29) { ... }
        output_voltage_rise () { ... }
        output_voltage_fall () { ... }
        propagated_noise_low () { ... }
        propagated_noise_high () { ... }
      } /* ccsn_last_stage */
      /* pin-based CCS noise last stage for Condition 2 */
      ccsn_last_stage () {
    }
  }
```
... when : "!I"; /* or using mode as next commented line */
/* mode(rw, read); */
dc_current (ccsn_dc_29x29) { ... }
output_voltage_rise() { ... }
output_voltage_fall() { ... }
propagated_noise_low() { ... }
propagated_noise_high() { ... }
} /* ccsn_last_stage */

timing() {
  related_pin : "I";
  timing_sense : negative_unate;
  ...
} /* timing I -> Z */
} /* Z */
} /* cell(inv0d0 */
} /* library */

---

**CCS Noise Modeling for Unbuffered Cells With a Pass Gate**

Unbuffered input and output latches are a special type of cell that has an internal memory node connected to an input or output pin. To increase the speed of the design and lower power consumption, these cells do not use inverters.

The major difference between an unbuffered output cell and an unbuffered input cell is as follows:

- **Unbuffered Output Cell**

  An unbuffered output cell has the *feedback*, or back-driving path, from the unbuffered output pin to an internal node. In Figure 13-1, Q is connected to internal node \( D_1 \) through the IR inverter.

*Figure 13-1 Unbuffered Output Latch*

- **Unbuffered Input Cell**
The input pin of an unbuffered cell is not buffered and can be connected through a pass gate to the internal node. (A pass gate is a special gate that has an input and an output and a control input. If the control is set to true, the output is driven by the input. Otherwise, it is a floating output.) For example, in Figure 13-2, D is connected to internal node D_1 through a pass gate.

Figure 13-2  Unbuffered Input Latch

To correctly model this category of cells, you must determine:

- If a pin is buffered or unbuffered.
- If a pin is implemented with a pass gate.
- If the ccsn_*_stage information models a pass gate.

Syntax for Unbuffered Output Latches

```plaintext
/* unbuffered output pin */
pin (pin_name) {
  direction : inout | output;
  is_unbuffered : true | false;
  has_pass_gate : true | false;
  ccsn_first_stage () {
    is_pass_gate : true | false;
    ...
  }
  ...
  ccsn_last_stage () {
    is_pass_gate : true | false;
    ...
  }
  ...
  timing() {
    ccsn_first_stage () {
      is_pass_gate : true | false;
      ...
    }
  }
```

Pin-Level Attributes

The following attributes are pin-level attributes for unbuffered output latches.

is_unbuffered Attribute

The is_unbuffered simple Boolean attribute indicates that a pin is unbuffered. This optional attribute can be specified on the pins of any library cell. The default is false.
hs_pass_gate Attribute

The has_pass_gate simple Boolean attribute can be defined in a pin group to indicate whether the pin is internally connected to at least one pass gate.

ccsn_first_stage Group

The ccsn_first_stage group specifies CCS noise for the first stage of the channel-connected block (CCB). When the ccsn_first_stage group is defined at the pin level, it can only be defined in an input pin or an inout pin.

The ccsn_first_stage group syntax models back-driving CCS noise propagation information from the output pin to the internal node.

is_pass_gate Attribute

The is_pass_gate Boolean attribute is defined in a ccsn_*_stage group (such as ccsn_first_stage) to indicate whether the ccsn_*_stage information is modeled for a pass gate. The attribute is optional and its default is false.

CCS Noise Modeling for Multivoltage Designs

In multivoltage designs, a complex macro cell can have multiple power domains with different power supplies internal to the cell. The internal power supplies that provide power to some of the inputs and outputs of the channel-connected blocks (CCBs), cannot be modeled at the pin level. Therefore, pin-level attributes do not correctly describe the operation of the CCS noise stages for these channel-connected blocks. To correctly model the CCS noise stages, specify the internal power supplies in the ccsn_first_stage and ccsn_last_stage groups that are specified within the pin or timing groups.

Figure 13-3 shows a macro cell with three power domains, PD1, PD2, and PD3. The input pin, IN, is connected to the channel-connected blocks (not shown in the figure) for two CCS noise stages in PD1 and PD2. The outputs of the channel-connected blocks go to PD3. The CCS noise stage for the output pin, Y, is in PD1 and is driven by an internal stage in PD3. The example in Figure 13-3 shows that the channel-connected block inputs or outputs for the CCS stages do not always map to a pin.
To model such CCS noise stages, use the `output_signal_level` and the `input_signal_level` attributes respectively within the `ccsn_first_stage` and `ccsn_last_stage` groups, as shown in Example 13-2.

**Example 13-2 Pin and Timing Based CCS Noise Model of Low-to-High Level-Shifter Cells**

```liberty
library (test) {
  technology(cmos);
  nom_voltage : 1.080000;
  voltage_unit : "1V";
  voltage_map(VDD1,1.080000);
  voltage_map(GND1,0.000000);
  voltage_map(VDD2,0.900000);
  ... 
  cell (LVL SHIFT L2H 1) {
    is_level_shifter : true;
    level_shifter_type : LH;
    input_voltage_range(0.900000,1.320000);
    output_voltage_range(0.900000,1.320000);
    pg_pin (VDD) {
      voltage_name : VDD1;
      ... 
    }
    pg_in (VDDL) {
      voltage_name : VDD2;
      ... 
    }
  }
  pin (I) {
    direction : input;
    level_shifter_data_pin : true;
    related_ground_pin : VSS;
    related_power_pin : VDDL;
    ccsn_first_stage () {/* pin-based model */
      is_needed : true;
      stage_type : both;
    }
  }
}
```
Chapter 13: Composite Current Source Signal Integrity Modeling

... output_signal_level : VDD1;
/* Specifies that the power supply to the ccsn_first_stage output is VDD1. The power supply to the ccsn_first_stage_input is specified at the pin level by the related_power_pin attribute as VDDL. */

dc_current (ccsn_dc)...}
}

...)
/* pin-based model */

is_needed : "true";
stage_type : "both";
...

input_signal_level : VDD2;
/* Specifies that the power supply to the ccsn_last_stage input is VDD2. The power supply to the ccsn_last_stage output is specified at the pin level by the related_power_pin attribute as VDD. */

dc_current (ccsn_dc)...}
...
}

cell (LVL_SHIFT_L2H_2) {
  is_level_shifter : true;
  level_shifter_type : LH;
  input_voltage_range(0.900000,1.320000);
  output_voltage_range(0.900000,1.320000);
  pg_pin (VDD) {
    voltage_name : VDD1;
    ...
  }
  pg_in (VDDL) {
    voltage_name : VDD2;
    ...
  }
  pin (I) {
    direction : input;
    level_shifter_data_pin : true;
    related_ground_pin : VSS;
    related_power_pin : VDDL;
  }
  pin (Z) {
    direction : output;
    power_down_function : "!VDD + !VDDL + VSS";
    function : "I";
  }
}
Referenced CCS Noise Modeling

For CCS noise characterization, a circuit is partitioned into channel-connected blocks (CCBs). An input pin might simultaneously drive multiple channel-connected blocks (CCBs).

For different input states and capacitive loads, you can represent each channel-connected block using multiple input_ccb and output_ccb groups. The input_ccb and output_ccb groups have names and are defined in pin groups.

In each timing arc, you can use these names to reference the input_ccb or output_ccb groups that

- Contribute to the noise but do not propagate the noise in the timing arc; this also applies to pin-level receiver_capacitance groups
- Propagate the noise in the timing arc

Because you define the input_ccb and output_ccb groups only in the pin groups, this model eliminates multiple definitions of the same channel-connected block noise in different timing groups, such as for a ccsn_last_stage group representing an inverter in a conventional two-stage CCS noise model.
To better model noise, the input_ccb and output_ccb groups include the output_voltage_rise and output_voltage_fall groups. The output_voltage_rise and output_voltage_fall values are characterized with driver waveforms instead of ramp waveforms as in the conventional two-stage CCS noise model.

For more information about the conventional two-stage CCS noise model, see “CCS Signal Integrity Modeling Syntax” on page 13-2.

---

**Modeling Syntax**

```
library (library_name) {
  cell (cell_name) {
    driver_waveform : "waveform_name";
    driver_waveform_rise : "waveform_name";
    driver_waveform_fall "waveform_name";
    ...
    pin (pin_name) {
      direction : input;
      ...
      input_ccb (input_ccb_name1) {
        when : logical_expression;
        mode(mode_name, mode_value);
        related_ccb_node : "spice_node_name1";
        output_voltage_rise {
          ...
        }
        ...
      } /* end input_ccb group */
      input_ccb (input_ccb_name2) {
        when : logical_expression
        mode(mode_name, mode_value);
        related_ccb_node : "spice_node_name2";
        output_voltage_rise {
          ...
        }
        ...
      } /* end input_ccb group */
      receiver_capacitance (lu_template) {
        when : logical_expression;
        mode(mode_name, mode_value);
        active_input_ccb(input_ccb_name1, input_ccb_name2,...]);
        } /* end receiver_capacitance group */
    } /* end pin a group */
  } /* end cell a group */
  pin (pin_name) {
    direction : output;
    ...
    output_ccb(output_ccb_name1) {
      when : logical_expression;
      mode(mode_name, mode_value);
    } /* end output_ccb group */
  } /* end pin b group */
} /* end library a group */
```
related_ccb_node : "$spice_node_name3$";
}
timing() { /* Arc has propagating noise (full arc-based) */
...
  active_input_ccb(input_ccb_name1, input_ccb_name2, ...);
  propagating_ccb(input_ccb_name2, output_ccb_name);
}
timing() { /* Arc has no propagating noise because it involves
three inverting stages*/
...
  active_input_ccb(input_ccb_name1, input_ccb_name2);
  active_output_ccb(output_ccb_name);
}
} /* end pin group */
} /* end cell group */
} /* end library group */

---

Cell-Level Attributes

This section describes the optional cell-level attributes specific to the referenced CCS noise model.

**driver_waveform Attribute**

The `driver_waveform` attribute specifies a cell-specific driver waveform. The specified driver waveform must be a predefined `driver_waveform_name` attribute value in the library-level `normalized_driver_waveform` group table.

**driver_waveform_rise and driver_waveform_fall Attributes**

The `driver_waveform_rise` and `driver_waveform_fall` specify the rise and fall driver waveforms. These attributes override the `driver_waveform` attribute.

For more information about driver waveform modeling syntax, see “Driver Waveform Support” on page 7-88.

---

Pin-Level Groups

This section describes the pin-level groups and attributes specific to the referenced CCS noise model.

**input_ccb and output_ccb Groups**

The `input_ccb` and `output_ccb` groups include all the attributes and subgroups of the `ccsn_first_stage` and `ccsn_last_stage` groups respectively.
Except for pins where the `input_ccb` and `output_ccb` groups are not required, you must name all these groups so that they can be referenced. For the exceptions, you must set the `is_needed` attribute to `false` in the `input_ccb` and `output_ccb` groups.

The name includes the pin identifier and the group identifier. The pin identifier is the same in all the `input_ccb` and `output_ccb` groups of the pin. The pin identifier enables homogeneous treatment of these groups in buses and bundles.

For example, in the `input_ccb("CCB_D1")` group, the `CCB_D` part of the name indicates that the channel-connected block group belongs to pin D and `1` indicates that this is the first `input_ccb` group of pin D.

Note: The library characterization tools can use an additional cell identifier in the `input_ccb` and `output_ccb` group names.

**Conditional Data Modeling**

Use the `mode` and `when` attributes in the `input_ccb` and `output_ccb` groups for conditional data modeling.

**related_ccb_node Attribute**

The optional `related_ccb_node` attribute specifies the SPICE node in the subcircuit netlist that is used for the `dc_current` table characterization. The attribute is defined in the `input_ccb` group of an input pin and the `output_ccb` group of an output pin.

**output_voltage_rise and output_voltage_fall Groups**

The `output_voltage_rise` and `output_voltage_fall` groups have the following differences from those in the CCS noise stage groups.

- The output waveform specified by the `index_3` and `values` attributes is characterized using a driver waveform and not a ramp waveform. So, the `index_1` or `input_net_transition` is a slew value based on slew trip points and library slew derate and not a rail-to-rail ramp transition time.

- The `index_1` (`input_net_transition`) values must match one of the `index_1` values of the driver waveform specified at the cell-level.

- The `index_2` (`total_output_net_capacitance`) values must match one of the `index_2` values of the `cell_rise` and `cell_fall` groups.

- The values at `index_1` and `index_2` of all the `output_voltage_rise` or `output_voltage_fall` groups in an `input_ccb` or `output_ccb` group must form a grid without duplicated or missing points.
For an input_ccb group that does not directly drive an output pin, the number of output_voltage_rise groups must be M x 1. For an output_ccb or an input_ccb that directly drives an output pin, the number of groups must be M x N.

M is the number of index_1 values and N is the number of index_2 values.

Note:
For libraries with 7 x 7 or 8 x 8 nonlinear delay model (NLDM) tables, both M and N can be 4.

---

**Timing-Level Attributes**

This section describes the timing-level attributes specific to the referenced CCS noise model.

**active_input_ccb Attribute**

The active_input_ccb attribute lists the input_ccb groups of the input pin that are active or switching in the timing arc or the receiver capacitance load but do not propagate the noise to the output pin. In the timing group, the input pin is specified by the related_pin attribute.

You can also specify this attribute in the receiver_capacitance group of the input pin.

**active_output_ccb Attribute**

The active_output_ccb attribute lists the output_ccb groups in the timing arc that drive the output pin, but do not propagate the noise. You must define both the output_ccb and timing groups in the same pin group.

**propagating_ccb Attribute**

The propagating_ccb attribute lists all the channel-connected block groups that propagate the noise in a particular timing arc, to the output pin.

In the list, the first name is the input_ccb group of the input pin (specified by the related_pin attribute in the timing group). The second name, if present, is for the output_ccb group of the output pin.

Note:
You can list at most two names. This attribute does not support timing arcs with three or more inverting stages. For such cases and complex circuits with converging paths, reference the input_ccb group name in the active_input_ccb attribute and the output_ccb group name in the active_output_ccb attribute.
Examples

The following example shows a simple noise model with two timing arcs sharing the same output channel-connected block.

cell (AND2) {
    ...
    pin (A) {
        ...
        input_ccb("CCB_A") {
            related_ccb_node : "net1:15";
            ...
        }
    }
    pin (B) {
        ...
        input_ccb("CCB_B") {
            related_ccb_node : "net1:15";
            ...
        }
        input_ccb("CCB_CP2") {
            related_ccb_node : "net3:3";
            ...
        }
    }
    pin (Y) {
        ...
        output_ccb("CCB_Y") {
            related_ccb_node : "net1:15";
            ...
        }
        timing () {
            related_pin : A;
            propagating_ccb("CCB_A", "CCB_Y");
        }
        timing () {
            related_pin : B;
            propagating_ccb("CCB_B", "CCB_Y");
        }
        ...
    }
}

The following figure is a schematic with the input, A, driving three logic gates. The relevant channel-connected blocks are shown.
Figure 13-4  A Schematic With Multiple Active Channel-Connected Blocks

The following table shows which of the four active channel-connected blocks propagate the noise in the A→Z timing arc.

<table>
<thead>
<tr>
<th>Active channel-connected block</th>
<th>Propagates noise in A→Z arc?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_NOT1</td>
<td>No</td>
</tr>
<tr>
<td>A_NOT2</td>
<td>No</td>
</tr>
<tr>
<td>A_NAND1</td>
<td>Yes</td>
</tr>
<tr>
<td>Z_INV</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The following example shows how the blocks are referenced in the timing group.

```plaintext
pin(Z) {
    direction : output;
    timing () { /* A→Z arc */
        related_pin : A;
        ...
        active_input_ccb("A_NOT1", "A_NOT2");
        propagating_ccb("A_NAND1", "Z_INV");
    } /* end timing */
}
```
This chapter provides an overview of composite current source (CCS) modeling to support advanced technologies. It covers the syntax for CCS power modeling in the following sections:

- Composite Current Source Power Modeling
- Compact CCS Power Modeling
- Composite Current Source Dynamic Power Examples
Composite Current Source Power Modeling

The library nonlinear power model format captures leakage power numbers in multiple input combinations to generate a state-dependent table. It also captures dynamic power of various input transition times and output load capacitance to create the state-dependent and path-dependent internal energy data.

The composite current source (CCS) power modeling format extends current library models to include current-based waveform data to provide a complete solution that addresses static and dynamic power. It also addresses dynamic IR drop. The following are features of this approach as compared to the nonlinear power model:

- Creates a single unified power library format suitable for power optimization, power analysis, and rail analysis.
- Captures a supply current waveform for each power or ground pin.
- Provides finer time resolution.
- Offers full multivoltage support.
- Captures equivalent parasitic data to perform fast and accurate rail analysis.
- Reduces the characterization runtime.

Cell Leakage Current

Because CCS power is current-based data, leakage current on the power and ground pins is captured instead of leakage power as specified in the nonlinear power model format. For information about gate leakage, see “gate Leakage Group” on page 14-4. The leakage current syntax is as follows:

Example 14-1 Leakage Current Syntax

```plaintext
cell(cell_name) {
    ...
    leakage_current() {
        when : "Boolean expression";
        pg_current(pg_pin_name) {
            value : float;
        }
    }
    ...
    leakage_current() { /* without the when statement */
        /* default state */
        ...
    }
}
```
Current conservation means that the sum of all current values must be zero. A positive value means power pin current, and a negative value means ground pin current.

Again, a simplified format is allowed for a cell with a single power and ground pin. For this case, no `pg_current` group is required within a `leakage_current` group.

**Example 14-2  Leakage Current Format Simplified**

```plaintext
cell(cell_name) {
 ...
 leakage_current() { /* without pg_current group*/
     when : "Boolean expression";
     value : float;
 }

 leakage_current() { /* without the when statement */
     /* default state */
     ...
 }
 ...
}
```

---

**Gate Leakage Modeling in Leakage Current**

The syntax for these power models is described in the following section.

**Syntax**

```plaintext
cell(cell_name) {
 ...
 leakage_current() {
     when : "Boolean expression";
     pg_current(pg pin name) {
         value : float;
     }
 ...
 gate_leakage(input pin name) {
         input_low_value : float;
         input_high_value : float;
     }
 ...
 }
 leakage_current() {
     /* group without when statement */
     /* default state */
     ...
 }
 ...
}```
gate_leakage Group

This group specifies the cell’s gate leakage current on input or inout pins within the leakage_current group in a cell. For information about cell leakage, see “Cell Leakage Current” on page 14-2.

The following information pertains to a gate_leakage group:

• Groups can be placed in any order if there are more than one gate_leakage groups within a leakage_current group.

• Leakage current of a cell is characterized with opened outputs, which means outputs of a modeling cell do not drive any other cells. Outputs are assumed to have zero static current during the measurement.

• A missing gate_leakage group is allowed for certain pins.

• Current conservation is applicable if it can be applied to higher error tolerance.

input_low_value Attribute

This attribute specifies gate leakage current on an input or inout pin when the pin is in a low state.

• A negative float value is required.

• The gate leakage current is measured from the power pin of the cell to the ground pin of its driver cell.

• The input pin is pulled low.

• The input_low_value attribute is not required for a gate_leakage group.

• Defaults to 0 if no gate_leakage group is specified for certain pins.

• Defaults to 0 if no input_low_value attribute is specified in the gate_leakage group.

input_high_value Attribute

This attribute specifies gate leakage current on an input or inout pin when the pin is in a high state.

• A positive float value is required.

• The gate leakage current is measured from the power pin of its driver cell to the ground pin of the cell.

• The input pin is pulled high.

• The input_high_value is not required for a gate_leakage group.
Intrinsic Parasitic Models

The intrinsic parasitic model syntax consists of two parts: intrinsic resistance and intrinsic capacitance.

cell (cell_name) {
  mode_definition (mode_name) {
    mode_value(namestring) {
      when : "Boolean expression";
      sdf_cond : "Boolean expression";
    }
  }...
  intrinsic_parasitic() {
    mode (mode_name, mode_value);
    when : "Boolean expression";
    intrinsic_resistance(pg_pin_name) {
      related_output : output_pin_name;
      value : float;
    }
    intrinsic_capacitance(pg_pin_name) {
      value : float;
    }
  }
}

/*without when statement */
/* default state */

Example 14-3  Typical Intrinsic Parasitic Model Using Modes

library (csm13os120_typ) {
  technology ( cmos ) ;
  delay_model : table_lookup;
  lu_table_template(ccsn_dc_29x29) {
    variable_1 : input_voltage;
    variable_2 : output_voltage;
  }
  cell(inv0d0) {
    area : 0.75;
    pg_pin(V1) {
      voltage_name : VDD1;
      pg_type : primary_power;
    }
    pg_pin(G1) {
voltage_name : GND1;
pq_type : primary_ground;
}
mode_definition(rw) {
 mode_value(read) {
 when : "A1";
 sdf_cond : "A1 == 1";
 }
 mode_value(write) {
 when : "!A1";
 sdf_cond : "A1 == 0";
 }
}

pin(A1) {
 direction : input;
capacitance : 0.1;
 related_power_pin : V1;
 related_ground_pin : G1;
}

pin(A2) {
 direction : input;
capacitance : 0.1;
 related_power_pin : V1;
 related_ground_pin : G1;
}

pin(ZN) {
 direction : output;
max_capacitance : 0.1;
 function : "!A1+A2";
 related_power_pin : V1;
 related_ground_pin : G1;
 timing() {
 timing_sense : "negative_unate"
 related_pin : "A1";
 ... 
}
}

pin(ZN1) {
 direction : output;
max_capacitance : 0.1;
 function : "!A1";
 related_power_pin : V1;
 related_ground_pin : G1;
 timing() {
 timing_sense : "negative_unate"
 related_pin : "A1 A2";
 ... 
}
}
intrinsic_parasitic() {
    mode(rw, read);
    intrinsic_resistance(G1) {
        related_output : "ZN";
        value : 9.0;
    }
    intrinsic_capacitance(G1) {
        value : 8.2;
    }
}

/* cell(inv0d0) */
} /* library */

**Voltage-Dependent Intrinsic Parasitic Models**

Intrinsic parasitics are conventionally modeled as voltage-independent or steady-state values. However, intrinsic parasitics are voltage-dependent. To better represent intrinsic parasitics in a CCS power model, use a lookup table for intrinsic parasitics instead of a single steady-state value. You can use the steady-state value when your design requirements are not critical.

The lookup table is one-dimensional and consists of intrinsic parasitic values for different values of VDD. You can selectively add these values to any `intrinsic_resistance` or `intrinsic_capacitance` subgroup. Use lookup tables when correct estimation of voltage drops is critical, such as power-switch designs.

The following are the advantages of using lookup tables.

- Accurate estimation of peak-inrush current and wake-up time
- Optimal power-up and power-down sequencing
- Optimal power-switch design, that is, minimum number of used and placed power-switch cells

**Example 14-4 Syntax of Voltage-Dependent Intrinsic Parasitic Model With Lookup Tables**

```plaintext
lu_table_template (template_name) {
    variable_1 : pg_voltage | pg_voltage_difference;
    index_1 ("float, ... float");
}
cell (cell_name) {
    ...
    intrinsic_parasitic() {
        when : "Boolean expression";
        intrinsic_resistance (pg_pin_name) {
            related_output : output_pin_name;
            value : float;
            reference_pg_pin : pg_pin_name;
            lut_values (template_name) {
                index_1 ("float, ... float");
                values ("float, ... float");
            }
        }
    }
}
```
intrinsic_capacitance(pg_pin_name) {
    value : float;
    reference_pg_pin : pg_pin_name;
    lut_values (template_name) {
        index_1 ("float, ... float");
        values ("float, ... float");
    }
}

Example 14-5  Example of Voltage-Dependent Intrinsic Parasitic Model

library(example_library) {
    ..... 
    lu_table_template (test_voltage) {
        variable_1 : pg_voltage;
        index_1 ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
    }
    cell (AND3) {
        ..... 
        intrinsic_parasitic() {
            when : "A1 & A2 & ZN";
            intrinsic_resistance(G1) {
                related_output : "ZN";
                value : 9.0;
                reference_pg_pin : G1;
                lut_values (test_voltage) {
                    index_1 ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
                    values ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
                }
            }
        }
    }
    intrinsic_capacitance(G2) {
        value : 8.2;
        reference_pg_pin : G2;
        lut_values (test_voltage) {
            index_1 ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
            values ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
        }
    }
    intrinsic_resistance(G1) {
        related_output : "ZN1";
        value : 62.2;
    }
}
intrinsic_parasitic() {
}
/* default state */
intrinsic_resistance(G1) {
  related_output : "ZN";
  value : 9.0;
  reference_pg_pin : G1;
  lut_values ( test_voltage ) {
    index_1 ( "0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0" );
    values ( "0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0" );
  }
}
intrinsic_resistance(G1) {
  related_output : "ZN1";
  value : 9.0;
}
intrinsic_capacitance(G2) {
  value : 8.2;
  reference_pg_pin : G2;
  lut_values ( test_voltage ) {
    index_1 ( "0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0" );
    values ( "0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0" );
  }
}

Library-Level Group

The following library-level group models voltage-dependent intrinsic parasitics.

lu_table_template Group

This group defines the template for the lut_values group. The lu_table_template group includes a one-dimensional variable, variable_1. The valid values of the variable_1 variable are pg_voltage and pg_voltage_difference. When the variable_1 variable is set to the pg_voltage value, the values of the intrinsic capacitance or resistance directly vary with the power supply voltage of the cell. When the variable_1 variable is set to the pg_voltage_difference value, the values of the intrinsic capacitance or resistance vary with the difference between the power supply voltage and the power pin voltage specified by the reference_pg_pin attribute.

Note:
The reference_pg_pin attribute specifies the reference pin for the intrinsic_resistance and intrinsic_capacitance groups. The reference pin must be a valid PG pin.
Pin-Level Group

The following pin-level group models voltage-dependency in intrinsic parasitics by using lookup tables.

lut_values Group

To use the lookup table for intrinsic parasitics, use the lut_values group. You can add the lut_values group to both the intrinsic_resistance and intrinsic_capacitance groups. The lut_values group uses the variable_1 variable, which is defined within the lu_table_template group, at the library level.

Parasitics Modeling in Macro Cells

For macro cells, the total_capacitance group is provided within the intrinsic_parasitic group.

total_capacitance Group

The total_capacitance group specifies the macro cell’s total capacitance on a power or ground net within the intrinsic_parasitic group.

- This group can be placed in any order if there is more than one total_capacitance group within an intrinsic_parasitic group.
- If the total_capacitance group is not defined for a certain power and ground pin, the value of capacitance defaults to 0.0. The default is provided by the tool.
- The parasitics modeling of the total capacitance in macros cells is not state dependent. This means that there is no state condition specified in the intrinsic_parasitic group.

Parasitics Modeling Syntax

```plaintext
cell (cell_name) {
    power_cell_type : enum(stdcell, macro);
    ...
    intrinsic_parasitic() {
        total_capacitance(pg pin name) {
            value : float;
        }
    }
    ...
}
```
Dynamic Power

Because CCS power is current-based data, instantaneous power data on the power or ground pin is captured instead of internal energy specified in the nonlinear power model format. The current-based data provides higher accuracy than the existing model.

In the CCS modeling format, instantaneous power data is specified as a table of current waveforms. The table is dependent on the transition time of a toggling input and the capacitance of the toggling outputs.

As the number of output pins increases in a cell, the number of waveform tables becomes large. However, the cell with multiple output pins (more than one output) does not need to be characterized for all possible output load combinations. Therefore, two types of methods can be introduced to simplify the captured data.

- Cross type - Only one output capacitance is swept, while all other output capacitances are held in a typical value or fixed value.
- Diagonal type - The capacitance to all the output pins is swept together by an identical value.

A table that is modeled based on these two types is defined as a sparse table. Otherwise it is defined as a dense table, meaning that all combinations of the output load variable are specified in tables.

Dynamic Power and Ground Current Table Syntax

You can use the following syntax for dynamic current:

```
pg_current_template(template_name_1) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  variable_3 : time;
  index_1(float, );                /* optional */
  index_2(float, );                /* optional */
  index_3(float, );                /* optional */
}

pg_current_template(template_name_2) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  variable_3 : total_output_net_capacitance;
  variable_4 : time;
  index_1(float, );                /* optional */
  index_2(float, );                /* optional */
  index_3(float, );                /* optional */
  index_4(float, );                /* optional */
}
```
Dynamic Power Modeling in Macro Cells

The extensions to CCS dynamic power format provides more accurate models for macro cells. The current dynamic power model only supports current waveforms for single-input events.

The model can also be applied to memory modeling with synchronous events, which are triggered by toggling either a single read_enable or write_enable.

However, for asynchronous event, the read access can be triggered by more than one bit of the address bus toggling. To support asynchronous memory access for macro cells, use min_input_switching_count and max_input_switching_count, in dynamic power as shown in the next section.

The following syntax for dynamic power format provides more accurate models for macro cells:

```plaintext
... cell(cell_name) { 
  mode_definition (mode_name) { 
    mode_value(namestring) { 
      when : "Boolean expression";
      sdf_cond : "Boolean expression";
    } 
  } 
  power_cell_type : enum(stdcell, macro)
  dynamic_current() { 
    mode (mode_name, mode_value);
    when : "Boolean expression";
    related_inputs : "input_pin_name";
    switching_group() { 
      min_input_switching_count : integer;
      max_input_switching_count : integer;
      pg_current(pg_pin_name) { 
        vector(template_name) { 
          reference_time : float;
          index_1(float);
          index_2("float,...");
          values("float,...");
        } /* end vector group */
        ... 
      } /* end pg_current group */
      ... 
    } /* end switching_group */
    ... 
  } /* end dynamic_current group */
  ... 
} /* end cell group* /
```

The min_input_switching_count and max_input_switching_count attributes specify the number of bits in the input bus that are switching simultaneously while an asynchronous
event occurs. A single switching bit can be defined by setting the same value in both attributes.

The following example shows that any three bits specified in related_inputs are switching simultaneously.

```c
... min_input_switching_count : 3;
max_input_switching_count : 3;
...
```

A range of switching bits can be defined by setting the minimum and maximum value. The following example shows that any 2, 3, 4 or 5 bits specified in related_inputs are switching simultaneously.

```c
... min_input_switching_count : 2;
max_input_switching_count : 5;
...
```

**min_input_switching_count Attribute**

This attribute specifies the minimum number of bits in an input bus that are switching simultaneously.

- The count must be integer.
- The count must greater than 0 and less than max_input_switching_count.

**max_input_switching_count Attribute**

This attribute specifies the maximum number of bits in an input bus that are switching simultaneously.

- The count must be integer.
- The count must greater than min_input_switching_count.
- The count must be less than the total number of bits listed in related_inputs.

**Examples for CCS Dynamic Power for Macro Cells**

```c
... pg_current_template ( CCS_power_1 ) {
  variable_1 : input_net_transition ;
  variable_2 : time ;
}
type(bus3) {
  base_type : array ;
  bit_width : 3 ;
  ...
}
```
cell ( example ) {
  bus(addr_in) {
    bus_type : bus3;
    direction : input;
    ...
  }
  pin(data_in) {
    direction : input;
    ...
  }
}

power_cell_type : macro;
  dynamic_current() {
    when: "!WE";
    related_inputs : "addr_in";
    switching_group () {
      min_input_switching_count : 1;
      max_input_switching_count : 3;
      pg_current (VSS) {
        vector ( CCS_power_1 ) {
          reference_time : 0.01;
          index_1 ("0.01")
          index_2 ("4.6, 5.9, 6.2, 7.3")
          values ("0.002, 0.009, 0.134, 0.546")
        }
        ...
        vector ( CCS_power_1 ) {
          reference_time : 0.01;
          index_1 ("0.03")
          index_2 ("2.4, 2.6, 2.9, 4.0")
          values ("0.012, 0.109, 0.534, 0.746")
        }
        vector ( CCS_power_1 ) {
          reference_time : 0.01;
          index_1 ("0.08")
          index_2 ("1.0, 1.6, 1.8, 1.9")
          values ("0.102, 0.209, 0.474, 0.992")
        }
        ...
      } /* pg_current */
    } /* switching_group */
  } /* dynamic_current */
} /* intrinsic_parasitic */
leakage_current() {
    when : WE;
    gate_leakage(data_in) {
        input_low_value : -0.3;
        input_high_value : 0.5;
    }
    ...
} /* leakage_current */
...
} /* end of cell */
...

Conditional Data Modeling for Dynamic Current Example

library (csm13os120_typ) {
    technology ( cmos ) ;
    delay_model : table_lookup;
    lu_table_template(ccsn_dc_29x29) {
        variable_1 : input_voltage;
        variable_2 : output_voltage;
    }
    cell(inv0d0) {
        area : 0.75;
        pg_pin(V1) {
            voltage_name : VDD1;
            pg_type : primary_power;
        }
        pg_pin(G1) {
            voltage_name : GND1;
            pg_type : primary_ground;
        }
        mode_definition(rw) {
            mode_value(read) {
                when : "A1";
                sdf_cond : "A1 == 1";
            }
            mode_value(write) {
                when : "!A1";
                sdf_cond : "A1 == 0";
            }
        }
        power_cell_type : stdcell;
        dynamic_current() { /* dense table */
            mode(rw, read);
            related_inputs : "A2";
            related_outputs : "ZN ZN1";
            switching_group() {
                output_switching_condition(rise rise);
                pg_current(V1) {
                    vector(test_1) {
reference time : 23.7;
index_1("0.8");
index_2("0.7");
index_3("10.4");
index_4("8.2 8.5 9.1 9.4 9.8");
values("0.7 34.6 3.78 92.4 100.1");
}
...
}
}
}

pin(A1) {
direction : input;
capacitance : 0.1 ;
related_power_pin : V1;
related_ground_pin : G1;
}
pin(A2) {
direction : input;
capacitance : 0.1 ;
related_power_pin : V1;
related_ground_pin : G1;
}
pin(ZN) {
direction : output;
max_capacitance : 0.1;
function : "!A1+A2";
related_power_pin : V1;
related_ground_pin : G1;
timing() {
    timing_sense : "negative_unate"
    related_pin : "A1"
    ...
}
}

pin(ZN1) {
direction : output;
max_capacitance : 0.1;
function : "!A1";
related_power_pin : V1;
related_ground_pin : G1;
timing() {
    timing_sense : "negative_unate"
    related_pin : "A1 A2"
    ...
}
}

} /* cell(inv0d0) */
} /* library */
Dynamic Current Syntax

The syntax in Example 14-6 is used for instantaneous power data, which is captured at the cell level.

Example 14-6  Dynamic Current Syntax

```c
cell(cell_name) {
    power_cell_type : enum(stdcell, macro)
    dynamic_current() {
        when : "Boolean expression";
        related_inputs : input_pin_name;
        related_outputs : output_pin_name;
        typical_capacitances(float, );/* applied for cross type;*/
        switching_group() {
            input_switching_condition(enum(rise, fall));
            output_switching_condition(enum(rise, fall));
            pg_current(pg_pin_name) {
                vector(template_name) {
                    reference_time : float;
                    index_output : output_pin_name; /* applied for cross type;*/
                    index_1(float);
                    index_n(float);
                    index_n+1(float, );
                    values(float, );
                    } /* vector */
            } /* pg_current */
        } /* switching_group */
    } /* dynamic_current */
} /* cell */
```

Note:
For the input_switching_condition and input_switching_condition groups, the rise and fall values can be separated by commas or space.

Compact CCS Power Modeling

CCS power compaction uses base curve technology to significantly reduce the library size of CCS power libraries. Greater control of the Liberty file size allows you to include additional data points and more accurately capture CCS power data for dynamic current waveforms.
Base curve technology was first introduced for compact CCS timing. Each timing current waveform is split into two segments in the I-V domain, and the shape of each segment is modeled by a base curve that has a similar shape. This segmentation prevents direct modeling with the piecewise linear data points.

Compact CCS power modeling differs from compact CCS timing modeling in that power waveforms can include both positive sections and negative sections. They must be modeled in an I(t) domain. In addition, the segmentation is more flexible. This is important because the current waveform shape might contain one or more bumps. The compact CCS power modeling segmentation points can be selected at:

- The point where the current waveform crosses zero
- The peak of a current bump

In Figure 14-1, the red curve shows an example CCS power waveform in piecewise linear format with fifteen data points. Thirty float numbers must be stored in the library. This is an expensive storage cost for a single I(t) waveform because libraries generally include a large number of I(t) waveforms. In addition, the waveform shape is not smooth, despite the fifteen data points, due to the inefficiency of the piecewise linear representation. The current value error is larger than 5% in some regions of the waveform.

Figure 14-1  Power Current Vector

Segmenting the waveform and using base curve technology for each segment provides greater accuracy. In Figure 14-1, the blue dots and blue curve show the waveform using base curve technology. The blue dots represent five segmentation points that divide the waveform into four sections. You can save the segmentation points as characteristic points and model the shape of each segment using a base curve. The blue curve is the third segment that can be represented by a base curve.

The following example describes the format for each current waveform:

\[ t_{\text{start}}, I_{\text{start}}, \text{bcid}_1, t_{\text{ip}1}, I_{\text{ip}1}, \text{bcid}_2, [ t_{\text{ip}2}, I_{\text{ip}2}, \text{bcid}_3, \ldots ], t_{\text{end}}, I_{\text{end}}; \]
The arguments are defined as follows:

\( t_{\text{start}} \)
- The current start time.

\( I_{\text{start}} \)
- The initial current.

\( t_{\text{ip}} \)
- The time of an internal segmentation point.

\( I_{\text{ip}} \)
- The current value of an internal segmentation point.

\( t_{\text{end}} \)
- The time when transition ends and current value becomes stable.

\( I_{\text{end}} \)
- The current value at the endpoint.

\( \text{bcid} \)
- The ID of the base curve that models the shape between two neighboring points.

---

### Compact CCS Power Syntax and Requirements

The expanded, or dynamic, CCS power model syntax provides an important reference and criteria for compact CCS power modeling. See “Dynamic Current Syntax” on page 14-17 for the dynamic_current syntax and see “Dynamic Power and Ground Current Table Syntax” on page 14-11 for the pg_current_template syntax.

The following requirements must be met in the pg_current_template group:

- The last variable_ * value must be time. The time variable is required.
- The input_net_transition and total_output_net_capacitance values are available for all variable_ * attributes except the last variable_*. They can be placed in any order except last.

The following conditions describe the pg_current_template group:

- There can be zero or one input_net_transition variable.
- There can be zero, one, or two total_output_net_capacitance variables.
• Each vector group in the pg_current group describes a current waveform that is compacted into a table in the compact CCS power model.

Similar to the compact CCS timing modeling syntax, compact CCS power modeling syntax uses the base_curves group to describe normalized base curves and the compact_lut_template group as the compact current waveform template. However, the compact_lut_template group attributes are extended from three dimensions to four dimensions when CCS power models need two total_output_net_capacitance attributes.

The compact CCS power modeling syntax is as follows:

```plaintext
library (my_library) {
  base_curves(bc_name) {
    base_curve_type : enum (ccs_half_curve, ccs_timing_half_curve);
    curve_x ("float, ..., float");
    curve_y ("integer, float, ..., float"); /* base curve #1 */
    curve_y ("integer, float, ..., float"); /* base curve #2 */
    ...  
    curve_y ("integer, float, ..., float"); /* base curve #n */
  }
  compact_lut_template (template_name) {
    base_curves_group : bc_name;
    variable_1 : input_net_transition | total_output_net_capacitance;
    variable_2 : input_net_transition | total_output_net_capacitance;
    variable_3 : input_net_transition | total_output_net_capacitance;
    variable_4 : curve_parameters;
    index_1 ("float, ..., float");
    index_2 ("float, ..., float");
    index_3 ("float, ..., float");
    index_4 ("string, ..., string");
  }
  ...  
  cell(cell_name) {
    dynamic_current() {
      switching_group() {
        pg_current(pg_pin_name) {
          compact_ccs_power (template_name) {
            base_curves_group : bc_name;
            index_output : pin_name;
            index_1 ("float, ..., float");
            index_2 ("float, ..., float");
            index_3 ("float, ..., float");
            index_4 ("string, ..., string");
            values ("float | integer, ..., float | integer");
          } /* end of compact_ccs_power */
          ...  
        } /* end of pg_current */
        ...  
      } /* end of switching_group */
      ...  
    } /* end of dynamic_current */
  } /* end of cell */
```

Chapter 14: Composite Current Source Power Modeling
Compact CCS Power Modeling
Library-Level Groups and Attributes

This section describes library-level groups and attributes used for compact CCS power modeling.

**base_curves Group**

The *base_curves* group contains the detailed description of normalized base curves. The *base_curves* group has the following attributes:

**base_curve_type Complex Attribute**

The *base_curve_type* attribute specifies the type of base curve. The *ccs_half_curve* value allows you to model compact CCS power and compact CCS timing data within the same *base_curves* group. You must specify *ccs_half_curve* before specifying *ccs_timing_half_curve*.

**curve_x Complex Attribute**

The data array contains the x-axis values of the normalized base curve. Only one *curve_x* value is allowed for each *base_curves* group.

For a *ccs_timing_half_curve* base curve, the *curve_x* value must be between 0 and 1 and increase monotonically.

**curve_y Complex Attribute**

Each base curve consists of one *curve_x* and one *curve_y* attributes. You should define the *curve_x* base curve before *curve_y* for better clarity and easier implementation. The valid region for *curve_y* is [-30, 30] for compact CCS power.

There are two data sections in the *curve_y* complex attribute:

- The *curve_id* integer specifies the identifier of the base curve.
- The data array specifies the y-axis values of the normalized base curve.
**compact_lut_template Group**

The `compact_lut_template` group is a lookup table template used for compact CCS timing and power modeling.

The following requirements must be met for compact CCS power modeling:

- The last `variable_*` value must be `curve_parameters`.
- The `input_net_transition` and `total_output_net_capacity` values are available for all `variable_*` attributes except the last `variable_*`. They can be placed in any order except last.

The following conditions describe the `pg_current_template` group:

- There can be zero or one `input_net_transition` variable.
- There can be zero, one, or two `total_output_net_capacity` variables.
- The element type for all `index_*` values except the last one is a list of floating-point numbers.
- The element type for the last `index_*` value is a string.
- The only valid value for `index_*`, when it is last and when it is specified with `curve_parameters` as the last `variable_*`, is `init_time`, `init_current`, `bc_id1`, `point_time1`, `point_current1`, `bc_id2`, `[point_time2, point_current2, bc_id3, ...]`, `end_time`, `end_current`.

The valid value in the last index is the pattern that all curve parameter series should follow. It is a pattern rather than a specified series because the table varies in size. Curve parameters define how to describe a current waveform. There should be at least two segments. The reference time for each current waveform is always zero. The negative time values, such as the values with corresponding parameters `init_time`, `point_time` and `end_time`, are permitted.

Note that this index is only for clarity. It is not used to determine the curve parameters. Curve parameters can be uniquely determined by the size of the values. A valid size can be represented as `(8+3i)`, where `i` is an integer and `i>=0`. The current waveform has `(i+2)` segments.
Cell-Level Groups and Attributes
This section describes cell-level groups and attributes used for compact CCS power modeling.

**compact_ccs_power Group**

The `compact_ccs_power` group contains a detailed description for compact CCS power data. The `compact_ccs_power` group includes the following optional attributes: `base_curves_group`, `index_1`, `index_2`, `index_3` and `index_4`. The description for these attributes in the `compact_ccs_power` group is the same as in the `compact_lut_template` group. However, the attributes have a higher priority in the `compact_ccs_power` group. For more information, see "compact_lut_template Group" on page 14-22.

The `index_output` attribute is also optional. It is used only on cross type tables.

**values Attribute**

The `values` attribute is required in the `compact_ccs_power` group. The data within the quotation marks (""), or *line*, represent the current waveform for one index combination. Each value is determined by the corresponding curve parameter. In the following line,

"t0, c0, 1, t1, c1, 2, t2, c2, 3, t3, c3, 4, t4, c4"

the size is 14 = 8+3*2. Therefore, the curve parameters are as follows:

"init_time, init_current, bc_id1, point_time1, point_current1, bc_id2, point_time2, point_current2, bc_id3, point_time3, point_current3, bc_id4, end_time, end_current"

The elements in the `values` attribute are floating-point numbers for time and current and integers for the base curve ID. The number of current waveform segments can be different for each slew and load combination, which means that each line size can be different. As a result, Liberty syntax supports tables with varying sizes, as shown:

```
compact_ccs_power (template_name) {
...  
  index_1("0.1, 0.2"); /* input_net_transition */
  index_2("1.0, 2.0"); /* total_output_net_capacitance */
  index_3("init_time, init_current, bc_id1, point_time1, point_current1, bc_id2, point_time2, point_current2, bc_id3, point_time3, point_current3, bc_id4, end_time, end_current"); /* curve_parameters */
  values ("t0, c0, 1, t1, c1, 2, t2, c2, 3, t3, c3, 4, t4, c4", /* segment=4 */
    "t0, c0, 1, t1, c1, 2, t2, c2", /* segment=2 */
    "t0, c0, 1, t1, c1, 2, t2, c2, 3, t3, c3", /* segment=3 */
    "t0, c0, 1, t1, c1, 2, t2, c2, 3, t3, c3"); /* segment=3 */
}
```
Composite Current Source Dynamic Power Examples

This section provides the following CCS dynamic power examples:

- **Design Cell With a Single Output Example**
- **Dense Table With Two Output Pins Example**
- **Cross Type With More Than One Output Pin Example**
- **Diagonal Type With More Than One Output Pin Example**

### Design Cell With a Single Output Example

```
pg_current_template ( CCS_power_1 ) {
  variable_1 : input_net_transition ;
  variable_2 : total_output_net_capacitance ;
  variable_3 : time ;
}

cell (example) {
  dynamic_current() {
    when: D;
    related_inputs : CP;
    related_outputs : Q;
    switching_group ( ) {
      input_switching_condition(rise);
      output_switching_condition(rise);
      pg_current (VDD ) {
        vector ( CCS_power_1 ) {
          reference_time : 0.01;
          index_1 ( 0.01 )
          index_2 ( 1.0 )
          index_3 ( 0.000, 0.0873, 0.135, 0.764)
          values ( 0.002, 0.009, 0.134, 0.546)
        }
      }
    }
  }
}
```

### Dense Table With Two Output Pins Example

```
pg_current_template ( CCS_power_1 ) {
  variable_1 : input_net_transition ;
  variable_2 : total_output_net_capacitance ;
  variable_3 : total_output_net_capacitance ;
  variable_4 : time ;
```
cell ( example ) {
  dynamic_current() {
    related_inputs : "A" ;
    related_outputs : "Z" ;
    typical_capacitances(0.04);
    switching_group() {
      input_switching_condition(rise);
      output_switching_condition(rise);
      pg_current(VDD) {
        vector(ccsp_switching_load_time) {
          reference_time : 0.0015 ;
          index_1("0.0019");
          index_2("0.001");
          index_3("0, 0.006, 0.03, 0.07, 0.09, 0.1, 0.2, 0.3, 0.4,
                         0.5");
          values("5e-06, 0.001, 0.02, 0.03, 0.05, 0.08, 0.09, 0.04,
                             0.009, 5.0e-06");
        }
      }
    }
  }
}

Cross Type With More Than One Output Pin Example

pg_current_template ( CCS_power_1 ) {
  variable_1 : input_net_transition ;
  variable_2 : total_output_net_capacitance ;
  variable_3 : time ;
}

cell ( example )

dynamic_current() {
  when: D;
  related_inputs : CP;
  related_outputs : Q QN QN1 QN2;
  typical_capacitances(10.0 10.0 10.0 10.0);
  switching_group () {
    input_switching_condition(rise);
    output_switching_condition(rise, fall, fall, fall);
    pg_current (VSS) {
      vector ( CCS_power_1 ) {
        index_output : Q;
        reference_time : 0.01;
        index_1 (~0.01 )
        index_2 ( 5.0)
        index_3 ( 0.000, 0.0873, 0.135, 0.764)
        values ( 0.002, 0.009, 0.134, 0.546)
Diagonal Type With More Than One Output Pin Example

pg_current_template ( CCS_power_1 ) {
  variable_1 : input_net_transition ;
  variable_2 : total_output_net_capacitance ;
  variable_3 : time ;
}

cell ( example )
  dynamic_current() {
    when: D ;
    related_inputs : CP;
    related_outputs : Q QN QN1 QN2;
    switching_group ( ) {
      input_switching_condition(rise);
      output_switching_condition(rise, fall, fall, fall);
    }
  }
}

pgCurrent (VSS) {
  vector ( CCS_power_1 ) {
    reference_time : 0.01;
    index_1 ( 0.01 )
    index_2 ( 5.0 )
    index_3 ( 0.000, 0.0873, 0.135, 0.764)
    values ( 0.002, 0.009, 0.134, 0.546)
  }
}

}
On-chip variation (OCV) causes variation in the timing performance of each transistor in a design.

In advanced OCV models, these variations are modeled using derating factors based on the path depth and path distance.

In parametric OCV models, the variations are modeled using random variation and path distance of cells.

The OCV model definitions are covered in the following sections:

• OCV Model Overview
• Advanced OCV Modeling
• LVF Models For Cell Delay, Transition, and Constraint
• LVF Retain Arc Models
• LVF Moment-Based Models For Ultra-Low Voltage Designs
OCV Model Overview

In Table 15-1, path depth is the position of a cell in a delay path, and path distance is the length of the path.

Advanced OCV Modeling

In an advanced OCV model, the derating values are based on the path depth and path distance.

The modeling syntax is:

```
library (name) {
  ocv_table_template(ocv_template_name) {
    variable_1: path_depth|path_distance;
    variable_2: path_depth|path_distance;
    index_1 ("float,..., float");
    index_2 ("float,..., float");
  }
  ocv_derate(ocv_derate_group_name) {
    ocv_derate_factors(ocv_template_name) {
      rf_type: rise|fall|rise_and_fall;
      derate_type: early|late;
      path_type: clock|data|clock_and_data;
      index_1 ("float,..., float");
      index_2 ("float,..., float");
      values ( "float,..., float", \ 
        ..., \ 
        "float,..., float");
    }
  }
  ...
  }
  default_ocv_derate_group: ocv_derate_group_name;
  ocv_arc_depth: float;
  ...
}
```
Library-Level Groups and Attributes

This section describes library-level groups and attributes for advanced OCV modeling that also apply to cells and timing arcs where specified.

ocv_table_template Group

The `ocv_table_template` group defines the template for an `ocv_derate_factors` group defined within the `ocv_derate` group.

The template definition includes two one-dimensional variables, `variable_1` and `variable_2`. These variables are used as indexes in the `ocv_derate_factors` group. You must specify `variable_1`. Specifying `variable_2` is optional.

Both `variable_1` and `variable_2` can have one of the following values:

- `path_depth`: The number of cells in a delay path.
- `path_distance`: The physical distance spanned by the path. Use the `distance_unit` attribute in the library to specify the path distance unit.

You can specify the following types of lookup tables within the `ocv_table_template` group:

- One-dimensional tables with the index consisting of `path_depth` or `path_distance` for a table
- Two-dimensional tables with one index consisting of `path_depth`, and the other index consisting of `path_distance`

This means that for a two-dimensional table, the values of `variable_1` and `variable_2` are different.
**ocv_derate Group**

The `ocv_derate` group contains a set of `ocv_derate_factors` groups. The `ocv_derate` group specifies a set of lookup tables with OCV derating factors for timing.

You must specify at least one `ocv_derate_factors` group within an `ocv_derate` group.

You can specify the `ocv_derate` group in the `library` and `cell` groups. To define multiple `ocv_derate` groups in a library, use different group names. If multiple `ocv_derate` groups have the same name, the group that is last specified overrides the previous ones.

**ocv_derate_factors Group**

The `ocv_derate_factors` group specifies a lookup table of the OCV derating factors. The lookup table can be one-dimensional, two-dimensional, or scalar. For a one-dimensional lookup table, the index consists of `path_depth` or `path_distance` values. For a two-dimensional lookup table, the indexes consist of the `path_depth` and `path_distance` values. Use the scalar to specify a single derating factor irrespective of the path depth and path distance.

Use the following attributes to define specific OCV derating factors in the `ocv_derate_factors` lookup table:

- **rf_type**
  The `rf_type` attribute defines the type of delay specified in the `ocv_derate_factors` lookup table. Valid values are `rise`, `fall`, and `rise_and_fall`.

- **derate_type**
  The `derate_type` attribute defines the type of arrival time specified in the `ocv_derate_factors` lookup table. The valid values are `early` and `late`.

- **path_type**
  The `path_type` attribute defines the type of path specified in the `ocv_derate_factors` group. The valid values are `clock`, `data`, and `clock_and_data`.

These attributes do not have defaults. You must specify these attributes in the `ocv_derate_factors` group.

**Applying OCV Derating Factors to Library Cells**

To apply the set of derating factors defined within an `ocv_derate` group to a cell, use the cell-level `ocv_derate_group` attribute to specify the name of the `ocv_derate` group.

To apply the same set of derating factors to multiple cells in a library, specify the `ocv_derate` group at the library-level and define the name of the `ocv_derate` group with the `ocv_derate_group` attribute in the corresponding `cell` groups.
The set of derating factors defined within a cell-level `ocv_derate` group can be applied only to the particular cell.

If you do not specify the `ocv_derate_group` attribute, the default `ocv_derate` group defined at the library level applies to the cell.

**default_ocv_derate_group Attribute**

The `default_ocv_derate_group` attribute specifies the name of the default `ocv_derate` group for a library. The default `ocv_derate` group applies to the cell groups where the `ocv_derate` group name is not defined using the cell-level `ocv_derate_group` attribute.

**ocv_arc_depth Attribute**

The optional `ocv_arc_depth` attribute specifies the effective logic depth of a cell or a timing arc. The default is 1.0.

The path depth is used to determine the OCV derating factors from the lookup tables in the `ocv_derate_factors` group.

You can define the `ocv_arc_depth` attribute in the `library`, `cell`, or `timing` groups. If you define this attribute in different groups, the attribute value in the `timing` group overrides the value defined in the `cell` group, and the value defined in the `cell` group overrides the value defined in the `library` group.

---

**Cell-Level Attribute**

This section describes a cell-level attribute for advanced OCV modeling.

**ocv_derate_group Attribute**

The `ocv_derate_group` attribute specifies the name of the `ocv_derate` group that applies to a cell.

---

**Advanced OCV Library Example**

*Example 15-1  A Typical Advanced OCV Library*

```plaintext
library (OCV_lib) {  
    ocv_table_template(2D_ocv_template) {  
        variable_1: path_depth;  
        variable_2: path_distance;  
        index_1 ("1,2,3");  
        index_2 ("1,2,3");  
    }  
}  
```
ocv_table_template(1D_ocv_template1) {
    variable_1: path_depth;
    index_1 ("1,2,3");
}

ocv_table_template(1D_ocv_template2) {
    variable_1: path_distance;
    index_1 ("1,2,3");
}

ocv_derate(advanced_ocv) {
    ocv_derate_factors(scalar) {
        rf_type: rise_and_fall;
        derate_type: early;
        path_type: clock_and_data;
        values ( "0.100");
    }
    ocv_derate_factors(scalar) {
        rf_type: rise_and_fall;
        derate_type: late;
        path_type: clock_and_data;
        values ( "0.200");
    }
}

ocv_arc_depth: 1.0;
default_ocv_derate_group: advanced_ocv;

cell (cell1) {
    ocv_arc_depth: 2.0;
    ocv_derate_group: aocv1;
    ocv_derate(aocv1) {
        ocv_derate_factors (2D_ocv_template) {
            rf_type: rise_and_fall;
            derate_type: early;
            path_type: clock_and_data;
            index_1 ("1,2,3");
            index_2 ("4, 5");
            values ( "0.1, 0.2, 0.3", 
                    "0.2, 0.3, 0.4");
        }
        ocv_derate_factors (2D_ocv_template) {
            rf_type: rise_and_fall;
            derate_type: late;
            path_type: clock_and_data;
            index_1 ("1,2,3");
            index_2 ("100, 200");
            values ( "0.1, 0.2, 0.3", 
                    "0.2, 0.3, 0.4");
        }
    }
    ocv_derate(aocv2) {
        ocv_derate_factors(scalar) {
            rf_type: rise_and_fall;
            derate_type: late;
            path_type: clock_and_data;
            values ( "0.900");
        }
    }
}

Chapter 15: On-Chip Variation (OCV) Modeling
Advanced OCV Modeling 15-6
LVF Models For Cell Delay, Transition, and Constraint

In a parametric OCV (POCV) model, the random variation is specific to a cell or a timing arc, unlike an advanced OCV model where a specific derating factor applies to multiple cells or an entire library. The Liberty variation format (LVF) is used to represent the parametric OCV (POCV) library data, such as the slew-load table per delay timing arc. The following Liberty variation format (LVF) syntax consists of groups for sigma variation cell delay, output transition or slew, and constraint tables that are load and input-slew dependent. The variation values are used during parametric OCV analysis.

Syntax

The parametric OCV Liberty variation format (LVF) modeling syntax for cell delay, transition, and constraint is:

```plaintext
library (name) {
  lu_table_template(lu_template_name) {
```
variable_1: input_net_transition;
variable_2: total_output_net_capacitance;
variable_3: related_out_total_output_net_capacitance;
index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
}

lu_table_template(constraint_lu_template_name) {
  variable_1: related_pin_transition;
  variable_2: constrained_pin_transition;
  variable_3: related_out_total_output_net_capacitance | \ 
    related_out_output_net_length | \ 
    related_out_output_net_wire_cap | \ 
    related_out_output_net_pin_cap;
  index_1 ("float, ..., float");
  index_2 ("float, ..., float");
  index_3 ("float, ..., float");
}

ocv_table_template(ocv_template_name) {
  variable_1: path_distance;
  index_1 ["float, ..., float"];}

ocv_derate(ocv_derate_group_name) {
  ocv_derate_factors(ocv_template_name) {
    rf_type: rise | fall | rise_and_fall;
    derate_type: early | late;
    path_type: clock | data | clock_and_data;
    index_1 ("float, ..., float");
    values ("float, ..., float");
  }
  ...
}
default_ocv_derate_distance_group: ocv_derate_group_name;
...
cell (cell_name) {
  ocv_derate_distance_group: ocv_derate_group_name;
  ...
  pin | bus | bundle (name) {
    direction: input | output;
    timing() {
      ...
      ocv_sigma_cell_rise(lu_template_name){
        sigma_type: early | late | early_and_late;
        index_1 ("float, ..., float");
        index_2 ("float, ..., float");
        values ("float, ..., float", \ 
          ..., \ 
          "float, ..., float");
      }
      ocv_sigma_cell_fall(lu_template_name){
        sigma_type: early | late | early_and_late;
        index_1 ("float, ..., float");
      }
    }
  }
}
index_2 ("float, ..., float");
values ( "float, ..., float", \\
        "float, ..., float");
}

ocv_sigma_rise_transition(\template_name) {
    sigma_type: early | late | early_and_late;
    index_1 ("float, ..., float");
    index_2 ("float, ..., float");
    index_3 ("float, ..., float");
    values ( "float, ..., float", \\
            "float, ..., float");
}

ocv_sigma_fall_transition(\template_name) {
    sigma_type: early | late | early_and_late;
    index_1 ("float, ..., float");
    index_2 ("float, ..., float");
    index_3 ("float, ..., float");
    values ( "float, ..., float", \\
            "float, ..., float");
}

...
Note:
Do not specify the sigma_type attribute in the ocv_sigma_rise_constraint and ocv_sigma_fall_constraint groups.

Library-Level Groups and Attributes
This section describes library-level groups and attributes for parametric OCV modeling that also apply to cells and timing arcs where specified.

**lu_table_template Group**
The lu_table_template group defines the template for the following groups specified in the timing group:
- ocv_sigma_cell_rise
- ocv_sigma_cell_fall
- ocv_sigma_rise_transition
- ocv_sigma_fall_transition
- ocv_sigma_rise_constraint
- ocv_sigma_fall_constraint

For the ocv_sigma_cell_rise, ocv_sigma_cell_fall, ocv_sigma_rise_transition, and ocv_sigma_fall_transition groups, the template definition includes the variable_1, variable_2, variable_3, index_1, index_2, and index_3 attributes. The variable_1, variable_2 and variable_3 attributes specify the index variables for the lookup tables (LUTs) in these groups. The index_1, index_2, and index_3 attributes specify the numerical values of the variables.

The values of variable_1, variable_2, and variable_3 attributes are input_net_transition, total_output_net_capacitance, and related_out_total_output_net_capacitance.

You can specify the following types of lookup tables within these groups:
- One-dimensional tables with a single index, such as input_net_transition
- Two-dimensional tables with the indexes, input_net_transition and total_output_net_capacitance
Three-dimensional tables with the indexes, `input_net_transition`, `total_output_net_capacitance`, and `related_out_total_output_net_capacitance`

A scalar that has a single variation value irrespective of the `input_net_transition`, `total_output_net_capacitance`, and `related_out_total_output_net_capacitance` values

For the `ocv_sigma_rise_constraint` and `ocv_sigma_fall_constraint` groups, the template definition includes the `variable_1`, `variable_2`, `variable_3`, `index_1`, `index_2`, and `index_3` attributes. The `variable_1`, `variable_2` and `variable_3` attributes specify the index variables for the lookup tables of the `ocv_sigma_rise_constraint` and `ocv_sigma_fall_constraint` groups. The `index_1`, `index_2`, and `index_3` attributes specify the numerical values of these variables.

The values of `variable_1` and `variable_2` are `related_pin_transition` and `constrained_pin_transition`, respectively. The values of `variable_3` can be `related_out_total_output_net_capacitance`, `related_out_output_net_length`, `related_out_net_wire_cap`, or `related_out_output_net_pin_cap`. You can specify the following types of lookup tables within these groups:

- One-dimensional tables with a single index, such as `related_pin_transition`
- Two-dimensional tables with the indexes, `related_pin_transition` and `constrained_pin_transition`
- Three-dimensional tables with the indexes, `related_pin_transition`, `constrained_pin_transition`, and a valid value of `variable_3`
- A scalar that has a single variation value irrespective of the `related_pin_transition`, `constrained_pin_transition`, and `variable_3` values

**ocv_table_template Group**

The `ocv_table_template` group defines the template for an `ocv_derate_factors` group defined within the `ocv_derate` group.

The template definition includes a one-dimensional variable, `variable_1`, that is used as an index in the `ocv_derate_factors` group. Valid value of `variable_1` is `path_distance` and represents the physical distance spanned by the path. Use the `distance_unit` attribute in the library to specify the path distance unit.

**ocv_derate Group**

The `ocv_derate` group contains a set of `ocv_derate_factors` groups. The `ocv_derate` group specifies a set of lookup tables with OCV derating factors for timing.

You must specify at least one `ocv_derate_factors` group within an `ocv_derate` group.
You can specify the `ocv_derate` group in the library and cell groups. To define multiple `ocv_derate` groups in a library, use different group names. If multiple `ocv_derate` groups have the same name, the group that is last-specified overrides the previous ones.

**ocv_derate_factors** Group

The `ocv_derate_factors` group specifies a lookup table of the OCV derating factors. The lookup table can be one-dimensional or scalar. For a one-dimensional lookup table, the index consists of `path_distance` values. Use the scalar to specify a single derating factor irrespective of the path distance.

Use the following attributes to define specific OCV derating factors in the `ocv_derate_factors` lookup table:

- `rf_type`
  - The `rf_type` attribute defines the type of delay specified in the `ocv_derate_factors` lookup table. Valid values are `rise`, `fall`, and `rise_and_fall`.

- `derate_type`
  - The `derate_type` attribute defines the type of arrival time specified in the `ocv_derate_factors` lookup table. The valid values are `early` and `late`.

- `path_type`
  - The `path_type` attribute defines the type of path specified in the `ocv_derate_factors` group. The valid values are `clock`, `data`, and `clock_and_data`.

These attributes do not have defaults. You must specify these attributes in the `ocv_derate_factors` group.

**Applying Parametric OCV Derating Factors to Library Cells**

To apply the set of derating factors defined within an `ocv_derate` group to a cell, use the cell-level `ocv_derate_distance_group` attribute to specify the name of the `ocv_derate` group.

To apply the same set of derating factors to multiple cells in a library, specify the `ocv_derate` group at the library-level and define the name of the `ocv_derate` group with the `ocv_derate_distance_group` attribute in the corresponding cell groups.

The set of derating factors defined within a cell-level `ocv_derate` group can be applied only to the particular cell.

If you do not specify the `ocv_derate_distance_group` attribute, the default `ocv_derate` group defined at the library level applies to the cell.
**default_ocv_derate_distance_group Attribute**

The `default_ocv_derate_distance_group` attribute specifies the name of the default `ocv_derate` group for a library. The default `ocv_derate` group applies to the cell groups where the `ocv_derate` group name is not defined using the cell-level `ocv_derate_distance_group` attribute.

---

**Cell-Level Attribute**

This section describes a cell-level attribute for parametric OCV modeling.

**ocv_derate_distance_group Attribute**

The `ocv_derate_distance_group` attribute specifies the name of the `ocv_derate` group that applies to a cell.

---

**Timing Arc Level Groups and Attributes**

This section describes the timing arc-level groups for modeling delay variations. Each of the groups specify a lookup table. The lookup table can be one-dimensional, two-dimensional, three-dimensional, or scalar.

Define the template of the lookup table in the library-level `lu_table_template` group. To use the template, specify the name of the `lu_table_template` group as the arc-level group name.

**ocv_sigma_cell_rise Group**

The `ocv_sigma_cell_rise` group specifies a lookup table for the rise delay variation. In the lookup table, each absolute rise-delay variation offset from the corresponding nominal rise delay is specified at one sigma (σ), where sigma is the standard deviation of the rise delay distribution.

**Note:**

The nominal rise delay value is specified by the `cell_rise` group within the `timing` group. For more information, see Chapter 7, “Timing Arcs”.

During parametric OCV analysis, the delay variation and the nominal rise delay are used to calculate the rise delay at one sigma (σ):

\[
\text{rise delay}_{\pm \sigma} = \text{nominal rise delay} \pm \text{ocv_sigma_cell_rise value}
\]
**ocv_sigma_cell_fall Group**

The `ocv_sigma_cell_fall` group specifies a lookup table for the fall delay variation. In the lookup table, each absolute fall-delay variation offset from the corresponding nominal fall delay is specified at one sigma ($\sigma$), where sigma is the standard deviation of the fall delay distribution.

**Note:**

The nominal fall delay value is specified by the `cell_fall` group within the `timing` group. For more information, Chapter 7, “Timing Arcs”.

During OCV analysis, the delay variation and the nominal fall delay are used to calculate the fall delay at one sigma ($\sigma$):

\[
\text{fall delay}_{(\pm \sigma)} = \text{nominal fall delay} \pm \text{ocv_sigma_cell_fall value}
\]

**ocv_sigma_rise_transition Group**

The `ocv_sigma_rise_transition` group specifies a lookup table of the rise transition variation values. In the lookup table, each absolute rise-transition variation offset from the corresponding nominal rise transition is specified at one sigma ($\sigma$), where sigma is the standard deviation of the rise transition distribution.

**Note:**

The nominal rise transition value is specified by the `rise_transition` group within the `timing` group. For more information, see Chapter 7, “Timing Arcs”.

During parametric OCV analysis, the rise transition variation and the nominal rise transition are used to calculate the rise transition at one sigma ($\sigma$):

\[
\text{rise transition}_{(\pm \sigma)} = \text{nominal rise transition} \pm \text{ocv_sigma_rise_transition value}
\]

**ocv_sigma_fall_transition Group**

The `ocv_sigma_fall_transition` group specifies a lookup table for the fall transition variation. In the lookup table, each absolute fall-transition variation offset from the nominal fall transition is specified at one sigma ($\sigma$), where sigma is the standard deviation of the fall transition distribution.

**Note:**

The nominal fall transition value is specified by the `fall_transition` group within the `timing` group. For more information, Chapter 7, “Timing Arcs”.

During parametric OCV analysis, the transition variation and the nominal fall transition are used to calculate the fall delay at one sigma ($\sigma$):

\[
\text{fall transition}_{(\pm \sigma)} = \text{nominal fall transition} \pm \text{ocv_sigma_fall_transition value}
\]
**sigma_type Attribute**

The optional *sigma_type* attribute defines the type of arrival time specified in the `ocv_sigma_cell_rise`, `ocv_sigma_cell_fall`, `ocv_sigma_rise_transition`, and `ocv_sigma_fall_transition` lookup tables. The values are *early*, *late*, and *early_and_late*. The default is *early_and_late*.

During parametric OCV analysis, when you specify the *sigma_type* attribute in these groups, the following values at ±σ are calculated as:

- **Rise delays**
  - \( \tau^{+} \) = nominal cell rise + \( ocv_{sigma\_cell\_rise\_late} \)
  - \( \tau^{-} \) = nominal cell rise - \( ocv_{sigma\_cell\_rise\_early} \)

- **Fall delays**
  - \( \tau^{+} \) = nominal cell fall + \( ocv_{sigma\_cell\_fall\_late} \)
  - \( \tau^{-} \) = nominal cell fall - \( ocv_{sigma\_cell\_fall\_early} \)

- **Rise transition**
  - \( \tau^{+} \) = nominal rise transition + \( ocv_{sigma\_rise\_transition\_late} \)
  - \( \tau^{-} \) = nominal rise transition - \( ocv_{sigma\_rise\_transition\_early} \)

- **Fall transition**
  - \( \tau^{+} \) = nominal fall transition + \( ocv_{sigma\_fall\_transition\_late} \)
  - \( \tau^{-} \) = nominal fall transition - \( ocv_{sigma\_fall\_transition\_early} \)

**ocv_sigma_rise_constraint Group**

The `ocv_sigma_rise_constraint` group specifies a lookup table of the rise constraint variation values. In the lookup table, each absolute rise-constraint variation offset from the nominal rise constraint is specified at one sigma (σ), where sigma is the standard deviation of the rise constraint distribution.

**Note:**

The nominal rise constraint value is specified by the `rise_constraint` group within the `timing` group. For more information, see Chapter 7, “Timing Arcs”.

During OCV analysis, the rise constraint variation and the nominal rise constraint are used to calculate the rise constraint at one sigma (σ):

\[ \tau^{\pm} = nominal\ rise\ constraint \pm ocv_{sigma\_rise\_constraint\ value} \]

**ocv_sigma_fall_constraint Group**

The `ocv_sigma_fall_constraint` group specifies a lookup table for the fall constraint variation. In the lookup table, each absolute fall-constraint variation offset from the nominal fall constraint is specified at one sigma (σ), where sigma is the standard deviation of the fall constraint distribution.

**Note:**

The nominal fall constraint value is specified by the `fall_constraint` group within the `timing` group. For more information, Chapter 7, “Timing Arcs”.


During parametric OCV (POCV) analysis, the delay variation and the nominal fall delay are used to calculate the fall delay at one sigma ($\sigma$):

$$\text{fall constraint}_{(\pm \sigma)} = \text{fall constraint} \pm \text{ocv sigma fall constraint value}$$

### Parametric OCV Library Example

**Example 15-2**  A Library Model for OCV Delays, Transitions, and Constraints

```plaintext
library (lib1) {
  lu_table_template(2D_lu_template) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    index_1 ("1,2,3");
    index_2 ("1,2,3");
  }
  lu_table_template(3D_lu_template) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: related_out_total_output_net_capacitance;
    index_1 ("1,2,3");
    index_2 ("1,2,3");
    index_3 ("1,2,3");
  }
  lu_table_template(2D_constraint_lu_template) {
    variable_1: related_pin_transition;
    variable_2: constrained_pin_transition;
    index_1 ("1,2,3");
    index_2 ("1,2,3");
  }
  ocv_table_template(1D_ocv_template2) {
    variable_1: path_distance;
    index_1 ("1,2,3");
  }
  ocv_derate(location_based_ocv) {
    ocv_derate_factors(1D_ocv_template2) {
      rf_type: rise_and_fall;
      derate_type: early;
      path_type: clock_and_data;
      index_1 ("1, 2");
      values ( "5.0, 6.0");
    }
    ocv_derate(locv1) {
      ocv_derate_factors(1D_ocv_template2) {
        rf_type: rise_and_fall;
        derate_type: early;
        path_type: clock_and_data;
        index_1 ("1, 2");
        values ( "5.0, 6.0");
      }
    }
  }
}
```
default_ocv_derate_distance_group: location_based_ocv;

cell (cell1) {
    ocv_derate_distance_group: locv1;
    ...
}

pin (pin1) {
    direction: output;
    timing() {
        ...
        related_pin: "rpin1";
        timing_type: rising_edge;
        cell_rise( 2D_lu_template ){
            index_1 ( "0.1, 0.2, 0.3");
            index_2 ( "0.4, 0.5, 0.6");
            values ( "0.1, 0.2, 0.3",
                    "0.2, 0.3, 0.4",
                    "0.3, 0.4, 0.5");
        }
        cell_fall( scalar ){
            values ("0.100");
        }
        rise_transition( 2D_lu_template ){
            index_1 ( "0.1, 0.2, 0.3");
            index_2 ( "0.4, 0.4, 0.5");
            values ( "0.1, 0.2, 0.3",
                    "0.3, 0.4, 0.5",
                    "0.5, 0.6, 0.7");
        }
        fall_transition( scalar ){
            values ("0.100");
        }
        ocv_sigma_cell_rise( 2D_lu_template ){
            sigma_type: early;
            index_1 ( "0.3, 0.4, 0.5");
            index_2 ( "0.1, 0.2, 0.3");
            values ( "0.1, 0.2, 0.3",
                    "0.2, 0.3, 0.4",
                    "0.3, 0.4, 0.5");
        }
        ocv_sigma_cell_rise( 2D_lu_template ){
            sigma_type: late;
            index_1 ( "0.3, 0.4, 0.5");
            index_2 ( "0.1, 0.2, 0.3");
            values ( "0.1, 0.2, 0.3",
                    "0.2, 0.3, 0.4",
                    "0.3, 0.4, 0.5");
        }
        ocv_sigma_cell_fall( scalar ){
            sigma_type: early;
            values ("0.1");
        }
        ocv_sigma_cell_fall( scalar ){
            sigma_type: late;
            values ("0.1");
ocv_sigma_rise_transition( 3D_lu_template ){
    sigma_type: early;
    index_1 ( "0.1, 0.2, 0.3");
    index_2 ( "0.4, 0.5, 0.6");
    index_3 ( "0.7, 0.8");
    values ("0.1, 0.2, 0.3",
            "0.2, 0.3, 0.4",
            "0.3, 0.4, 0.5",
            "0.4, 0.5, 0.6",
            "0.5, 0.6, 0.7",
            "0.6, 0.7, 0.8");
}

ocv_sigma_rise_transition( 3D_lu_template ){
    sigma_type: late;
    index_1 ( "0.1, 0.2, 0.3");
    index_2 ( "0.4, 0.5, 0.6");
    index_3 ( "0.7, 0.8");
    values ("0.1, 0.2, 0.3",
            "0.2, 0.3, 0.4",
            "0.3, 0.4, 0.5",
            "0.4, 0.5, 0.6",
            "0.5, 0.6, 0.7",
            "0.6, 0.7, 0.8");
}

ocv_sigma_fall_transition( scalar ){
    sigma_type: early_and_late;
    values ( "0.200");
}

pin (pin2) {
    direction: input | inout;
    timing() {
        ...
    } /* end of timing */
}

...
LVF Retain Arc Models

A retain arc models the time during which an output port retains its current logical value after a voltage rise or fall at a related input port.

For information about the nominal retain arc tables that store the retain arc delay and transition time, see Chapter 7, “Timing Arcs.”

The parametric OCV (POCV) models for retain arcs are stored in the Liberty variation format (LVF) in library source files.

Syntax

The following is the modeling syntax of LVF retain arc models.

```plaintext
library (name) {
  lu_table_template(lu_template_name) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: related_out_total_output_net_capacitance;
    index_1 (float,, float);
    index_2 (float,, float);
    index_3 (float,, float);
  }
  cell (name) {
    pin|bus|bundle(name) {
      direction: inout|output;
      timing() {
        ocv_sigma_retaining_rise (lu_template_name) {
          sigma_type: early|late|early_and_late;
          index_1 (float,, float);
          index_2 (float,, float);
          index_3 (float,, float);
          values (float,, float,
```
Library-Level Groups

This section describes library-level groups to model retain arc timing variation.

**lu_table_template Group**

The `lu_table_template` group defines the template for the following timing-level retain arc groups:

- `ocv_sigma_retaining_rise`
- `ocv_sigma_retaining_fall`
- `ocv_sigma_retain_rise_slew`
The template definition includes the `variable_1`, `variable_2`, `variable_3`, `index_1`, `index_2`, and `index_3` attributes. The `variable_1`, `variable_2`, and `variable_3` attributes specify the variables that are used as indexes in these groups. The `index_1`, `index_2`, and `index_3` attributes specify the numerical values of the variables. The values of `variable_1`, `variable_2`, and `variable_3` are input_net_transition, total_output_net_capacitance, and related_out_total_output_net_capacitance respectively. You can define a scalar, one-dimensional, two-dimensional, or three-dimensional lookup table using the `lu_table_template` group.

## Timing Arc Level Groups

This section describes timing-level groups to model retain arc timing delay and transition variation. These groups are lookup tables that can be scalar, one, two, or three-dimensional.

Define these groups in the `timing` group together with the nominal retain arc tables.

**ocv_sigma_retaining_rise and ocv_sigma_retaining_fall Groups**

The `ocv_sigma_retaining_rise` and `ocv_sigma_retaining_fall` groups are retain arc delay lookup tables that specify the absolute variation offset from the nominal retain arc table values at one sigma (σ), where sigma is the standard deviation of the delay distribution.

During parametric OCV analysis, the retain arc values at ±σ are calculated using the nominal and variation values as:

\[
\text{retaining_rise}(\pm \sigma) = \text{retaining_rise} \pm \text{ocv_sigma_retaining_rise} \\
\text{retaining_fall}(\pm \sigma) = \text{retaining_fall} \pm \text{ocv_sigma_retaining_fall}
\]

For more information about the nominal retain arc delay groups, see Chapter 7, “Timing Arcs”.

**ocv_sigma_retain_rise_slew and ocv_sigma_retain_fall_slew Groups**

The `ocv_sigma_retain_rise_slew` and `ocv_sigma_retain_fall_slew` groups are retain arc slew lookup tables that specify the absolute variation offset values from the nominal retain arc table values at one sigma (σ), where sigma is the standard deviation of the transition distribution.

During parametric OCV analysis, the retain arc values at ±σ are calculated using the variation values as:

\[
\text{retain_rise_slew}(\pm \sigma) = \text{retain_rise_slew} \pm
\]
For more information about the nominal retain arc slew groups, see Chapter 7, “Timing Arcs”.

**sigma_type Attribute**

The optional `sigma_type` attribute defines the type of arrival time specified in the `ocv_sigma_retaining_rise`, `ocv_sigma_retaining_fall`, `ocv_sigma_retain_rise_slew`, and `ocv_sigma_retain_fall_slew` lookup tables. The values are `early`, `late`, and `early_and_late`. The default is `early_and_late`.

During OCV analysis, if the `sigma_type` attribute is specified in these groups, the following values at $\pm \sigma$ are calculated as:

- \( \text{retaining\_rise}_{(+\sigma)} = \text{retaining\_rise} + \text{ocv\_sigma\_retaining\_rise}_{late} \)
- \( \text{retaining\_rise}_{(-\sigma)} = \text{retaining\_rise} - \text{ocv\_sigma\_retaining\_rise}_{early} \)
- \( \text{retaining\_fall}_{(+\sigma)} = \text{retaining\_fall} + \text{ocv\_sigma\_retaining\_fall}_{late} \)
- \( \text{retaining\_fall}_{(-\sigma)} = \text{retaining\_fall} - \text{ocv\_sigma\_retaining\_fall}_{early} \)

- \( \text{retain\_rise\_slew}_{(+\sigma)} = \text{retain\_rise\_slew} + \text{ocv\_sigma\_retain\_rise\_slew}_{late} \)
- \( \text{retain\_rise\_slew}_{(-\sigma)} = \text{retain\_rise\_slew} - \text{ocv\_sigma\_retain\_rise\_slew}_{early} \)
- \( \text{retain\_fall\_slew}_{(+\sigma)} = \text{retain\_fall\_slew} + \text{ocv\_sigma\_retain\_fall\_slew}_{late} \)
- \( \text{retain\_fall\_slew}_{(-\sigma)} = \text{retain\_fall\_slew} - \text{ocv\_sigma\_retain\_fall\_slew}_{early} \)

---

**Library Example**

The following example is a library with LVF retain arc models.

```latex
library (lib1) {
   lu_table_template(2D_lu_template) {
      variable_1: input_net_transition;
      variable_2: total_output_net_capacitance;
      index_1 (1,2,3);
      index_2 (1,2,3);
   }
   lu_table_template(1D_lu_template) {
      variable_1: total_output_net_capacitance;
      index_1 (1,2,3);
   }
   cell (cell1) {
      pin(pin1) {
         direction: output;
         timing() {
```
related_pin : "CLKIN";
timing_type : rising_edge;
cell_rise( 2D_lu_template ){
  ...
}
cell_fall( 2D_lu_template ){
  ...
}
rise_transition( 2D_lu_template ){
  ...
}
fall_transition( 2D_lu_template ){
  ...
}
retaining_rise( 2D_lu_template ){
  index_1 ( "0.10000, 0.20000, 0.30000" );
  index_2 ( "0.40000, 0.60000, 0.60000" );
  values ( "0.10000, 0.20000, 0.30000",
          "0.40000, 0.50000, 0.60000",
          "0.70000, 0.80000, 0.90000" );
}
retaining_fall( 1D_lu_template ){
  values ( "0.10000, 0.20000, 0.30000" );
}
retain_rise_slew( 2D_lu_template ){
  index_1 ( "0.10000, 0.20000, 0.30000" );
  index_2 ( "0.40000, 0.60000, 0.60000" );
  values ( "0.10000, 0.20000, 0.30000",
          "0.40000, 0.50000, 0.60000",
          "0.70000, 0.80000, 0.90000" );
}
retain_fall_slew( 1D_lu_template ){
  values ( "0.10000, 0.20000, 0.30000" );
}
ocv_sigma_retaining_rise( 2D_lu_template ){
  sigma_type: early;
  index_1 ( "0.10000, 0.20000, 0.30000" );
  index_2 ( "0.40000, 0.50000, 0.60000" );
  values ( "0.10000, 0.20000, 0.30000",
          "0.40000, 0.50000, 0.60000",
          "0.70000, 0.80000, 0.90000" );
}
ocv_sigma_retaining_rise( 2D_lu_template ){
  sigma_type: late;
  index_1 ( "0.10000, 0.20000, 0.30000" );
  index_2 ( "0.40000, 0.50000, 0.60000" );
  values ( "0.10000, 0.20000, 0.30000",
          "0.40000, 0.50000, 0.60000",
          "0.70000, 0.80000, 0.90000" );
}
ocv_sigma_retaining_fall( 1D_lu_template ){
  sigma_type: early;
  values ( "0.10000, 0.20000, 0.30000" );
}
Accurate models of ultra-low voltage libraries might require support for asymmetric, biased, or non-Gaussian distributions of timing variation. To capture the shape of the biased timing variation distribution, the tool supports moment-based Liberty variation format (LVF) OCV models.

```plaintext
ocv_sigma_retaining_fall (1D.lu_template)
  sigma_type: late;
  values ("0.10000, 0.20000, 0.30000");
}

ocv_sigma_retain_rise_slew(2D.lu_template)
  sigma_type: early_and_late;
  index_1 ("0.10000, 0.20000, 0.30000");
  index_2 ("0.40000, 0.50000, 0.60000");
  values ("0.10000, 0.20000, 0.30000", "0.40000, 0.50000, 0.60000", "0.70000, 0.80000, 0.90000");
}

ocv_sigma_retain_fall_slew (1D.lu_template)
  values ("0.10000, 0.20000, 0.30000");
}

/* end of timing group */
/* end of pin group */
/* end of cell group */
/* end of library group */
```
As shown in Figure 15-1, the moment-based LVF syntax is based on the mean, standard deviation, and skewness of the timing variation distribution.

**Mean**

Mean is the weighted average of the possible values, using their probabilities as weights or the continuous analog thereof. The mean of a random variable $X$ is defined, as shown:

$$E[X] = x_1p_1 + x_2p_2 + \ldots + x_kp_k$$

where $X$ can take values $x_1$ with probability $p_1$, $x_2$ with probability $p_2$, up to $x_k$ with probability $p_k$.

**Standard Deviation**

Standard deviation is a measure of the variation or dispersion of the distribution. It is defined as the square root of the variance, as shown:

$$\sqrt{E[(X-E[X])^2]}$$

**Skewness**

Skewness is a measure of the asymmetry of the distribution of a random variable $X$ about its mean. The definition follows the Pearson’s coefficient of skewness. It is defined, as shown:
\( \left( E[(X-E[X])^3]\right)^{1/3} \)

The following sections describe the moment-based LVF OCV models of random variation in cell delay, transition, and constraints.

**Syntax**

The following shows the moment-based LVF OCV modeling syntax.

```plaintext
library (lib_name) {
  lu_table_template (lu_template_name) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: related_out_total_output_net_capacitance;
    index_1 ("float,..., float");
    index_2 ("float,..., float");
    index_3 ("float,..., float");
  }
  ...
  cell (cell_name) {
    ...
    pin | bus | bundle (name) {
      direction: inout | output;
      timing() {
        ...
        /* delay nominal values */
        cell_rise (lu_template_name) {
          ...
        }
        cell_fall (lu_template_name) {
          ...
        }
        /* delay variation values */
        ocv_std_dev_cell_rise(lu_template_name){
          index_1 ("float,..., float");
          index_2 ("float,..., float");
          index_3 ("float,..., float");
          values ("float,..., float",
          ...
          "float,..., float");
        }
        ocv_std_dev_cell_fall(lu_template_name){
          index_1 ("float,..., float");
          index_2 ("float,..., float");
          index_3 ("float,..., float");
          values ("float,..., float",
          ...
          "float,..., float");
        }
    }
  }
}
```
ocv_std_dev_retaining_rise(lu_template_name){
    index_1 ("float,…, float");
    index_2 ("float,…, float");
    index_3 ("float,…, float");
    values ( "float,…, float", 
            …,
            "float,…, float");
}

ocv_std_dev_retaining_fall(lu_template_name){
    index_1 ("float,…, float");
    index_2 ("float,…, float");
    index_3 ("float,…, float");
    values ( "float,…, float", 
            …,
            "float,…, float");
}

ocv_mean_shift_cell_rise(lu_template_name){
    index_1 ("float,…, float");
    index_2 ("float,…, float");
    index_3 ("float,…, float");
    values ( "float,…, float", 
            …,
            "float,…, float");
}

ocv_mean_shift_cell_fall(lu_template_name){
    index_1 ("float,…, float");
    index_2 ("float,…, float");
    index_3 ("float,…, float");
    values ( "float,…, float", 
            …,
            "float,…, float");
}

ocv_mean_shift_retaining_rise(lu_template_name){
    index_1 ("float,…, float");
    index_2 ("float,…, float");
    index_3 ("float,…, float");
    values ( "float,…, float", 
            …,
            "float,…, float");
}

ocv_mean_shift_retaining_fall(lu_template_name){
    index_1 ("float,…, float");
    index_2 ("float,…, float");
    index_3 ("float,…, float");
    values ( "float,…, float", 
            …,
            "float,…, float");
}

ocv_skewness_cell_rise(lu_template_name){
    index_1 ("float,…, float");
    index_2 ("float,…, float");
    index_3 ("float,…, float");
}
values ( "float,..., float", \n    ..., \n    "float,..., float");
}
ocv_skewness_cell_fall(lu_template_name){
    index_1 ("float,..., float");
    index_2 ("float,..., float");
    index_3 ("float,..., float");
    values ( "float,..., float", \n    ..., \n    "float,..., float");
}
ocv_skewness_retaining_rise(lu_template_name){
    index_1 ("float,..., float");
    index_2 ("float,..., float");
    index_3 ("float,..., float");
    values ( "float,..., float", \n    ..., \n    "float,..., float");
}
ocv_skewness_retaining_fall(lu_template_name){
    index_1 ("float,..., float");
    index_2 ("float,..., float");
    index_3 ("float,..., float");
    values ( "float,..., float", \n    ..., \n    "float,..., float");
}

/* transition nominal values */
rise_transition (lu_template_name) { ... 
}
fall_transition (lu_template_name) { ... 
}
/* transition variation values */
ocv_std_dev_rise_transition(lu_template_name){ ... 
}
ocv_std_dev_fall_transition(lu_template_name){ ... 
}
ocv_std_dev_retain_rise_slew(lu_template_name){ ... 
}
ocv_std_dev_retain_fall_slew(lu_template_name){ ... 
}
ocv_mean_shift_rise_transition(lu_template_name){ ... 
}
ocv_mean_shift_fall_transition(lu_template_name){ ... 
}
...}
} ocv_mean_shift_retain_rise_slew(lu_template_name){
 ...}
} ocv_mean_shift_retain_fall_slew(lu_template_name){
 ...}
} ocv_skewness_rise_transition(lu_template_name){
 ...}
} ocv_skewness_fall_transition(lu_template_name){
 ...}
} ocv_skewness_retain_rise_slew(lu_template_name){
 ...}
} ocv_skewness_retain_fall_slew(lu_template_name){
 ...}
} /* end of timing */
} /* end of inout | output pin */
... pin | bus | bundle (name) {
direction: inout | input;
timing() {
 .../* constraint nominal values */
 rise_constraint(lu_template_name){
 ...}
 fall_constraint(lu_template_name){
 ...}
/* constraint variation values */
 ocv_std_dev_rise_constraint(lu_template_name){
 ...}
 ocv_std_dev_fall_constraint(lu_template_name){
 ...}
 ocv_mean_shift_rise_constraint(lu_template_name){
 ...}
 ocv_mean_shift_fall_constraint(lu_template_name){
 ...}
 ocv_skewness_rise_constraint(lu_template_name){
 ...}
 ocv_skewness_fall_constraint(lu_template_name){
 ...}
Library-Level Groups

This section describes library-level groups to model asymmetric timing variation distributions of delays, transitions, and constraints.

lu_table_template Group

The lu_table_template group defines the template for the following timing-level groups:

• ocv_std_dev_*
• ocv_mean_shift_*
• ocv_skewness_*

The template definition is same as that of the “lu_table_template Group” on page 15-20 described in “LVF Retain Arc Models.”

Timing Arc Level Groups

This section describes timing-level groups to model asymmetric timing variation distributions of delays, transitions, and constraints. These groups are lookup tables that can be scalar, one, two, or three-dimensional.

Define these groups in the timing group together with the nominal retain arc tables.

These groups are defined under the timing group together with the nominal tables.

ocv_std_dev_* Groups

The ocv_std_dev_cell_rise, ocv_std_dev_cell_fall, ocv_std_dev_rise_transition, ocv_std_dev_fall_transition, ocv_std_dev_retaining_rise, ocv_std_dev_retaining_fall, ocv_std_dev_retain_rise_slew, ocv_std_dev_retain_fall_slew, ocv_std_dev_rise_constraint, and ocv_std_dev_fall_constraint look up tables store the values of standard deviation of the timing variation distribution.
**ocv_mean_shift_* Groups**

The `ocv_mean_shift_cell_rise`, `ocv_mean_shift_cell_fall`, `ocv_mean_shift_rise_transition`, `ocv_mean_shift_fall_transition`, `ocv_mean_shift_retaining_rise`, `ocv_mean_shift_retaining_fall`, `ocv_mean_shift_retain_rise_slew`, `ocv_mean_shift_retain_fall_slew`, `ocv_mean_shift_rise_constraint`, and `ocv_mean_shift_fall_constraint` lookup tables specify the offset value from the mean of the timing variation distribution to the nominal value.

The mean values of the timing variation distribution are calculated as:

- mean rise delay = nominal rise delay + `ocv_mean_shift_cell_rise`
- mean fall delay = nominal fall delay + `ocv_mean_shift_cell_fall`
- mean rise transition = nominal rise transition + `ocv_mean_shift_rise_transition`
- mean fall transition = nominal fall transition + `ocv_mean_shift_fall_transition`
- mean retaining rise = nominal retaining rise + `ocv_mean_shift_retaining_rise`
- mean retaining fall = nominal retaining fall + `ocv_mean_shift_retaining_fall`
- mean retain rise slew = nominal retain rise slew + `ocv_mean_shift_retain_rise_slew`
- mean retain fall skew = nominal retain fall skew + `ocv_mean_shift_retain_fall_slew`
- mean rise constraint = nominal rise constraint + `ocv_mean_shift_rise_constraint`
- mean fall constraint = nominal fall constraint + `ocv_mean_shift_fall_constraint`

**ocv_skewness_* Groups**

These `ocv_skewness_cell_rise`, `ocv_skewness_cell_fall`, `ocv_skewness_rise_transition`, `ocv_skewness_fall_transition`, `ocv_skewness_retaining_rise`, `ocv_skewness_retaining_fall`, `ocv_skewness_retain_rise_slew`, `ocv_skewness_retain_fall_slew`, `ocv_skewness_rise_constraint`, and `ocv_skewness_fall_constraint` lookup tables specify the skewness values of the timing variation distribution.
Library Example

The following example shows a library with moment-based LVF models of a biased timing variation distribution.

library (lib1) {
  lu_table_template (2D_lu_template) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    index_1 ("1,2,3");
    index_2 ("1,2,3");
  }
  cell (cell1) {
    pin(pin1) {
      direction: output;
      timing() {
        related_pin : "CLKIN" ;
        timing_type : rising_edge ;
        cell_rise( 2D_lu_template ){
          index_1 ( "0.3, 0.4, 0.6");
          index_2 ( "0.01, 0.02, 0.03");
          values ( "0.04, 0.05, 0.06",
                    "0.07, 0.08, 0.09",
                    "0.1, 0.2, 0.3");
        }
        cell_fall ( scalar ){
          values ("0.095");
        }
        rise_transition( 2D_lu_template ){
          index_1 ( "0.3, 0.4, 0.6");
          index_2 ( "0.01, 0.02, 0.03");
          values ( "0.04, 0.05, 0.06",
                    "0.07, 0.08, 0.09",
                    "0.1, 0.2, 0.3");
        }
        fall_transition ( scalar ){
          values ("0.095");
        }
        ocv_std_dev_cell_rise( 2D_lu_template ){
          index_1 ( "0.3, 0.4, 0.6");
          index_2 ( "0.01, 0.02, 0.03");
          values ( "0.04, 0.05, 0.06",
                    "0.07, 0.08, 0.09",
                    "0.1, 0.2, 0.3");
        }
        ocv_std_dev_cell_fall ( scalar ){
          values ("0.005");
        }
        ocv_std_dev_rise_transition( 2D_lu_template ){
          ...
ocv_std_dev_fall_transition ( scalar ){  
values ("0.005");
}

ocv_std_dev_retaining_rise ( 2D_lu_template ){  
index_1 ("0.3, 0.4, 0.6");  
index_2 ("0.01, 0.02, 0.03");  
values ("0.04, 0.05, 0.06",  
      "0.07, 0.08, 0.09",  
      "0.1, 0.2, 0.3");
}

ocv_std_dev_retaining_fall ( scalar ){  
values ("0.005");
}

ocv_std_dev_retain_rise_slew ( 2D_lu_template ){  
index_1 ("0.3, 0.4, 0.6");  
index_2 ("0.01, 0.02, 0.03");  
values ("0.04, 0.05, 0.06",  
      "0.07, 0.08, 0.09",  
      "0.1, 0.2, 0.3");
}

ocv_std_dev_retain_fall_slew ( scalar ){  
values ("0.005");
}

ocv_mean_shift_cell_rise ( 2D_lu_template ){  
index_1 ("0.3, 0.4, 0.6");  
index_2 ("0.01, 0.02, 0.03");  
values ("0.04, 0.05, 0.06",  
      "0.07, 0.08, 0.09",  
      "0.1, 0.2, 0.3");
}

ocv_mean_shift_cell_fall ( scalar ){  
values ("0.015");
}

ocv_mean_shift_rise_transition ( 2D_lu_template ){  
index_1 ("0.3, 0.4, 0.6");  
index_2 ("0.01, 0.02, 0.03");  
values ("0.04, 0.05, 0.06",  
      "0.07, 0.08, 0.09",  
      "0.1, 0.2, 0.3");
}

ocv_mean_shift_fall_transition ( scalar ){  
values ("0.005");
}

ocv_mean_shift_retaining_rise ( 2D_lu_template ){  
index_1 ("0.3, 0.4, 0.6");  
index_2 ("0.01, 0.02, 0.03");  
values ("0.04, 0.05, 0.06",  
      "0.07, 0.08, 0.09",  
      "0.1, 0.2, 0.3");
"0.07, 0.08, 0.09",
"0.1, 0.2, 0.3"};
}

ocv_mean_shift_retaining_fall ( scalar ){
  values("0.005");
}

ocv_mean_shift_retain_rise_slew( 2D_lu_template ){
  index_1 ( "0.3, 0.4, 0.6");
  index_2 ( "0.01, 0.02, 0.03");
  values ( "0.04, 0.05, 0.06",
    "0.07, 0.08, 0.09",
    "0.1, 0.2, 0.3");
}

ocv_mean_shift_retain_fall_slew ( scalar ){
  values("0.005");
}

ocv_skewness_cell_rise( 2D_lu_template ){
  index_1 ( "0.3, 0.4, 0.6");
  index_2 ( "0.01, 0.02, 0.03");
  values ( "0.04, 0.05, 0.06",
    "0.07, 0.08, 0.09",
    "0.1, 0.2, 0.3");
}

ocv_skewness_cell_fall( scalar ){
  values("0.015");
}

ocv_skewness_rise_transition( 2D_lu_template ){
  index_1 ( "0.3, 0.4, 0.6");
  index_2 ( "0.01, 0.02, 0.03");
  values ( "0.04, 0.05, 0.06",
    "0.07, 0.08, 0.09",
    "0.1, 0.2, 0.3");
}

ocv_skewness_fall_transition( scalar ){
  values("0.015");
}

ocv_skewness_retaining_rise( 2D_lu_template ){
  index_1 ( "0.3, 0.4, 0.6");
  index_2 ( "0.01, 0.02, 0.03");
  values ( "0.04, 0.05, 0.06",
    "0.07, 0.08, 0.09",
    "0.1, 0.2, 0.3");
}

ocv_skewness_retaining_fall ( scalar ){
  values("0.005");
}

ocv_skewness_retain_rise_slew( 2D_lu_template ){
  index_1 ( "0.3, 0.4, 0.6");
  index_2 ( "0.01, 0.02, 0.03");
  values ( "0.04, 0.05, 0.06",
    "0.07, 0.08, 0.09",
    "0.1, 0.2, 0.3");
}
ocv_skewness_retain_fall_slew ( scalar ){
    values ("0.005");
}

} /* end of timing */

} /* end of pin */

} /* end of cell */

} /* end of library */
Defining I/O Pads

To define I/O pads, you use the library, cell, and pin group attributes that describe input, output, and bidirectional pad cells.

To model I/O pads, you must understand the following concepts covered in this chapter:

- Special Characteristics of I/O Pads
- Identifying Pad Cells
- Defining Units for Pad Cells
- Describing Input Pads
- Describing Output Pads
- Modeling Wire Load for Pads
- Programmable Driver Type Support in I/O Pad Cell Models
- Pad Cell Examples
Special Characteristics of I/O Pads

I/O pads are the special cells at the chip boundaries that allow communication with the world outside the chip. Their characteristics distinguish them from the other cells that make up the core of an integrated circuit.

Pads typically have longer delays and higher drive capabilities than the cells in an integrated circuit’s core. Because of their higher drive, CMOS pads sometimes exhibit noise problems. Slew-rate control is available on output pads to help alleviate this problem.

A distinguishing feature of pad cells is the voltage level at which input pads transfer logic 0 or logic 1 signals to the core or at which output pad drivers communicate logic values from the core.

Integrated circuits that communicate with one another must have compatible voltage levels at their pads. Because pads communicate with the world outside the integrated circuit, you must describe the pertinent units of peripheral library properties, such as external load, drive capability, delay, current, power, and resistance. This description makes it easier to design chips from multiple technologies.

You must capture all these properties in the library to make it possible for the integrated circuit designer to insert the correct pads during synthesis.

Identifying Pad Cells

Use the attributes described in the following sections to specify I/O pads and pad pin behaviors.

is_pad Attribute

After you identify a cell as a pad cell, you must indicate which pin represents the pad. The is_pad simple attribute must be used on at least one pin of a cell with a pad_cell attribute.

The direction attribute indicates whether the pad is an input, output, or bidirectional pad.

Syntax

is_pad : true | false ;

Example

cell(INBUF){
  ...
  pin(PAD){
    direction : input ;
    is_pad : true ;
  }
}
driver_type Attribute

A driver_type attribute defines two types of signal modifications: transformation and resolution.

- Transformation specifies an actual signal transition from 0/1 to L/H/Z. This signal transition performs a function on an input signal and requires only a straightforward mapping.

- Resolution resolves the value Z on an existing circuit node without actually performing a function and implies a constant (0/1) signal source as part of the resolution.

Syntax

```
driver_type : pull_up | pull_down | open_drain | open_source | bus_hold | resistive | resistive_0 | resistive_1 ;
```

pull_up

The pin is connected to power through a resistor. If it is a three-state output pin, it is in the Z state and its function is evaluated as a resistive 1 (H). If it is an input or inout pin and the node to which it is connected is in the Z state, it is considered an input pin at logic 1 (H). For a pull_up cell, the pin constantly stays at logic 1 (H).

pull_down

The pin is connected to ground through a resistor. If it is a three-state output pin, it is in the Z state and its function is evaluated as a resistive 0 (L). If it is an input or inout pin and the node to which it is connected is in the Z state, it is considered an input pin at logic 0 (L). For a pull_down cell, the pin constantly stays at logic 0 (L).

open_drain

The pin is an output pin without a pull-up transistor. Use this driver type only for off-chip output or inout pins representing pads. The pin goes to high impedance (Z) when its function is evaluated as logic 1.

Note:

- An n-channel open-drain pad is flagged with open_drain, and a p-channel open-drain pad is flagged with open_source.

open_source

The pin is an output pin without a pull-down transistor. Use this driver type only for off-chip output or inout pins representing pads. The pin goes to high impedance (Z) when its function is evaluated as logic 0.
bus_hold

The pin is a bidirectional pin on a bus holder cell. The pin holds the last logic value present at that pin when no other active drivers are on the associated net. Pins with this driver type cannot have function or three_state attributes.

resistive

The pin is an output pin connected to a controlled pull-up or pull-down resistor with a control port EN. When EN is disabled, the pull-up or pull-down resistor is turned off and has no effect on the pin. When EN is enabled, a functional value of 0 evaluated at the pin is turned into a weak 0, a functional value of 1 is turned into a weak 1, but a functional value of Z is not affected.

resistive_0

The pin is an output pin connected to power through a pull-up resistor that has a control port EN. When EN is disabled, the pull-up resistor is turned off and has no effect on the pin. When EN is enabled, a functional value of 1 evaluated at the pin turns into a weak 1, but a functional value of 0 or Z is not affected.

resistive_1

The pin is an output pin connected to ground through a pull-down resistor that has a control port EN. When EN is disabled, the pull-down resistor is turned off and has no effect on the pin. When EN is enabled, a functional value of 0 evaluated at the pin turns into a weak 0, but a functional value of 1 or Z is not affected.
Table 16-1 lists the driver types, their signal mappings, and the applicable pin types.

<table>
<thead>
<tr>
<th>Driver type</th>
<th>Description</th>
<th>Signal mapping</th>
<th>Applicable pin types</th>
</tr>
</thead>
<tbody>
<tr>
<td>pull_up</td>
<td>Resolution</td>
<td>01Z -&gt; 01H</td>
<td>in, out</td>
</tr>
<tr>
<td>pull_down</td>
<td>Resolution</td>
<td>01Z -&gt; 01L</td>
<td>in, out</td>
</tr>
<tr>
<td>bus_hold</td>
<td>Resolution</td>
<td>01Z -&gt; 01S</td>
<td>inout</td>
</tr>
<tr>
<td>open_drain</td>
<td>Transformation</td>
<td>01Z -&gt; 0ZZ</td>
<td>out</td>
</tr>
<tr>
<td>open_source</td>
<td>Transformation</td>
<td>01Z -&gt; Z1Z</td>
<td>out</td>
</tr>
<tr>
<td>resistive</td>
<td>Transformation</td>
<td>01Z -&gt; LHZ</td>
<td>out</td>
</tr>
<tr>
<td>resistive_0</td>
<td>Transformation</td>
<td>01Z -&gt; 0HZ</td>
<td>out</td>
</tr>
<tr>
<td>resistive_1</td>
<td>Transformation</td>
<td>01Z -&gt; L1Z</td>
<td>out</td>
</tr>
</tbody>
</table>

In Table 16-1, the pull up, pull down, and bus hold driver types define a resolution scheme. The remaining driver types define transformations.

The following example describes an output pin with a pull-up resistor and the bidirectional pin on a bus_hold cell.

**Example**

```plaintext
cell (bus) {
  pin(Y) {
    direction : output ;
    driver_type : pull_up ;
    pulling_resistance : 10000 ;
    function : "IO" ;
    three_state : "OE" ;
  }
}
cell (bus_hold) {
  pin(Y) {
    direction : inout ;
    driver_type : bus_hold ;
  }
}
```

Chapter 16: Defining I/O Pads
Identifying Pad Cells
Defining Units for Pad Cells

To process pads for full-chip synthesis, specify the units of time, capacitance, resistance, voltage, current, and power:

- `time_unit`
- `capacitive_load_unit`
- `pulling_resistance_unit`
- `voltage_unit`
- `current_unit`
- `leakage_power_unit`

All these attributes are defined at the library level, as described in "Input-Capacitance Characterization Attributes" on page 17-12. These values are required.

Capacitance

The `capacitive_load_unit` attribute defines the capacitance associated with a standard load. If you already represent capacitance values in terms of picofarads or femtofarads, use this attribute to define your base unit. If you represent capacitance in terms of the standard load of an inverter, define the exact capacitance for that inverter—for example, 0.101 pF.

Example

```plaintext
capacitive_load_unit( 0.1,ff );
```

Resistance

You must supply a `pulling_resistance` attribute for pull-up and pull-down devices on pads and identify the unit to use with the `pulling_resistance_unit` attribute.

Example

```plaintext
pulling_resistance_unit : "1kohm";
```

Voltage

You can use the `input_voltage` and `output_voltage` groups to define a set of input or output voltage ranges for your pads. To define the units of voltage you use for these groups, use the `voltage_unit` attribute. All the attributes defined inside `input_voltage` and
output_voltage groups are scaled by the value defined for voltage_unit. In addition, the voltage attribute in the operating_conditions groups also represents its values in these units.

Example
voltage_unit : "1V";

Current
You can define the drive current that can be generated by an output pad and also define the pulling current for a pull-up or pull-down transistor under nominal voltage conditions. Define all current values with the library-level current_unit attribute.

Example
current_unit : "1uA";

Describing Input Pads
To represent input pads in your technology library, you must describe the input voltage characteristics and indicate whether hysteresis applies.

The input pad properties are described in the next section. Examples at the end of this chapter describe a standard input buffer, an input buffer with hysteresis, and an input clock buffer.

hysteresis Attribute
Specify the hysteresis attribute for an input pad for a long transition time or when the pad is driven by a noisy line.

The default is false. When true, the vil and vol voltage ratings are actual transition points.

Example
hysteresis : true;

Pads with hysteresis can have derating factors that are different from the core cells. Use the scaled_cell group to describe the timing of cells with hysteresis. This construct provides derating capabilities and minimum, typical, or maximum timing for cells.
Describing Output Pads

To represent output pads in your technology library, you must describe the output voltage characteristics and the drive-current rating of output and bidirectional pads. Additionally, you must include information about the slew rate of the pad. These output pad properties are described in the sections that follow.

Examples at the end of this chapter show a standard output buffer and a bidirectional pad.

---

Drive Current

Output and bidirectional pads in a technology can have different drive-current capabilities. To define the drive current supplied by the pad buffer, use the `drive_current` attribute on an output or bidirectional pad or auxiliary pad pin. The value is in units consistent with the `current_unit` attribute you defined.

Example

```plaintext
pin(PAD) {
    direction : output;
    is_pad : true;
    drive_current : 1.0;
}
```

---

Slew-Rate Control

The `slew_control` attribute accepts one of four possible enumerations: none, low, medium, and high; the default is none. Increasing the slew control level slows down the transition rate. This method is the coarsest way to measure the level of slew-rate control associated with an output pad.

In Figure 16-1, A is a positive number and a linear approximation of the change in current as a function of time from the beginning of the rising transition to the threshold. B is a negative number and a linear approximation of the current change over time from the threshold to the end of transition.
Figure 16-1  Slew-Rate Attributes—Rising Transitions

For falling transitions, the graph is reversed and A becomes negative and B positive as shown in Figure 16-2.
Modeling Wire Load for Pads

You can define several wire_load groups that contain all the information to estimate interconnect wiring delays. Estimated wire loads for pads can be significantly different from those of core cells.
The wire load model for the net connecting an I/O pad to the core needs to be handled separately, because such a net is usually longer than most other nets in the circuit. Some pad nets extend completely across the chip.

You can define the `wire_load` group, which you want to use for wire load estimation, on the pad ring. Add a level of hierarchy by placing the pads in the top level and by placing all the core circuitry at a lower level.

### Programmable Driver Type Support in I/O Pad Cell Models

To support pull-up and pull-down circuit structures, the Liberty models for I/O pad cells support pull-up and pull-down driver information using the `driver_type` attribute with the `pull_up` or `pull_down` values.

Liberty syntax also supports conditional (programmable) pull-up and pull-down driver information for I/O pad cells. The programmable pin syntax has also been extended to other `driver_type` attribute values, such as `bus_hold`, `open_drain`, `open_source`, `resistive`, `resistive_0`, and `resistive_1`.

### Syntax

The following syntax supports programmable driver types in I/O pad cell models. Unlike the nonprogrammable driver type support, the programmable driver type support allows you to specify more than one driver type within a pin.

```
pin (pin_name) { /* programmable driver type pin */
  ...
  pull_up_function : "function string";
  pull_down_function : "function string";
  bus_hold_function : "function string";
  open_drain_function : "function string";
  open_source_function : "function string";
  resistive_function : "function string";
  resistive_0_function : "function string";
  resistive_1_function : "function string";
  ...
}
```
Programmable Driver Type Functions

The driver type function attributes in Table 16-2 help model the programmable driver types. The same rules that apply to nonprogrammable driver types also apply to these functions.

Table 16-2 Programmable Driver Type Functions

<table>
<thead>
<tr>
<th>Programmable driver type</th>
<th>Applicable on pin types</th>
</tr>
</thead>
<tbody>
<tr>
<td>pull_up_function</td>
<td>Input, output and inout</td>
</tr>
<tr>
<td>pull_down_function</td>
<td>Input, output and inout</td>
</tr>
<tr>
<td>bus_hold_function</td>
<td>Inout</td>
</tr>
<tr>
<td>open_drain_function</td>
<td>Output and inout</td>
</tr>
<tr>
<td>open_source_function</td>
<td>Output and inout</td>
</tr>
<tr>
<td>resistive_function</td>
<td>Output and inout</td>
</tr>
<tr>
<td>resistive_0_function</td>
<td>Output and inout</td>
</tr>
<tr>
<td>resistive_1_function</td>
<td>Output and inout</td>
</tr>
</tbody>
</table>

With the exception of pull_up_function and pull_down_function, if any of the driver type functions in Table 16-2 is specified on an inout pin, it is used only for output pins.

The following rules apply to programmable driver type functions (as well as nonprogrammable driver types in I/O pad cell models):

- The attribute can be applied to pad cell only.
- Only the input and inout pin can be specified in the function string.
- The function string is a Boolean function of input pins.

The following rules apply to an inout pin:

- If pull_up_function or pull_down_function and open_drain_function are specified within the same inout pin, pull_up_function or pull_down_function is used for the input pins.
- If bus_hold_function is specified on an inout pin, it is used for input and output pins.
Example

Example 16-2  Model of a Programmable Driver Type in an I/O Pad Cell

library(cond_pull_updown_example) {
    delay_model : table_lookup;

time_unit : 1ns;
voltage_unit : 1V;
capacitive_load_unit (1.0, pf);
current_unit : 1mA;

cell(conditional_PU_PD) {
    dont_touch : true;
    dont_use : true;
    pad_cell : true;
    pin(IO) {
        drive_current : 1;
        min_capacitance : 0.001;
        min_transition : 0.0008;
        is_pad : true;
        direction : inout;
        max_capacitance : 30;
        max_fanout : 2644;
        function : "(A*ETM')+(TA*ETM)"
        three_state : "(TEN*ETM')+(EN*ETM)"
        pull_up_function : "(!P1 * !P2)"
        pull_down_function : "( P1 * P2)"
        capacitance : 2.06649;
        timing() {
            related_pin : "IO A ETM TEN TA"
            cell_rise(scalar) {
                values("0")
            }
            rise_transition(scalar) {
                values("0")
            }
            cell_fall(scalar) {
                values("0")
            }
            fall_transition(scalar) {
                values("0")
            }
        }
    }
}

timing() {
    timing_type : three_state_disable;
    related_pin : "EN ETM TEN"
    cell_rise(scalar) {
        values("0")
    }
    rise_transition(scalar) {
        values("0")
    }
}
cell_fall(scalar) {
  values("0") ;
}
fall_transition(scalar) {
  values("0") ;
}

pin(ZI) {
  direction : output;
  function : "IO";
  timing() {
    related_pin : "IO";
    cell_rise(scalar) {
      values("0") ;
    }
    rise_transition(scalar) {
      values("0") ;
    }
    cell_fall(scalar) {
      values("0") ;
    }
    fall_transition(scalar) {
      values("0") ;
    }
  }
}

pin(A) {
  direction : input;
  capacitance : 1.0;
}

pin(EN) {
  direction : input;
  capacitance : 1.0;
}

pin(TA) {
  direction : input;
  capacitance : 1.0;
}

pin(TEN) {
  direction : input;
  capacitance : 1.0;
}

pin(ETM) {
  direction : input;
  capacitance : 1.0;
}

pin(P1) {
  direction : input;
  capacitance : 1.0;
}

pin(P2) {
Pad Cell Examples

These are examples of input, clock, output, and bidirectional pad cells.

Input Pads

Example 16-3  Standard Input Buffer

```liberty
library (example1) {
  date : "August 14, 2015" ;
  revision : 2015.05;
  ...
  time_unit : "ns";
  voltage_unit : "1V";
  current_unit : "1uA";
  pulling_resistance_unit : "1kohm";
  capacitive_load_unit( 0.1, ff );
  ...
  define_cell_area(bond_pads,pad_slots);
  define_cell_area(driver_sites,pad_driver_sites);
  ...
  input_voltage(CMOS) {
    vil : 1.5;
    vih : 3.5;
    vimin : -0.3;
    vimax : VDD + 0.3;
  }
  ...
  /***** INPUT PAD*****/
  cell(INBUF) {
    area : 0.000000;
    pad_cell : true;
    bond_pads : 1;
    driver_sites : 1;
    pin(PAD ) {
      direction : input;
      is_pad : true;
      input_voltage : CMOS;
      capacitance : 2.500000;
      fanout_load : 0.000000;
    }
    pin(Y ) {
      direction : output;
    }
  }
}
```
Example 16-4  Input Buffer With Hysteresis

library (example1) {
  date : "August 14, 2015" ;
  revision : 2015.05;
  ...
  time_unit : "ns";
  voltage_unit : "1V";
  current_unit : "1uA";
  pulling_resistance_unit : "1kohm";
  capacitive_load_unit( 0.1, ff );
  ...
  input_voltage(CMOS_SCHMITT) {
    vil : 1.0;
    vih : 4.0;
    vimin : -0.3;
    vimax : VDD + 0.3;
  }
  ...
  /*INPUT PAD WITH HYSTERESIS*/
  cell(INBUFH) {
    area : 0.000000;
    pad_cell : true;
    pin(PAD ) {
      direction : input;
      is_pad : true;
      hysteresis : true;
      input_voltage : CMOS_SCHMITT;
      capacitance : 2.500000;
      fanout_load : 0.000000;
    }
    pin(Y ) {
      direction : output;
      function : "PAD";
    }
  }
}
Example 16-5  Input Clock Buffer

library (example1) {
  date : "August 12, 2015" ;
  revision : 2015.05;
  ...
  time_unit : "ns";
  voltage_unit : "1V";
  current_unit : "1uA";
  pulling_resistance_unit : "1kohm";
  capacitive_load_unit( 0.1, ff );
  ...
  input_voltage(CMOS) {
    vil : 1.5;
    vih : 3.5;
    vimin : -0.3;
    vimax : VDD + 0.3;
  }
  ...
  /***/ C Lock INPUT BUFFER /***/
  cell (CLKBUF) {
    area : 0.000000;
    pad_cell : true;
    pad_type : clock;
    pin(PAD ) {
      direction : input;
      is_pad : true;
      input_voltage : CMOS;
      capacitance : 2.500000;
      fanout_load : 0.000000;
    }
    pin(Y ) {
      direction : output;
      function : "PAD";
  }
  }
}
max_fanout : 2000.000000;
timing() {
cell_rise(scalar) {
    values( " 5.70 ");
}
rise_transition(scalar) {
    values( " 0.009921 ");
}
cell_fall(scalar) {
    values( " 6.90 ");
}
fall_transition(scalar) {
    values( " 0.010238 ");
}
related_pin : "PAD";
}

Output Pads

Example 16-6 Standard Output Buffer

library (example1) {
date : "August 12, 2015" ;
revision : 2015.05;
... 
time_unit : "1ns";
voltage_unit : "1V";
current_unit : "1uA";
pulling_resistance_unit : "1kohm";
capacitive_load_unit( 0.1,ff );
...
output_voltage(GENERAL) {
    vol : 0.4;
    voh : 2.4;
    vomin : -0.3;
    vomax : VDD + 0.3;
}
/***** OUTPUT PAD *****/
cell(OUTBUF) {
    area : 0.000000;
    pad_cell : true;
    pin(D ) {
        direction : input;
        capacitance : 1.800000;
    }
    pin(PAD ) {
        direction : output;
        is_pad : true;
        drive_current : 2.0;
        output_voltage : GENERAL;
}
function : "D";
timing() {
cell_rise(scalar) {
    values( " 8.487 ");
}
rise_transition(scalar) {
    values( " 0.16974 ");
}
cell_fall(scalar) {
    values( " 9.347 ");
}
fall_transition(scalar) {
    values( " 0.18696 ");
}
related_pin :"D";
}

Bidirectional Pad

Example 16-7  Bidirectional Pad Cell With Three-State Enable

library (example1) {
    date : "August 12, 2015" ;
    revision : 2015.08;
    ...
    time_unit : "1ns";
    voltage_unit : "1V";
    current_unit : "1uA";
    pulling_resistance_unit : "1kohm";
    capacitance_load_unit( 0.1,ff );
    ...
    output_voltage(GENERAL) {
    vol : 0.4;
    voh : 2.4;
    vmin : -0.3;
    vmax : VDD + 0.3;
    }
    /***** BIDIRECTIONAL PAD *****/
    cell(BIBUF) {
        area : 0.000000;
        pad_cell : true;
        pin(E D ) {
            direction : input;
            capacitance : 1.800000;
        }
        pin(Y ) {
            direction : output;
            function : "PAD";
            driver_type : "open_source pull_up";
            pulling_resistance : 10000;
```
timing() { 
    cell_rise(scalar) { 
        values( " 3.07 ");
    }
    rise_transition(scalar) { 
        values( " 0.50 ");
    }
    cell_fall(scalar) { 
        values( " 2.95 ");
    }
    fall_transition(scalar) { 
        values( " 0.50 ");
    }
    related_pin :"PAD";
}
}

pin(PAD ) { 
    direction : inout;
    is_pad : true;
    drive_current : 2.0;
    output_voltage : GENERAL;
    input_voltage : CMOS;
    function : "D";
    three_state : "E";
    timing() { 
        cell_rise(scalar) { 
            values( " 8.483 ");
        }
        rise_transition(scalar) { 
            values( " 0.34686 ");
        }
        cell_fall(scalar) { 
            values( " 19.065 ");
        }
        fall_transition(scalar) { 
            values( " 0.3813 ");
        }
        related_pin :"E";
    }
    timing() { 
        cell_rise(scalar) { 
            values( " 17.466 ");
        }
        rise_transition(scalar) { 
            values( " 0.16974 ");
        }
        cell_fall(scalar) { 
            values( " 9.348 ");
        }
        fall_transition(scalar) { 
            values( " 0.10696 ");
        }
        related_pin :"D";
    }
}
```
Cell with contention_condition and x_function

Example 16-8 Cell With contention_condition and x_function Attributes

default_fanout_load : 0.1;
default_inout_pin_cap : 0.1;
default_input_pin_cap : 0.1;
default_output_pin_cap : 0.1;

capacitive_load_unit(1, pf);
pulling_resistance_unit : 1ohm;

voltage_unit : 1V;
current_unit : 1mA;
time_unit : 1ps;

cell (cell_a) {
    area : 1;
    contention_condition : "!ap & an";
        pin (ap, an) {
            direction : input;
            capacitance : 1;
        }
        pin (io) {
            direction : output;
            function : "!ap & !an";
            three_state : "ap & !an";
            x_function : "!ap & an";
        }
        timing() {
            related_pin : "ap an";
            timing_type : three_state_disable;
            cell_rise(scalar) {
                values( " 0.10 ");
            }
            rise_transition(scalar) {
                values( " 0.01 ");
            }
            cell_fall(scalar) {
                values( " 0.10 ");
            }
            fall_transition(scalar) {
                values( " 0.01 ");
            }
        }
        timing() {
            related_pin : "ap an";
            timing_type : three_state_enable;
        }
    }
}
cell_rise(scalar) {
    values(" 0.10 ");
}
rise_transition(scalar) {
    values(" 0.01 ");
}
cell_fall(scalar) {
    values(" 0.10 ");
}
fall_transition(scalar) {
    values(" 0.01 ");
}

pin(z) {
    direction: output;
    function: "!ap & !an";
    _x_function: "!ap & an | ap & !an";
}

See “contention_condition Attribute” on page 4-6 and the “x_function Attribute” description in “Describing Clock Pin Functions” on page 4-35 for more information about these attributes.
Library Characterization Configuration

Library information is generated by characterizing the behavior of the library cells under specific conditions. You can specify these conditions in one or more of the `char_config` groups. Specifying the library characterization settings includes the following concepts and tasks explained in this chapter:

- **The char_config Group**
- **Common Characterization Attributes**
- **CCS Timing Characterization Attributes**
- **Input-Capacitance Characterization Attributes**
The char_config Group

The char_config group represents library characterization configuration and is a group of attributes that specify the settings to characterize a library. The library characterization settings include general and specific settings. The general settings are for common tasks, such as characterizing delays, input waveforms, output loads, and handling simulation results. The specific settings include settings for specific characterization models, such as delay, slew, constraint, power, and capacitance models.

Without the appropriate settings, library data can be misinterpreted. This can result in significant differences between the library data and SPICE simulation results. These settings are also critical for accurate recharacterization of the library.

You can define the char_config group within the library, cell, pin, and timing groups. You should use only one char_config group within each of these groups.

Library Characterization Configuration Syntax

Example 17-1 shows the general syntax for library characterization configuration.

Example 17-1  General Syntax for Library Characterization Configuration

```plaintext
library (library_name) {
    char_config() { /* characterization configuration attributes */
    }
    ...
    cell (cell_name) {
        char_config() { /* characterization configuration attributes */
        }
        ...
        pin(pin_name) {
            char_config() { /* characterization configuration attributes */
            }
            timing() {
                char_config() { /* characterization configuration attributes */
                }
            } /* end of timing */
            ...
        } /* end of pin */
        ...
    } /* end of cell */
    ...
} /* end of library */
```
The `char_config` group includes simple, and complex characterization configuration attributes.

These characterization configuration attributes are divided into the following categories:

- Common configuration
- Three-state
- Composite current source (CCS) timing
- Input capacitance

Example 17-2 shows the use of these attributes to document the library characterization settings.

**Example 17-2  Library Characterization Configuration Example**

```plaintext
library (library_test) {
  lu_table_template (waveform_template) {
    variable_1 : input_net_transition;
    variable_2 : normalized_voltage;
    index_1 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7");
    index_2 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }

  normalized_driver_waveform (waveform_template) {
    driver_waveform_name : input_driver;
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", 
           "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", 
           "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }

  normalized_driver_waveform (waveform_template) {
    driver_waveform_name : input_driver_cell_test;
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", 
           "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", 
           "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }

  normalized_driver_waveform (waveform_template) {
    driver_waveform_name : input_driver_rise;
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", 
           "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", 
           "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }

  normalized_driver_waveform (waveform_template) {
    driver_waveform_name : input_driver_fall;
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", 
           "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", 
           "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }
}
```
char_config() {
    /* library level default attributes*/
    driver_waveform(all, input_driver);
    input_stimulus_transition(all, 0.1);
    input_stimulus_interval(all, 100.0);
    unrelated_output_net_capacitance(all, 1.0);
    default_value_selection_method(all, any);
    merge_tolerance_abs(nldm, 0.1);
    merge_tolerance_abs(constraint, 0.1);
    merge_tolerance_abs(capacitance, 0.01);
    merge_tolerance_abs(nlpm, 0.05);
    merge_tolerance_rel(all, 2.0);
    merge_selection(all, max);
    ccs_timing_segment_voltage_tolerance_rel: 1.0;
    ccs_timing_delay_tolerance_rel: 2.0;
    ccs_timing_voltage_margin_tolerance_rel: 1.0;
    receiver_capacitance1_voltage_lower_threshold_pct_rise: 20.0;
    receiver_capacitance1_voltage_upper_threshold_pct_rise: 50.0;
    receiver_capacitance1_voltage_lower_threshold_pct_fall: 50.0;
    receiver_capacitance1_voltage_upper_threshold_pct_fall: 80.0;
    receiver_capacitance2_voltage_lower_threshold_pct_rise: 20.0;
    receiver_capacitance2_voltage_upper_threshold_pct_rise: 50.0;
    receiver_capacitance2_voltage_lower_threshold_pct_fall: 50.0;
    receiver_capacitance2_voltage_upper_threshold_pct_fall: 80.0;
    capacitance_voltage_lower_threshold_pct_rise: 20.0;
    capacitance_voltage_lower_threshold_pct_fall: 50.0;
    capacitance_voltage_upper_threshold_pct_rise: 50.0;
    capacitance_voltage_upper_threshold_pct_fall: 80.0;
...
}

... cell (cell_test) {
    char_config() {
        /* input driver for cell test specifically */
        driver_waveform(all, input_driver_cell_test);
        /* specific default arc selection method for constraint */
        default_value_selection_method(constraint, max);
        default_value_selection_method_rise(nldm_transition, min);
        default_value_selection_method_fall(nldm_transition, max);
...
    }
    ...
    pin(pin1) {
        char_config() {
            driver_waveform_rise(delay, input_driver_rise);
        }
    }
    ...
    timing() {
        char_config() {
            driver_waveform_rise(constraint, input_driver_rise);
            driver_waveform_fall(constraint, input_driver_fall);
            /* specific ccs segmentation tolerance for this timing arc */
            ccs_timing_segment_voltage_tolerance_rel: 2.0;
        }
    }
}
Common Characterization Attributes

To specify the common characterization settings, set the common configuration attributes. All common configuration attributes of the `char_config` group are complex. A complex characterization configuration attribute has the following syntax:

**Syntax**

```plaintext
complex_attribute_name ( char_model, value ) ;
```

The first argument of the complex attribute is the characterization model. The second argument is a value of this attribute, such as a waveform name, a specific characterization method, a numerical value of a model parameter, or other values. Use the syntax to apply the attribute value to a specific characterization model. You can specify multiple complex attributes in the `char_config` group. You can also specify a single complex attribute multiple times for different characterization models.

However, when you specify the same attribute in multiple `char_config` groups at different levels, such as at the library, cell, pin, and timing levels, the attribute specified at the lower level gets priority over the ones specified at the higher levels. For example, the pin-level `char_config` group attributes have higher priority over the library-level `char_config` group attributes. **Table 17-1** lists the valid characterization models of the `char_config` group and their corresponding descriptions.

**Table 17-1  Valid Characterization Models of the char_config Group**

<table>
<thead>
<tr>
<th>Characterization model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Default model. The all model has the lowest priority among the characterization models. Any other characterization model overrides the all model.</td>
</tr>
<tr>
<td>ndlm</td>
<td>Nonlinear delay model (NLDM)</td>
</tr>
<tr>
<td>nldm_delay</td>
<td>Specific NLDMs with higher priority over the default NLDM</td>
</tr>
<tr>
<td>nldm_transition</td>
<td></td>
</tr>
<tr>
<td>capacitance</td>
<td>Capacitance model</td>
</tr>
<tr>
<td>constraint</td>
<td>Constraint model</td>
</tr>
</tbody>
</table>
Table 17-1  Valid Characterization Models of the char_config Group (continued)

<table>
<thead>
<tr>
<th>Characterization model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint_setup</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>constraint_hold</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>constraint_recovery</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>constraint_removal</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>constraint_skew</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>constraint_min_pulse_width</td>
<td>Nonlinear power model (NLPM)</td>
</tr>
<tr>
<td>constraint_no_change</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>constraint_non_seq_setup</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>constraint_non_seq_hold</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>constraint_minimum_period</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>nlpm</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>nlpm_leakage</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>nlpm_input</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
<tr>
<td>nlpm_output</td>
<td>Specific constraint models with higher priority over the general constraint model</td>
</tr>
</tbody>
</table>

Example 17-3 shows the syntax of the characterization configuration complex attributes.

Example 17-3  Syntax of Common Configuration Complex Attributes in the char_config Group

```plaintext
char_config() {
    driver_waveform(char_model, waveform_name);
    driver_waveform_rise(char_model, waveform_name);
    driver_waveform_fall(char_model, waveform_name);
    input_stimulus_transition(char_model, float);
    input_stimulus_interval(char_model, float);
    unrelated_output_net_capacitance(char_model, float);
    default_value_selection_method(char_model, method);
    default_value_selection_method_rise(char_model, method);
    default_value_selection_method_fall(char_model, method);
    merge_tolerance_abs(char_model, float);
    merge_tolerance_rel(char_model, float);
    merge_selection(char_model, method);
    ...
}
```

Figure 17-1 illustrates the use of `driver_waveform`, `input_stimulus_transition`, and `input_stimulus_interval` attributes for the delay arc characterization of a D flip-flop.
In Figure 17-1, an input stimulus with multiple transitions characterizes the delay arc, CP to Q or QN. The `input_stimulus_transition` and `input_stimulus_interval` attributes define and control the transitions and corresponding intervals, respectively. The `driver_waveform` attribute controls the last transition of the clock pulse, CP.

**driver_waveform Attribute**

The `driver_waveform` attribute defines the driver waveform to characterize a specific characterization model.

You can define the `driver_waveform` attribute within the `char_config` group at the library, cell, pin, and timing levels. If you define the `driver_waveform` attribute within the `char_config` group at the library level, the library-level `normalized_driver_waveform` group is ignored when the `driver_waveform_name` attribute is not defined.
If you do not define this attribute in the char_config group, the ramp waveform is used by default.

---

**driver_waveform_rise and driver_waveform_fall Attributes**

The `driver_waveform_rise` and `driver_waveform_fall` attributes define a specific rising and falling driver waveform, respectively, for a specific characterization model.

You can define the `driver_waveform_rise` and `driver_waveform_fall` attributes within the char_config group at the library, cell, pin, and timing levels. If you define the `driver_waveform_rise` and `driver_waveform_fall` attributes within the char_config group at the library level, the library-level `normalized_driver_waveform` group is ignored when the `driver_waveform_name` attribute is not defined.

If you do not define these attributes in the char_config group, the ramp waveform is used by default.

---

**input_stimulus_transition Attribute**

The `input_stimulus_transition` attribute specifies the transition time for all the input-signal edges except the arc input pin's last transition, during generation of the input stimulus for simulation. For example, in Figure 17-1 on page 17-7, the last transition of the clock pulse, CP, uses the `driver_waveform` attribute, and not the `input_stimulus_transition` attribute.

The time units of the `input_stimulus_transition` attribute are specified by the library-level `time_unit` attribute.

You must define this attribute.

---

**input_stimulus_interval Attribute**

The `input_stimulus_interval` attribute specifies the time-interval between the input-signal toggles to generate the input stimulus for a characterization cell. The time units of this attribute are specified by the library-level `time_unit` attribute.

You must define the `input_stimulus_interval` attribute.
unrelated_output_net_capacitance Attribute

The unrelated_output_net_capacitance attribute specifies a load value for an output pin that is not a related output pin of the characterization model. The valid value is a floating-point number, and is defined by the library-level capacitive_load_unit attribute.

If you do not specify this attribute for the nldm_delay and nlpm_output characterization models, the unrelated output pins use the load value of the related output pin. However, you must specify this attribute for any other characterization model.

default_value_selection_method Attribute

The default_value_selection_method attribute defines the method of selecting a default for

- The delay arc from state-dependent delay arcs.
- The constraint arc from state-dependent constraint arcs.
- Pin-based minimum pulse-width constraints from simulated results with side pin combinations.
- Internal power arcs from multiple state-dependent internal_power groups.
- The cell_leakage_power attribute from the state-dependent values in leakage power models.
- The input-pin capacitance from capacitance values for input-slew values used for timing characterization.

default_value_selection_method_rise and default_value_selection_method_fall Attributes

Use the default_value_selection_method_rise and default_value_selection_method_fall attributes when the selection methods for rise and fall are different.

You must define either the default_value_selection_method attribute, or the default_value_selection_method_rise and default_value_selection_method_fall attributes. Table 17-2 lists the valid selection methods for the default_value_selection_method,
default_value_selection_method_rise, default_value_selection_method_fall, and merge_selection attributes and their descriptions.

Table 17-2 Valid Common Configuration Selection Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>Selects a random value from the state-dependent data</td>
</tr>
<tr>
<td>min</td>
<td>Selects the minimum value from the state-dependent data at each index point.</td>
</tr>
<tr>
<td>max</td>
<td>Selects the maximum value from the state-dependent data at each index point.</td>
</tr>
<tr>
<td>average</td>
<td>Selects an average value from the state-dependent data at each index point.</td>
</tr>
<tr>
<td>min_mid_table</td>
<td>When the state-dependent data is a lookup table (LUT), this method selects the minimum value from the LUT. The minimum value is selected by comparing the middle value in the LUT, with each of the table-values. Note: The middle value corresponds to an index value. If the number of index values is odd, then the middle value is taken as the median value. However, if the number of index values is even, then the smaller of the two values is selected as the middle value.</td>
</tr>
<tr>
<td>max_mid_table</td>
<td>When the state-dependent data is a lookup table (LUT), this method selects the maximum value from the LUT. The maximum value is selected by comparing the middle value in the LUT, with each of the table-values. Note: The middle value corresponds to an index value. If the number of index values is odd, then the middle value is taken as the median value. However, if the number of index values is even, then the smaller of the two values is selected as the middle value.</td>
</tr>
<tr>
<td>follow_delay</td>
<td>Selects the value from the state-dependent data for delay selection. This method is valid only for the nldm_transition characterization model, that is, the follow_delay method applies specifically to default transition-table selection and not any other default selection.</td>
</tr>
</tbody>
</table>

merge_tolerance_abs and merge_tolerance_rel Attributes

The merge_tolerance_abs and merge_tolerance_rel attributes specify the absolute and relative tolerances, respectively, to merge arc simulation results. Specify the absolute tolerance value in the corresponding library unit, and the relative tolerance value in percent, for example, 10.0 for 10 percent.
If you specify both the `merge_tolerance_abs` and `merge_tolerance_rel` attributes, the results are merged if either or both the tolerance conditions are satisfied. If you do not specify any of these attributes, data including identical data is not merged.

---

**merge_selection Attribute**

The `merge_selection` attribute specifies the method of selecting the merged data. When multiple sets of state-dependent data are merged, the attribute selects a particular set of the state-dependent data to represent the merged data. See Table 17-2 on page 17-10 for the valid methods and their descriptions of the `merge_selection` attribute.

You must define the `merge_selection` attribute if you have defined either of the `merge_tolerance_abs` or `merge_tolerance_rel` attributes.

---

**CCS Timing Characterization Attributes**

To specify the CCS timing characterization settings, set the simple attributes that define CCS timing generation. Example 17-4 shows the syntax of these simple attributes.

**Example 17-4  Syntax of CCS Timing Simple Attributes**

```plaintext
char_config() {
  ccs_timing_segment_voltage_tolerance_rel: float;
  ccs_timing_delay_tolerance_rel: float;
  ccs_timing_voltage_margin_tolerance_rel: float;
  receiver_capacitance1_voltage_lower_threshold_pct_rise : float;
  receiver_capacitance1_voltage_upper_threshold_pct_rise : float;
  receiver_capacitance1_voltage_lower_threshold_pct_fall : float;
  receiver_capacitance1_voltage_upper_threshold_pct_fall : float;
  receiver_capacitance2_voltage_lower_threshold_pct_rise : float;
  receiver_capacitance2_voltage_upper_threshold_pct_rise : float;
  receiver_capacitance2_voltage_lower_threshold_pct_fall : float;
  receiver_capacitance2_voltage_upper_threshold_pct_fall : float;
  ...
}
```

You must define all these attributes if the library includes a CCS model.

---

**ccs_timing_segment_voltage_tolerance_rel Attribute**

The `ccs_timing_segment_voltage_tolerance_rel` attribute specifies the maximum permissible voltage difference between the simulation waveform and the CCS waveform to select the CCS model point. The floating-point value is specified in percent, where 100.0 represents 100 percent maximum permissible voltage difference.
**ccs_timing_delay_tolerance_rel Attribute**

The `ccs_timing_delay_tolerance_rel` attribute specifies the acceptable difference between the CCS waveform delay and the delay measured from simulation. The floating-point value is specified in percent, where 100.0 represents 100 percent acceptable difference.

**ccs_timing_voltage_margin_tolerance_rel Attribute**

The `ccs_timing_voltage_margin_tolerance_rel` attribute specifies the voltage tolerance to determine whether a signal has reached the rail-voltage value. The floating-point value is specified as a percentage of the rail voltage, such as 96.0 for 96 percent of the rail voltage.

**CCS Receiver Capacitance Attributes**

The following attributes specify the current integration limits, as a percentage of the voltage, to calculate the CCS receiver capacitances:

- `receiver_capacitance1_voltage_lower_threshold_pct_rise`
- `receiver_capacitance1_voltage_upper_threshold_pct_rise`
- `receiver_capacitance1_voltage_lower_threshold_pct_fall`
- `receiver_capacitance1_voltage_upper_threshold_pct_fall`
- `receiver_capacitance2_voltage_lower_threshold_pct_rise`
- `receiver_capacitance2_voltage_upper_threshold_pct_rise`
- `receiver_capacitance2_voltage_lower_threshold_pct_fall`
- `receiver_capacitance2_voltage_upper_threshold_pct_fall`

The floating-point values of these attributes can vary from 0.0 to 100.0.

**Input-Capacitance Characterization Attributes**

To specify input-capacitance characterization settings, set the simple attributes that define input-capacitance measurement. **Example 17-5** shows the syntax of these simple attributes.
Example 17-5 Syntax of Input-Capacitance Characterization Simple Attributes

```c
char_config() {
    capacitance_voltage_lower_threshold_pct_rise : float;
    capacitance_voltage_lower_threshold_pct_fall : float;
    capacitance_voltage_upper_threshold_pct_rise : float;
    capacitance_voltage_upper_threshold_pct_fall : float;
    ...
}
```

Each floating-point threshold value is specified as a percentage of the supply voltage, and can vary from 0.0 to 100.0.

You must define all the simple attributes mentioned in Example 17-5, in the `char_config` group for input-capacitance characterization.

---

**capacitance_voltage_lower_threshold_pct_rise and capacitance_voltage_lower_threshold_pct_fall** Attributes

The `capacitance_voltage_lower_threshold_pct_rise` and `capacitance_voltage_lower_threshold_pct_fall` attributes specify the lower-threshold value of a rising and falling voltage waveform, respectively, for calculating the NLDM input-pin capacitance.

---

**capacitance_voltage_upper_threshold_pct_rise and capacitance_voltage_upper_threshold_pct_fall** Attributes

The `capacitance_voltage_upper_threshold_pct_rise` and `capacitance_voltage_upper_threshold_pct_fall` attributes specify the upper-threshold value of a rising and falling voltage waveform, respectively, for calculating the NLDM input-pin capacitance.
1. Physical Library Group Description and Syntax
   Attributes and Groups ................................................................. 1-2
      phys_library Group ................................................................. 1-18
         bus_naming_style Simple Attribute ........................................ 1-18
         capacitance_conversion_factor Simple Attribute ......................... 1-19
         capacitance_unit Simple Attribute .......................................... 1-19
         comment Simple Attribute ................................................... 1-19
         current_conversion_factor Simple Attribute ................................ 1-20
         current_unit Simple Attribute ............................................... 1-20
         date Simple Attribute ......................................................... 1-21
         dist_conversion_factor Simple Attribute ................................... 1-21
         distance_unit Simple Attribute .............................................. 1-21
         frequency_conversion_factor Simple Attribute ............................ 1-22
         frequency_unit Simple Attribute ............................................ 1-22
         has_wire_extension Simple Attribute ....................................... 1-23
         inductance_conversion_factor Simple Attribute ............................ 1-23
         inductance_unit Simple Attribute .......................................... 1-23
         is_incremental_library Simple Attribute .................................. 1-24
         manufacturing_grid Simple Attribute .................................... 1-24
         power_conversion_factor Simple Attribute .................................. 1-25
         power_unit Simple Attribute ................................................ 1-25
         resistance_conversion_factor Simple Attribute ............................ 1-25
         resistance_unit Simple Attribute .......................................... 1-26
         revision Simple Attribute .................................................... 1-26
         SiO2_dielectric_constant Simple Attribute ................................... 1-27
         time_conversion_factor Simple Attribute .................................... 1-27
         time_unit Simple Attribute .................................................. 1-27
         voltage_conversion_factor Simple Attribute ............................... 1-28
2. Specifying Attributes in the resource Group

Syntax for Attributes in the resource Group ........................................... 2-2
resource Group .............................................................................. 2-2
  contact_layer Complex Attribute .................................................. 2-3
device_layer Complex Attribute ....................................................... 2-3
overlap_layer Complex Attribute ...................................................... 2-4
substrate_layer Complex Attribute .................................................. 2-4

3. Specifying Groups in the resource Group

Syntax for Groups in the resource Group ............................................. 3-2
array Group ..................................................................................... 3-2
  floorplan Group .......................................................................... 3-2
cont_layer Group ............................................................................ 3-11
corner_min_spacing Simple Attribute .............................................. 3-12
max_stack_level Simple Attribute .................................................... 3-12
  spacing Simple Attribute ........................................................... 3-13
  enclosed_cut_rule Group ............................................................. 3-13
max_current_ac_absavg Group ....................................................... 3-16
max_current_ac_avg Group .............................................................. 3-17
max_current_ac_peak Group ............................................................ 3-17
max_current_ac_rms Group .............................................................. 3-18
max_current_dc_avg Group .............................................................. 3-19
implant_layer Group ........................................................................ 3-19
  min_width Simple Attribute ......................................................... 3-20
  spacing Simple Attribute ........................................................... 3-20
  spacing_from_layer Complex Attribute ..................................... 3-21
ndiff_layer Group ........................................................................... 3-21
  max_current_ac_absavg Group ...................................................... 3-21
  max_current_ac_avg Group ............................................................ 3-22
  max_current_ac_peak Group ............................................................ 3-23
  max_current_ac_rms Group ............................................................ 3-23
  max_current_dc_avg Group ............................................................ 3-24
pdiff_layer Group  ................................................................. 3-25
  max_current_ac_absavg Group ............................................. 3-25
  max_current_ac_avg Group ................................................... 3-25
  max_current_ac_peak Group .................................................. 3-26
  max_current_ac_rms Group ..................................................... 3-27
  max_current_dc_avg Group ..................................................... 3-27

poly_layer Group ............................................................... 3-28
  avg_lateral_oxide_permittivity Simple Attribute ....................... 3-29
  avg_lateral_oxide_thickness Simple Attribute ........................... 3-29
  height Simple Attribute ...................................................... 3-30
  oxide_permittivity Simple Attribute ...................................... 3-30
  oxide_thickness Simple Attribute ........................................ 3-31
  res_per_sq Simple Attribute ................................................. 3-31
  shrinkage Simple Attribute ................................................ 3-32
  thickness Simple Attribute ................................................. 3-32
  conformal_lateral_oxide Complex Attribute ................................ 3-33
  lateral_oxide Complex Attribute ......................................... 3-33
  max_current_ac_absavg Group .............................................. 3-34
  max_current_ac_avg Group .................................................... 3-35
  max_current_ac_peak Group .................................................. 3-35
  max_current_ac_rms Group ..................................................... 3-36
  max_current_dc_avg Group ..................................................... 3-37

routing_layer Group .......................................................... 3-37
  avg_lateral_oxide_permittivity Simple Attribute ....................... 3-39
  avg_lateral_oxide_thickness Simple Attribute ........................... 3-39
  baseline_temperature Simple Attribute ................................... 3-40
  cap_multiplier Simple Attribute .......................................... 3-40
  cap_per_sq Simple Attribute ................................................. 3-41
  coupling_cap Simple Attribute ............................................. 3-41
  default_routing_width Simple Attribute .................................. 3-42
  edgcapacitance Simple Attribute .......................................... 3-42
  field_oxide_permittivity Simple Attribute ................................ 3-43
  field_oxide_thickness Simple Attribute ................................... 3-43
  fill_active_spacing Simple Attribute ...................................... 3-44
  fringe_cap Simple Attribute ................................................. 3-44
  height Simple Attribute ...................................................... 3-45
  inductance_per_dist Simple Attribute .................................... 3-45
  max_current_density Simple Attribute .................................... 3-45
  max_length Simple Attribute ............................................... 3-46
  max_observed_spacing_ratio_for_lpe Simple Attribute .................. 3-46
  max_width Simple Attribute ............................................... 3-47
min_area Simple Attribute ................................. 3-47
min_enclosed_area Simple Attribute ......................... 3-48
min_enclosed_width Simple Attribute ......................... 3-48
min_fat_wire_width Simple Attribute ......................... 3-49
min_fat_via_width Simple Attribute ......................... 3-49
min_length Simple Attribute .................................. 3-50
min_width Simple Attribute .................................. 3-50
min_wire_split_width Simple Attribute ....................... 3-51
offset Simple Attribute .................................... 3-51
oxide_permittivity Simple Attribute ......................... 3-52
oxide_thickness Simple Attribute ............................. 3-52
pitch Simple Attribute .................................... 3-53
process_scale_factor Simple Attribute ....................... 3-53
res_per_sq Simple Attribute ................................ 3-54
res_temperature_coefficient Simple Attribute ............... 3-54
routing_direction Simple Attribute .......................... 3-54
same_net_min_spacing Simple Attribute ...................... 3-55
shrinkage Simple Attribute .................................. 3-55
spacing Simple Attribute .................................. 3-56
thickness Simple Attribute .................................. 3-57
u_shaped_wire_spacing Simple Attribute ...................... 3-57
wire_extension Simple Attribute ............................ 3-57
wire_extension_range_check_connect_only
Simple Attribute ............................................. 3-58
wire_extension_range_check_corner
Simple Attribute ............................................. 3-58
conformal_lateral_oxide Complex Attribute .................. 3-59
lateral_oxide Complex Attribute ............................. 3-60
min_extension_width Complex Attribute ...................... 3-60
min_shape_edge Complex Attribute ........................... 3-61
plate_cap Complex Attribute .................................. 3-61
ranged_spacing Complex Attribute ............................ 3-62
spacing_check_style Complex Attribute ....................... 3-63
stub_spacing Complex Attribute ............................... 3-63
end_of_line_spacing_rule Group ............................... 3-64
extension_via_rule Group .................................... 3-67
max_current_ac_absavg Group ................................. 3-70
max_current_ac_avg Group ..................................... 3-71
max_current_ac_peak Group ................................... 3-71
max_current_ac_rms Group .................................... 3-72
max_current_dc_avg Group .................................... 3-73
min_edge_rule Group ......................................... 3-73
min_enclosed_area_table Group ...................................... 3-76
notch_rule Group ...................................................... 3-77
resistance_table Group .................................................. 3-79
shrinkage_table Group ...................................................... 3-80
spacing_table Group ....................................................... 3-81
wire_extension_range_table Group ...................................... 3-82
routing_wire_model Group ........................................... 3-83
wire_length_x Simple Attribute ..................................... 3-84
wire_length_y Simple Attribute ..................................... 3-85
adjacent_wire_ratio Complex Attribute ......................... 3-85
overlap_wire_ratio Complex Attribute .......................... 3-86
wire_ratio_x Complex Attribute ..................................... 3-87
wire_ratio_y Complex Attribute ..................................... 3-87
site Group ................................................................. 3-88
on_tile Simple Attribute ............................................. 3-89
site_class Simple Attribute .......................................... 3-89
symmetry Simple Attribute ........................................... 3-89
size Complex Attribute .................................................. 3-90
tile Group ................................................................. 3-91
tile_class Simple Attribute ......................................... 3-91
size Complex Attribute ................................................ 3-92
via Group ................................................................. 3-92
capacitance Simple Attribute ....................................... 3-93
inductance Simple Attribute ......................................... 3-93
is_default Simple Attribute .......................................... 3-94
is_fat_via Simple Attribute .......................................... 3-94
resistance Simple Attribute ......................................... 3-95
res_temperature_coefficient Simple Attribute ............ 3-95
top_of_stack_only Simple Attribute .............................. 3-96
via_id Simple Attribute .................................................. 3-96
foreign Group ............................................................. 3-97
via_layer Group ............................................................ 3-99
via_array_rule Group .................................................. 3-104

4. Specifying Attributes in the topological_design_rules Group

   Syntax for Attributes in the topological_design_rules Group .......... 4-2
topological_design_rules Group ...................................... 4-2
   antenna_inout_threshold Simple Attribute .................. 4-3
   antenna_input_threshold Simple Attribute .................. 4-3
   antenna_output_threshold Simple Attribute .............. 4-3
5. Specifying Groups in the topological_design_rules Group

Syntax for Groups in the topological_design_rules Group

antenna_rule Group

adjusted_gate_area_calculation_method
Simple Attribute

adjusted_metal_area_calculation_method
Simple Attribute

antenna_accumulation_calculation_method
Simple Attribute

antenna_ratio_calculation_method
Simple Attribute

apply_to
Simple Attribute

gameometry_calculation_method
Simple Attribute

metal_area_scaling_factor_calculation_method
Simple Attribute

pin_calculation_method
Simple Attribute

routing_layer_calculation_method
Simple Attribute

adjusted_gate_area
Group

adjusted_metal_area
Group

antenna_ratio
Group

metal_area_scaling_factor
Group

default_via_generate
Group

density_rule
Group

extension_wire_spacing_rule
Group

extension_wire_qualifier
Group

min_total_projection_length_qualifier
Group

spacing_check_qualifier
Group

stack_via_max_current
Group

bottom_routing_layer
Simple Attribute

top_routing_layer
Simple Attribute
6. Specifying Attributes and Groups in the process_resource Group

Syntax for Attributes in the process_resource Group .......................... 6-2
  baseline_temperature Simple Attribute ...................................... 6-2
  field_oxide_thickness Simple Attribute ..................................... 6-2
  process_scale_factor Simple Attribute ...................................... 6-3
  plate_cap Complex Attribute .................................................. 6-3

Syntax for Groups in the process_resource Group ............................. 6-4
  process_cont_layer Group ...................................................... 6-4
  process_routing_layer Group .................................................. 6-5
  cap_multiplier Simple Attribute ............................................. 6-6
cap_per_sq Simple Attribute ........................................... 6-6
coupling_cap Simple Attribute ....................................... 6-7
edge capacitance Simple Attribute ................................. 6-7
fringe_cap Simple Attribute .......................................... 6-8
height Simple Attribute .............................................. 6-8
inductance_per_dist Simple Attribute ............................... 6-8
lateral_oxide_thickness Simple Attribute ......................... 6-9
oxide_thickness Simple Attribute ................................. 6-9
res_per_sq Simple Attribute ....................................... 6-10
shrinkage Simple Attribute ....................................... 6-10
thickness Simple Attribute ...................................... 6-11
conformal_lateral_oxide Complex Attribute ....................... 6-11
lateral_oxide Complex Attribute ................................. 6-12
resistance_table Group ........................................... 6-12
shrinkage_table Group ............................................. 6-13
process_via Group .................................................. 6-14
  capacitance Simple Attribute .................................. 6-15
  inductance Simple Attribute .................................. 6-15
  resistance Simple Attribute .................................. 6-16
  res_temperature_coefficient Simple Attribute ............. 6-16
process_via_rule_generate Group ................................. 6-17
  capacitance Simple Attribute .................................. 6-17
  inductance Simple Attribute .................................. 6-18
  resistance Simple Attribute .................................. 6-18
  res_temperature_coefficient Simple Attribute ............. 6-19
process_wire_rule Group ........................................... 6-19
process_via Group .................................................. 6-20

7. Specifying Attributes and Groups in the macro Group

macro Group ............................................................ 7-2
  cell_type Simple Attribute ...................................... 7-3
  create_full_pin_geometry Simple Attribute .................. 7-4
  eq_cell Simple Attribute .................................... 7-5
  extract_via_region_within_pin_area Simple Attribute ...... 7-5
  in_site Simple Attribute ..................................... 7-6
  in_tile Simple Attribute ..................................... 7-6
  leq_cell Simple Attribute .................................... 7-7
  source Simple Attribute ...................................... 7-7
  symmetry Simple Attribute ................................... 7-7
8. Specifying Attributes and Groups in the pin Group

- capacitance Simple Attribute ........................................ 8-3
- direction Simple Attribute ........................................ 8-3
- eq_pin Simple Attribute ........................................... 8-4
- must_join Simple Attribute ....................................... 8-4
- pin_shape Simple Attribute ....................................... 8-5
- pin_type Simple Attribute ........................................ 8-5
- antenna_contact_accum_area Complex Attribute .............. 8-6
- antenna_contact_accum_side_area Complex Attribute ........ 8-6
- antenna_contact_area Complex Attribute ....................... 8-7
- antenna_contact_area_partial_ratio Complex Attribute ...... 8-7
- antenna_contact_side_area Complex Attribute ................. 8-8
- antenna_contact_side_area_partial_ratio Complex Attribute 8-8
- antenna_diffusion_area Complex Attribute .................... 8-9
- antenna_gate_area Complex Attribute ........................... 8-9
- antenna_metal_accum_area Complex Attribute .................. 8-10
- antenna_metal_accum_side_area Complex Attribute ........... 8-10
- antenna_metal_area Complex Attribute ......................... 8-11
- antenna_metal_area_partial_ratio Complex Attribute ........ 8-11
antenna_metal_side_area Complex Attribute ........................................... 8-12
antenna_metal_side_area_partial_ratio Complex Attribute ......................... 8-12
foreign Group ...................................................................................... 8-13
  orientation Simple Attribute ......................................................... 8-13
  origin Complex Attribute .............................................................. 8-14
port Group ......................................................................................... 8-15
  via Complex Attribute ................................................................. 8-15
  via_iterate Complex Attribute ..................................................... 8-16
geometry Group .................................................................................. 8-17

9. Developing a Physical Library

  Creating the Physical Library ............................................................ 9-2
  Naming the Source File .................................................................... 9-2
  Naming the Physical Library ......................................................... 9-2
  Defining the Units of Measure ....................................................... 9-2

10. Defining the Process and Design Parameters

  Defining the Technology Data .......................................................... 10-2
  Defining the Architecture .............................................................. 10-2
  Defining the Layers .......................................................................... 10-2
    Contact Layer ............................................................................... 10-3
    Overlap Layer ............................................................................. 10-3
    Routing Layer ............................................................................... 10-3
    Specifying Net Spacing .............................................................. 10-5
    Device Layer ................................................................................ 10-6
  Defining Vias .................................................................................... 10-6
    Naming the Via ............................................................................. 10-6
    Defining the Via Properties ........................................................ 10-7
    Defining the Geometry for Simple Vias ....................................... 10-7
    Defining the Geometry for Special Vias ....................................... 10-8
    Referencing a Foreign Structure ................................................. 10-10
  Defining the Placement Sites ........................................................... 10-10
    Standard Cell Technology .......................................................... 10-10
    Gate Array Technology ............................................................... 10-12

11. Defining the Design Rules

  Defining the Design Rules .............................................................. 11-2
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining Minimum Via Spacing Rules in the Same Net</td>
<td>11-2</td>
</tr>
<tr>
<td>Defining Same-Net Minimum Wire Spacing</td>
<td>11-2</td>
</tr>
<tr>
<td>Defining Same-Net Stacking Rules</td>
<td>11-3</td>
</tr>
<tr>
<td>Defining Nondefault Rules for Wiring</td>
<td>11-3</td>
</tr>
<tr>
<td>Defining Rules for Selecting Vias for Special Wiring</td>
<td>11-5</td>
</tr>
<tr>
<td>Defining Rules for Generating Vias for Special Wiring</td>
<td>11-6</td>
</tr>
<tr>
<td>Defining the Generated Via Size</td>
<td>11-8</td>
</tr>
</tbody>
</table>

### Appendix A. Parasitic RC Estimation in the Physical Library

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling Parasitic RC Estimation</td>
<td>A-2</td>
</tr>
<tr>
<td>Variables Used in Parasitic RC Estimation</td>
<td>A-2</td>
</tr>
<tr>
<td>Variables for Routing Layers</td>
<td>A-2</td>
</tr>
<tr>
<td>Variables for Estimated Routing Wire Model</td>
<td>A-3</td>
</tr>
<tr>
<td>Equations for Parasitic RC Estimation</td>
<td>A-4</td>
</tr>
<tr>
<td>Capacitance per Unit Length for a Layer</td>
<td>A-4</td>
</tr>
<tr>
<td>Resistance and Capacitance for Each Routing Direction</td>
<td>A-5</td>
</tr>
<tr>
<td>.plib Format</td>
<td>A-5</td>
</tr>
</tbody>
</table>
Physical Library Group Description and Syntax

This chapter describes the role of the `phys_library` group in defining a physical library.

The information in this chapter includes a description and syntax example for the attributes that you can define within the `phys_library` group.
Attributes and Groups

The `phys_library` group is the superior group in the physical library. The `phys_library` group contains all the groups and attributes that define the physical library.

**Example 1-1** lists the attributes and groups that you can define within a physical library.

The following chapters include descriptions and syntax examples for the groups that you can define within the `phys_library` group.

**Example 1-1  Syntax for the Attributes and Groups in the Physical Library**

```plaintext
phys_library(library_nameid) {
  bus_naming_style: string;
  capacitance_conversion_factor: integer;
  capacitance_unit: 1pf | 1ff | 10ff | 100ff;
  comment: string;
  current_conversion_factor: integer;
  current_unit: 100uA | 100mA | 1A | 1uA | 10uA | 1mA | 10mA;
  date: string;
  dist_conversion_factor: integer;
  distance_unit: 1mm | 1um;
  frequency_conversion_factor: integer;
  frequency_unit: 1mhz;
  gds2_conversion_factor: integer;
  has_wire_extension: Boolean;
  inductance_conversion_factor: integer;
  inductance_unit: 1fh | 1ph | 1nh | 1uh | 1mh | 1h;
  is_incremental_library: Boolean;
  manufacturing_grid: float;
  power_conversion_factor: integer;
  power_unit: 1uw | 10uw | 100uw | 1mw | 10mw | 100mw | 1w;
  resistance_conversion_factor: integer;
  resistance_unit: 1ohm | 100ohm | 10ohm | 1kohm;
  revision: string;
  Si02_dielectric_constant: float;
  time_conversion_factor: integer;
  time_unit: 1ns | 100 ps | 10ps | 1ps;
  voltage_conversion_factor: integer;
  voltage_unit: 1mv | 10mv | 100mv | 1v;
  antenna_lut_template(template_nameid) {
    variable_1: antenna_diffusion_area;
    index_1("float, float, float, ...");
  } /* end antenna_lut_template */
  resistance_lut_template(template_nameid) {
    variable_1: routing_width | routing_spacing;
    variable_2: routing_width | routing_spacing;
    index_1("float, float, float, ...");
    index_2("float, float, float, ...");
  } /* end resistance_lut_template */
  shrinkage_lut_template(template_nameid) {
    variable_1: routing_width | routing_spacing;
    variable_2: routing_width | routing_spacing;
    index_1("float, float, float, ...");
    index_2("float, float, float, ...");
  }
}
```
/* end shrinkage_lut_template */

spacing_lut_template (template_name_id) {
  variable_1: routing_width;
  variable_2: routing_width; routing_length;
  variable_3: routing_length;
  index_1 (*float, float, float, ...*);
  index_2 (*float, float, float, ...*);
  index_3 (*float, float, float, ...*);
} /* end *spacing_lut_template */

wire_lut_template (template_name_id) {
  variable_1: extension_width | extension_length | bottom_routing_width |
    top_routing_width | routing_spacing | routing_width;
  variable_2: extension_width | extension_length | bottom_routing_width |
    top_routing_width | routing_spacing | routing_width;
  variable_3: extension_width | extension_length |
    routing_spacing | routing_width;
  index_1 (*float, float, float, ...*);
  index_2 (*float, float, float, ...*);
  index_3 (*float, float, float, ...*);
} /* end wire_lut_template */

resource (architecture enum) {
  contact_layer (layer_name_id);
  device_layer (layer_name_id);
  overlap_layer (layer_name_id);
  substrate_layer (layer_name_id);
  cont_layer (layer_name_id) {
    corner_min_spacing : float;
    max_current_density; float;
    max_stack_level; integer;
    spacing : float;
    enclosed_cut_rule () {
      max_cuts : integer;
      max_neighbor_cut_spacing : float;
      min_cuts : integer;
      min_enclosed_cut_spacing : float;
      min_neighbor_cut_spacing : float;
    } /* end enclosed_cut_rule */
  } /* end resource */

max_current_ac_absavg (template_name_id) {
  index_1 (*float, float, float, ...*);
  index_2 (*float, float, float, ...*);
  index_3 (*float, float, float, ...*);
  values (*float, float, float, ...*);
}

max_current_ac_avg (template_name_id) {
  index_1 (*float, float, float, ...*);
  index_2 (*float, float, float, ...*);
  index_3 (*float, float, float, ...*);
  values (*float, float, float, ...*);
}

max_current_ac_peak (template_name_id) {
  index_1 (*float, float, float, ...*);
  index_2 (*float, float, float, ...*);
  index_3 (*float, float, float, ...*);
  values (*float, float, float, ...*);
}

max_current_ac_rms (template_name_id) {

index_1 ("float, float, float, ...")
index_2 ("float, float, float, ...")
index_3 ("float, float, float, ...")
values ("float, float, float, ...")

max_current_dc_avg (template_name_id) {
  index_1 ("float, float, float, ...")
  index_2 ("float, float, float, ...")
  values ("float, float, float, ...")
}

} /* end cont_layer */
extension_via_rule () {
  related_layer : name_id;
  min_cuts_table ( wire_lut_template_name ) {
    index_1
    index_2
    values
  } /* end min_cuts_table */
  reference_cut_table ( via_array_lut_template_name ) {
    index_1
    index_2
    values
  } /* end reference_cut_table */
} /* end extension_via_rule */
implant_layer () {
  min_width : float;
  spacing ; float;
  spacing_from_layer (float, layer_name_id);
} /* end implant_layer */
ndiff_layer () {
  max_current_ac_absavg (template_name_id) {
    index_1 ("float, float, float, ...")
    index_2 ("float, float, float, ...")
    index_3 ("float, float, float, ...")
    values ("float, float, float, ...")
  }
  max_current_ac_avg (template_name_id) {
    index_1 ("float, float, float, ...")
    index_2 ("float, float, float, ...")
    index_3 ("float, float, float, ...")
    values ("float, float, float, ...")
  }
  max_current_ac_peak (template_name_id) {
    index_1 ("float, float, float, ...")
    index_2 ("float, float, float, ...")
    index_3 ("float, float, float, ...")
    values ("float, float, float, ...")
  }
  max_current_ac_rms (template_name_id) {
    index_1 ("float, float, float, ...")
    index_2 ("float, float, float, ...")
    index_3 ("float, float, float, ...")
    values ("float, float, float, ...")
  }
  max_current_dc_avg (template_name_id) {
    index_1 ("float, float, float, ...")
    index_2 ("float, float, float, ...")
    index_3 ("float, float, float, ...")
    values ("float, float, float, ...")
  }
} /* end ndiff_layer */
values ("float, float, float, ...") ;
}) /*end ndiff_layer */
pdiff_layer () {
  max_current_ac_absavg (template_nameid) {
    index_1 ("float, float, float, ...") ;
    index_2 ("float, float, float, ...") ;
    index_3 ("float, float, float, ...") ;
    values ("float, float, float, ...") ;
  }
  max_current_ac_avg (template_nameid) {
    index_1 ("float, float, float, ...") ;
    index_2 ("float, float, float, ...") ;
    index_3 ("float, float, float, ...") ;
    values ("float, float, float, ...") ;
  }
  max_current_ac_peak (template_nameid) {
    index_1 ("float, float, float, ...") ;
    index_2 ("float, float, float, ...") ;
    index_3 ("float, float, float, ...") ;
    values ("float, float, float, ...") ;
  }
  max_current_ac_rms (template_nameid) {
    index_1 ("float, float, float, ...") ;
    index_2 ("float, float, float, ...") ;
    index_3 ("float, float, float, ...") ;
    values ("float, float, float, ...") ;
  }
  max_current_dc_avg (template_nameid) {
    index_1 ("float, float, float, ...") ;
    index_2 ("float, float, float, ...") ;
    index_3 ("float, float, float, ...") ;
    values ("float, float, float, ...") ;
  }
} /*end pdiff_layer */
poly_layer (layer_nameid) {
  avg_lateral_oxide_permittivity : float ;
  avg_lateral_oxide_thickness : float ;
  conformal_lateral_oxide (thicknessfloat, topwall_thicknessfloat, sidewall_thicknessfloat, permittivityfloat)
  height : float ;
  lateral_oxide : (thicknessfloat, permittivityfloat)
  oxide_permittivity : float ;
  oxide_thickness : float ;
  res_per_sq : float ;
  shrinkage : float ;
  thickness : float ;
  max_current_ac_absavg (template_nameid) {
    index_1 ("float, float, float, ...") ;
    index_2 ("float, float, float, ...") ;
    index_3 ("float, float, float, ...") ;
    values ("float, float, float, ...") ;
  }
  max_current_ac_avg (template_nameid) {
    index_1 ("float, float, float, ...") ;
    index_2 ("float, float, float, ...") ;
    index_3 ("float, float, float, ...") ;
  }
max_current_ac_peak {template_nameid} {
    index_1 (*"float, float, float, ..."*) ;
    index_2 (*"float, float, float, ..."*) ;
    index_3 (*"float, float, float, ..."*) ;
    values (*"float, float, float, ..."*) ;
}
max_current_ac_rms {template_nameid} {
    index_1 (*"float, float, float, ..."*) ;
    index_2 (*"float, float, float, ..."*) ;
    index_3 (*"float, float, float, ..."*) ;
    values (*"float, float, float, ..."*) ;
}
max_current_dc_avg {template_nameid} {
    index_1 (*"float, float, float, ..."*) ;
    index_2 (*"float, float, float, ..."*) ;
    index_3 (*"float, float, float, ..."*) ;
    values (*"float, float, float, ..."*) ;
}
/* end poly_layer */
routing_layer(layer_nameid)) {
    avg_lateral_oxide_permittivity
    avg_lateral_oxide_thickness
    baseline_temperature : float ;
    cap_multiplier : float ;
    cap_per_sq : float ;
    conformal_lateral_oxide (thickness float ,
                         topwall_thickness float ,
                         sidewall_thickness float ,
                         permittivity float ) ;
    coupling_cap : float ;
    default_routing_width : float ;
    edgecapacitance : float ;
    field_oxide_permittivity : float ;
    field_oxide_thickness : float ;
    fill_active_spacing : float ;
    fringe_cap : float ;
    height : float ;
    inductance_per_dist : float ;
    lateral_oxide : (thickness float ,
                         permittivity float ) ;
    max_current_density : float ;
    max_length : float ;
    max_observed_spacing_ratio_for_lpe : float ;
    max_width : float ;
    min_area : float ;
    min_enclosed_area : float ;
    min_enclosed_width : float ;
    min_extension_width ;
    min_fat_wire_width : float ;
    min_fat_via_width : float ;
    min_length : float ;
    min_shape_edge (float , integer , Boolean ) ;
    min_width : float ;
    min_wire_split_width : float ;
    offset : float ;
    oxide_permittivity : float ;
    oxide_thickness : float ;
    pitch : float ;
    plate_cap(float , ..., float ) ;
process_scale_factor : float;
ranged_spacing(float, float, float);
res_per_sq : float;
res_temperature_coefficient : float;
routing_direction : vertical | horizontal;
same_net_min_spacing : float;
shrinkage : float;
spacing : float;
spacing_check_style : manhattan | diagonal;
stub_spacing(spacing,float, max_length_threshold,float);
thickness : float;
u_shaped_wire_spacing : float;
wire_extension : float;
wire_extension_range_check_connect_only : Boolean;
wire_extension_range_check_connect_corner : Boolean;
array(array_name) {
  floorplan(floorplan_nameid) {
    /* floorplan_name is optional */
    /* when omitted, results in default floorplan */
    site_array(site_nameid) {
      iterate(num_x_int, num_y_int, spacing_x,float, spacing_y,float);
      orientation : FE | FN | E | FS | FW | N | S | W;
      origin(x,float, y,float);
      placement_rule : regular | can_place | cannot_occupy;
    } /* end site_array */
  } /* end floorplan */
  routing_grid () {
    grid_pattern (float, integer, float);
    routing_direction : horizontal | vertical;
  } /* end routing_grid */
  tracks() {
    layers : "layer1_nameid, ..., layern_nameid";
    routing_direction : horizontal | vertical;
    track_pattern(float, integer, float);
    /* starting coordinate, number, spacing */
  } /* end tracks */
} /* end array */
end_of_line_spacing_rule () {
  end_of_line_corner_keepout_width : float;
  end_of_line_edge_checking : valueenum;
  end_of_line_metal_max_width : float;
  end_of_line_min_spacing : float;
}
max_current_ac_absavg (template_nameid) {
  index_1 ("float, float, float, ...");
  index_2 ("float, float, float, ...");
  index_3 ("float, float, float, ...");
  values ("float, float, float, ...");
}
max_current_ac_avg (template_nameid) {
  index_1 ("float, float, float, ...");
  index_2 ("float, float, float, ...");
  index_3 ("float, float, float, ...");
  values ("float, float, float, ...");
max_current_ac_peak (template_nameid) {
  index_1 (*float, float, float, ...) ;
  index_2 (*float, float, float, ...) ;
  index_3 (*float, float, float, ...) ;
  values (*float, float, float, ...) ;
}

max_current_ac_rms (template_nameid) {
  index_1 (*float, float, float, ...) ;
  index_2 (*float, float, float, ...) ;
  index_3 (*float, float, float, ...) ;
  values (*float, float, float, ...) ;
}

max_current_dc_avg (template_nameid) {
  index_1 (*float, float, float, ...) ;
  index_2 (*float, float, float, ...) ;
  index_3 (*float, float, float, ...) ;
  values (*float, float, float, ...) ;
}

min_edge_rule () {
  concave_corner_required : Boolean ;
  max_number_of_min_edges : valueint ;
  max_total_edge_length : float ;
  min_edge_length : float ;
}

min_enclosed_area_table () {
  index_1 (*float, float, float, ...) ;
  values (*float, float, float, ...) ;
}

notch_rule () {
  min_notch_edge_length : float ;
  min_notch_width : float ;
  min_wire_width : float ;
}

resistance_table (template_nameid) {
  index_1 (*float, float, float, ...) ;
  index_2 (*float, float, float, ...) ;
  values (*float, float, float, ...) ;
} /* end resistance_table */

shrinkage_table (template_nameid) {
  index_1 (*float, float, float, ...) ;
  index_2 (*float, float, float, ...) ;
  values (*float, float, float, ...) ;
} /* end shrinkage_table */

spacing_table (template_nameid) {
  index_1 (*float, float, float, ...) ;
  index_2 (*float, float, float, ...) ;
  index_3 (*float, float, float, ...) ;
  values (*float, float, float, ...) ;
} /* end spacing_table */

wire_extension_range_table (template_nameid) {
  index_1 (*float, float, float, ...) ;
  values (*float, float, float, ...) ;
} /* end wire_extension_range_table */

) /* end routing_layer */

routing_wire_model(model_nameid) {
  adjacent_wire_ratio(float, ..., float) ;
  overlap_wire_ratio(float, ..., float) ;
  wire_length_x(float) ;
wire_length_y(float);
wire_ratio_x(float, ..., float);
wire_ratio_y(float, ..., float);
} /* end routing_wire_model */
site(site_nameid) {
  on_tile : valueid;
site_class : pad | core ; /* default = core */
size(size_xfloat, size_yfloat);
symmetry : x | y | r | xy | rxy ; /* default = none */
} /* end site */
tile(tilename) {
  size (float, float);
tile_class : pad | core ; /* default = core */
}
via(via_nameid) {
capacitance : float;
inductance : float;
is_default : Boolean;
is_fat_via : Boolean;
res_temperature_coefficient : float;
resistance : float ; /* per contact-cut rectangle */
same_net_min_spacing(layer_nameid, layer_nameid, spacing_valuefloat,
is_stackBoolean);
top_of_stack_only : Boolean;
via_id : valueint;
foreign(foreign_object_nameid) {
  orientation : FE | FN | E | FS | FW | N | S | W ;
  origin(xfloat, yfloat);
} /* end foreign */
via_layer(layer_nameid) {
  contact_array_spacing ( float, float ) ;
  contact_spacing ( float, float ) ;
  enclosure ( float, float ) ;
  max_cuts ( valueint, valueint ) ;
  max_wire_width : float ;
  min_cuts ( integer , integer ) ;
  min_wire_width : float ;
  rectangle(X0float, Y0float, X1float, Y1float) ;
  /* 1 or more rectangle attributes allowed */ ;
  rectangle_iterate ( valueint, valueint, float, float, float,
                float,float,float ) ;
} /* end via_layer */
} /* end via */
via_array_rule () {
  min_cuts_table ( via_array_lut_template_name ) {
    index_1
    index_2
    values
  } * end min_cuts_table */
  reference_cut_table ( via_array_lut_template_name ) {
    index_1
    index_2
    values
  } /* end reference_cut_table */
} /* end via_array_rules */
} /*end resource */
topological_design_rules() {
    antenna_inout_threshold : float ;
    antenna_input_threshold : float ;
    antenna_output_threshold : float ;
    contact_min_spacing (layer_name_id, layer_name_id, float) ;
    corner_min_spacing (value_id, value_id, float) ;
    diff_net_min_spacing (value_id, value_id, float) ;
    end_of_line_enclosure (value_id, value_id, float) ;
    min_enclosure (value_id, value_id, float) ;
    min_enclosed_area_table_surrounding_metal : value_enum ;
    min_generated_via_size (float, float) ; /* x, y */
    min_overhang (layer_1_string, layer_2_string, spacing_value_float) ;
    same_net_min_spacing (layer_name_id, layer_name_id, spacing_value_float, is_stack Boolean) ;
    antenna_rule (antenna_rule_name_id) {
        adjusted_gate_area_calculation_method () ;
        adjusted_metal_area_calculation_method () ;
        antenna_accumulation_calculation_method () ;
        antenna_ratio_calculation_method () ;
        apply_to : gate_area | gate_perimeter | diffusion_area ;
        geometry_calculation_method : all_geometries | connected_only ;
        layer_antenna_factor (layer_name_string, antenna_factor_float) ;
        metal_area_scaling_factor_calculation_method : value_enum ;
        pin_calculation_method : all_pins | each_pin ;
        routing_layer_calculation_method : side_wall_area | top_area |
        side_wall_and_top_area | segment_length | segment_perimeter ;
        adjusted_gate_area () {
            index_1
            values
        }
        adjusted_metal_area () {
            index_1
            values
        }
        antenna_ratio (template_name_id) {
            index_1 (float, float, float, ...)
            values (float, float, float, ...)
        }
        metal_area_scaling_factor () {
            index_1 (float, float, float, ...)
            values (float, float, float, ...)
        }
    } /* end antenna_rule */
}

default_via_generate () {
    via_routing_layer() ()
    via_contact_layer () ()
}

density_rule () {
    check_window_size () ;
    check_step ;
    density_range () ;
}

extension_wire_spacing_rule () {
    extension_wire_qualifier () {
        
    }
}
connected_to_fat_wire : Boolean;
corner_wire : Boolean;
not_connected_to_fat_wire : Boolean;
) /* end extension_wire_spacing_rule */
min_total_projection_length_qualifier () {
  non_overlapping_projection : Boolean;
  overlapping_projection : Boolean;
  parallel_length : Boolean;
} /* end min_total_projection_length_qualifier */
spac ing_check_qualifier () {
  corner_to_corner : Boolean;
  non_overlapping_projection_wires : Boolean;
  overlapping_projection_wires : Boolean;
  wires_to_check : value enum;
} /* end spac ing_check_qualifier */
) /* end extension_wire_spacing_rule */
stack via max current () {
  bottom_routing_layer : routing_layer_name id;
  top_routing_layer : routing_layer_name id;
  max_current_ac_absavg (template_name id) {
    index_1 ("float, float, float, ...");
    index_2 ("float, float, float, ...");
    index_3 ("float, float, float, ...");
    values ("float, float, float, ...");
  }
  max_current_ac_avg (template_name id) {
    index_1 ("float, float, float, ...");
    index_2 ("float, float, float, ...");
    index_3 ("float, float, float, ...");
    values ("float, float, float, ...");
  }
  max_current_ac_peak (template_name id) {
    index_1 ("float, float, float, ...");
    index_2 ("float, float, float, ...");
    index_3 ("float, float, float, ...");
    values ("float, float, float, ...");
  }
  max_current_ac_rms (template_name id) {
    index_1 ("float, float, float, ...");
    index_2 ("float, float, float, ...");
    index_3 ("float, float, float, ...");
    values ("float, float, float, ...");
  }
  max_current_dc_avg (template_name id) {
    index_1 ("float, float, float, ...");
    index_2 ("float, float, float, ...");
    index_3 ("float, float, float, ...");
    values ("float, float, float, ...");
  }
) /* end stack via max current */
via rule (via_rule_name id) {
  routing_layer_rule (layer_name id) { /* 2 or more */
    contact_overhang : float;
    max_wire_width : float;
    metal_overhang : float;
    min_wire_width : float;
    routing_direction : horizontal | vertical;
via_list ;
} /* end routing_layer_rule */

vias : "via_name1id, ..., via_nameNid;"
} /* end via_rule */

via_rule_generate(via_rule_generate_nameid)
{
  capacitance : float;
  inductance : float;
  res_temperature_coefficient : float;
  resistance : float;
  routing_formula(layer_nameid)
  {
    contact_overhang : float;
    enclosure (float, float);
    max_wire_width : float;
    metal_overhang : float;
    min_wire_width : float;
    routing_direction : horizontal | vertical;
  }
} /* end routing_formula */

contact_formula(layer_nameid)
{
  contact_array_spacing(float, float);
  contact_spacing(Xfloat, Yfloat);
  max_cuts (valueint, valueint);
  max_cut_rows_current_direction : float;
  min_number_of_cuts : float;
  rectangle(X0float, Y0float, X1float, Y1float);
  resistance : float;
  routing_direction : valueenum;
} /* end contact_formula */

wire_rule(wire_rule_nameid)
{
  via(via_nameid)
  {
    capacitance : float;
    inductance : float;
    res_temperature_coefficient : float;
    resistance : float;
    same_net_min_spacing(layer_nameid, layer_nameid, spacing_valuefloat, is_stackBoolean);
  }
  foreign(foreign_object_nameid)
  {
    orientation : FE | FN | E | FS | FW | N | S | W;
    origin(float, float);
  }
} /* end foreign */

via_layer(layer_nameid)
{
  contact_array_spacing (float, float);
  enclosure (float, float);
  max_cuts (valueint, valueint);
  rectangle(X0float, Y0float, X1float, Y1float);
  /* 1 or more rectangles */
} /* end via_layer */

layer_rule(layer_nameid)
{
  min_spacing : float;
  same_net_min_spacing(layer_nameid, layer_nameid, spacing_valuefloat, is_stackBoolean);
  /* layer1, layer2, spacing, is_stack */
  wire_extension : float;
  wire_width : float;
wire_slotting_rule (wire_slotting_rule_nameid) {
    max_metal_density : float;
    min_length : float;
    min_width : float;
    slot_length_range (minfloat, maxfloat);
    slot_length_side_clearance (minfloat, maxfloat);
    slot_length_wise_spacing (minfloat, maxfloat);
    slot_width_range (minfloat, maxfloat);
    slot_width_side_clearance (minfloat, maxfloat);
    slot_width_wise_spacing (minfloat, maxfloat);
}

process_resource(process_nameid) {
    baseline_temperature : float;
    field_oxide_thickness : float;
    plate_cap(float, ..., float);
    process_scale_factor : float;
    process_cont_layer () {
        process_routing_layer(layer_nameid) {
            cap_multiplier : float;
            cap_per_sq : float;
            conformal_lateral_oxide (thicknessfloat, topwall_thicknessfloat,
                sidewall_thicknessfloat, permittivityfloat);
            coupling_cap : float;
            edgecapacitance : float;
            fringe_cap : float;
            height : float;
            inductance_per_dist : float;
            lateral_oxide (thicknessfloat, permittivityfloat);
            lateral_oxide_thickness : float;
            oxide_thickness : float;
            res_per_sq : float;
            shrinkage : float;
            thickness : float;
            resistance_table (template_nameid) {
                index_1 ("float, float, float, ...");  
                index_2 ("float, float, float, ...");  
                values ("float, float, float, ...");  
            } /* end resistance_table */
            shrinkage_table (template_nameid) {
                index_1 ("float, float, float, ...");  
                index_2 ("float, float, float, ...");  
                values ("float, float, float, ...");  
            } /* end shrinkage_table */
        } /* end process_routing_layer */
    } /* end process_resource */
}

process_via(via_nameid) {
    capacitance : float;
    inductance : float;
    res_temperature_coefficient : float;
    resistance : float; /* per contact-cut rectangle */
} /* end process_via */

process_via_rule_generate(via_nameid) {
    capacitance : float;
} /* end process_v
inductance : float;
res_temperature_coefficient : float;
resistance : float;
} /* end process_via_rule_generate */
process_wire_rule(wire_rule_name_id) {
    process_via(via_name_id) {
        capacitance : float;
        inductance : float;
        res_temperature_coefficient : float;
        resistance : float;
    } /* end process_via */
} /* end process_wire_rule */
} /* end process_resource */
visual_settings () {
    stipple (stipple_name_id) {
        height : integer;
        width : integer;
        pattern (value_1enum, ..., value_Nenum;
    } /* end stipple */
    primary_color () {
        light_blue : integer;
        light_green : integer;
        light_red : integer;
        medium_blue : integer;
        medium_green : integer;
        medium_red : integer;
    } /* end primary color */
    color (color_name_id) {
        blue_intensity : integer;
        green_intensity : integer;
        red_intensity : integer;
    } /* end color */
    height : integer;
    line_style (line_name_id) {
        pattern (value_1enum, ..., value_Nenum;
        width : integer;
    } /* end line_styles */
} /* end visual settings */
layer_panel () {
    display_layer (display_layer_name_id) {
        blink : Boolean;
        color : color_namestring;
        is_mask_layer : Boolean;
        line_style : line_style_namestring;
        mask_layer : layer_namestring;
        stipple : stipple_namestring;
        selectable : Boolean;
        visible : Boolean;
    } /* end display_layer */
} /* end layer_panel */
milkyway_layer_map () {
    stream_layer (layer_name_id) {
        gds_map (layer_int, datatype_int);
        mw_map (layer_int, datatype_int);
        net_type : power | ground | clock | signal | viabot | viatop;
        object_type : data | text | data_text;
pr_preparation_rules() {
    pr_view_extraction_rules() {
        pr_view_extraction_rules() {
            pr_view_extraction_rules() {
                apply_to_cell_type : valueenum;
                generate_cell_boundary : Boolean;
                blockage_extraction() {
                    max_dist_to_combine_blockage ("valuestring", valuefloat);
                    preserve_all_metal_blockage : Boolean;
                    routing_blockage_display : Boolean;
                    routing_blockage_includes_spacing : Boolean;
                    treat_all_layers_as_thin_wires : Boolean;
                    treat_layer_as_thin_wire (valuestring, valuestring, ... ) ;
                }
                pin_extraction () {
                    expand_small_pin_on_blockage : Boolean;
                    extract_connectivity : Boolean;
                    extract_connectivity_thru_cont_layers(valuestring, valuestring, ... ) ;
                    /* these three attributes can have multiple pair-statements */
                    must_conn_area_layer_map ("valuestring", valuestring);
                    must_conn_area_min_width ("valuestring", valuefloat);
                    pin2text_layer_map (valuestring, valuestring) ;
                }
                via_region_extraction () {
                    apply_to_vias (via_namestring, via_namestring, ... ) ;
                    apply_to_macro : Boolean;
                    use_rotated_vias : Boolean;
                    top_routing_layer : valuestring;
                }
            }
        }
    }
}

pr_boundary_generation_rules () {
    pr_boundary_generation () {
        bottom_boundary_offset : valuefloat;
        bottom_boundary_reference : valueenum;
        doubleback_pg_row : Boolean;
        left_boundary_offset : valuefloat;
        left_boundary_reference : valueenum;
        on_overlap_layer : Boolean;
        use_overlap_layer_as_boundary
    } tile_generation () {
        all_cells_single_height : Boolean;
        pg_rail_orientation : valueenum;
        tile_name : valueid;
        tile_height : valuefloat;
        tile_width : valuefloat;
    }
}

streamin_rules () {
    boundary_layer_map ( valueint, valueint ) ;
    overwrite_existing_cell : Boolean;
    save_unmapped_mw_layers : Boolean;
save_unmapped_stream_layers : Boolean;

update_existing_cell : Boolean;

use_boundary_layer_as_geometry : Boolean;

macro(cell_name_id) {
  cell_type : cover | bump_cover | ring | block | blackbox_block | pad |
  areaio_pad | input_pad | output_pad | inout_pad |
  power_pad | spacer_pad | core | antennadiode_core |
  feedthru_core | spacer_core | tiehigh_core | tielow_core |
  pre_endcap | post_endcap | topleft_endcap |
  topright_endcap | bottomleft_endcap |
  bottomright_endcap;

  create_full_pin_geometry : Boolean; /* default TRUE */
  eq_cell : eq_cell_name_id;
  extract_via_region_from_cont_layer (string, string, ...);
  extract_via_region_within_pin_area : Boolean;
  in_site : site_name_id;
  in_tile : tile_name_id;
  leq_cell : leq_cell_name_id;
  obs_clip_box(float, float, float, float); /* top, right, bottom, left */
  origin(float, float);
  source : user | generate | block;
  size(float, float);
  symmetry : x | y | xy | r | rxy; /* default = none */
  foreign(foreign_object_name_id) {
    orientation : FE | FN | E | FS | FW | N | S | W;
    origin(float, float);
  } /* end foreign */

  obs() {
    via(via_name_id, xfloat, yfloat);
    via_iterate(int, int, float, string, float, float);
    /* num_x, num_y, spacing_x, spacing_y, via_name_id, start_x, start_y */

    geometry(layer_name_id) {
      core_blockage_margin : valuefloat;
      feedthru_area_layer : valuestring;
      generate_core_blockage : Boolean;
      max_dist_to_combine_current_layer_blockage( valuefloat, valuefloat);
      path(float, float, float, ...);
      /* width, numX, numY, spaceX, spaceY, width, x0, y0, x1, y1, ... */
      path_iterate(integer, integer, float, float, float, ...);
      /* width, numX, numY, spaceX, spaceY, width, x0, y0, x1, y1, ... */
      polygon(float, float, float, float, float, float, ...);
      /* x, y, x0, y0, x1, x2, ..., */
      polygon_iterate(integer, integer, float, float, float, float, float, float, float, ...);
      /* numX, numY, spaceX, spaceY, x0, y0, x1, y1, ... */
      preserve_current_layer_blockage : Boolean;
      treat_current_layer_as_thin_wires : Boolean;
      rectangle(x0float, y0float, x1float, y1float);
      rectangle_iterate(integer, integer, float, float, float, float, float, float);
      /* numX, numY, spaceX, spaceY, x0, y0, x1, y1 */
      treat_current_layer_as_thin_wire : Boolean;
    } /* end geometry */
pin(pin_nameid) {
  antenna_contact_accum_area (float, float, float, ...) ;
  antenna_contact_accum_side_area (float, float, float, ...) ;
  antenna_contact_area (float, float, float, ...) ;
  antenna_contact_area_partial_ratio (float, float, float, ...) ;
  antenna_contact_side_area (float, float, float, ...) ;
  antenna_contact_side_area_partial_ratio (float, float, float, ...) ;
  antenna_gate_area (float, float, float, ...) ;
  antenna_metal_accum_area (float, float, float, ...) ;
  antenna_metal_accum_side_area (float, float, float, ...) ;
  antenna_metal_area (float, float, float, ...) ;
  antenna_metal_area_partial_ratio (float, float, float, ...) ;
  antenna_metal_side_area (float, float, float, ...) ;
  antenna_metal_side_area_partial_ratio (float, float, float, ...) ;
  antenna_diffusion_area (float, float, float, ...) ;
  antenna_gate_area (float, float, float, ...) ;
  antenna_metal_accum_area (float, float, float, ...) ;
  antenna_metal_accum_side_area (float, float, float, ...) ;
  antenna_metal_area (float, float, float, ...) ;
  antenna_metal_side_area (float, float, float, ...) ;
  antenna_metal_side_area_partial_ratio (float, float, float, ...) ;
  antenna_gate_area (float, float, float, ...) ;
  antenna_metal_accum_area (float, float, float, ...) ;
  antenna_metal_accum_side_area (float, float, float, ...) ;
  antenna_metal_area (float, float, float, ...) ;
  antenna_metal_side_area (float, float, float, ...) ;
  antenna_metal_side_area_partial_ratio (float, float, float, ...) ;
  capacitance : float ;
  direction : inout | input | feedthru | output | tristate ;
  eq_pin : pin_nameid ;
  must_join : pin_nameid ;
  pin_shape : clock | power | signal | analog | ground ;
  pin_type : clock | power | signal | analog | ground ;
foreign(foreign_object_nameid) {
  orientation : FE | FN | E | FS | FW | N | S | W ;
  origin(Xfloat, Yfloat) ;
} /* end foreign */
port() {
  via(via_nameid, float, float) ;
  via_iterate(integer, integer, float, float, string, float, float) ;
  /*num_x, num_y, spacing_x, spacing_y, via_nameid, start_x, start_y */
geometry(layer_nameid) {
  path(float, float, float, ...) ;
  /* width, numX, numY, spaceX, spaceY, width, x0, y0, x1, y1, ... */
  path_iterate(integer, integer, float, float, float, float, ...) ;
  /* width, numX, numY, spaceX, spaceY, width, x0, y0, x1, y1, ... */
  polygon(float, float, float, float, float, float, ...) ;
  /* x, y, x0, y0, x1, x2, ..., */
  polygon_iterate(integer, integer, float, float, float, ...) ;
  /* numX, numY, spaceX, spaceY, x0, x1, y0, y1, yl, ... */
  rectangle(X0float, Y0float, X1float, Y1float) ;
  /* numX, numY, spaceX, spaceY, x0, x0, x1, y0, y1 */
  rectangle_iterate(integer, integer, float, float, float, float, float, float, float, float) ;
  /* numX, numY, spaceX, spaceY, x0, y0, x1, y1 */
} /* end geometry */
} /* end port */
} /* end pin */
site_array(site_nameid) {
  orientation : FE | FN | E | FS | FW | N | S | W ;
  origin(Xfloat, Yfloat) ;
  iterate(num_xint, num_yint, spacing_Xfloat, spacing_Yfloat) ;
} /* end site_array */
} /* end macro */
} /* end phys_library */
**phys_library Group**
The first line in the `phys_library` group names the library. This line is the first executable statement in your library.

**Syntax**
```
phys_library (library_name_id) { 
    ... library description ...
}
```

*library_name*

The name of your physical library.

**Example**
```
phys_library(sample) { 
    ...library description...
}
```

**bus_naming_style Simple Attribute**
Defines a naming convention for bus pins.

**Syntax**
```
phys_library(library_name_id) { 
    ...bus_naming_style :"value_string";
    ...
}
```

*value*

Can contain alphanumeric characters, braces, underscores, dashes, or parentheses. Must contain one %s symbol and one %d symbol. The %s and %d symbols can appear in any order, but at least one nonnumeric character must separate them.

The colon character is not allowed in a bus_naming_style attribute value because the colon is used to denote a range of bus members.

You construct a complete bused-pin name by using the name of the owning bus and the member number. The owning bus name is substituted for the %s, and the member number replaces the %d.

**Example**
```
bus_naming_style : "%s[%d]" ;
```
capacitance_conversion_factor Simple Attribute

The `capacitance_conversion_factor` attribute specifies the capacitance resolution in the physical library database. For example, when you specify a value of 1000, all the capacitance values are stored in the database as 1/1000 of the `capacitance_unit` value.

Syntax

```plaintext
phys_library(library_name_id) {
  ...
  capacitance_conversion_factor : value_int ;
  ...
}
```

value

Valid values are any multiple of 10.

Example

```plaintext
capacitance_conversion_factor : 1000 ;
```

capacitance_unit Simple Attribute

The `capacitance_unit` attribute specifies the unit for capacitance.

Syntax

```plaintext
phys_library(library_name_id) {
  ...
  capacitance_unit : value_enum ;
  ...
}
```

value

Valid values are 1pf, 1ff, 10ff, 100ff, 1nf, 1uf, 1mf, and 1f.

Example

```plaintext
capacitance_unit : 1pf ;
```

comment Simple Attribute

This optional attribute lets you provide additional descriptive information about the library.

Syntax

```plaintext
phys_library(library_name_id) {
  comment : "value_string" ;
  ...
}
```
**value**

Any alphanumeric sequence.

**Example**

```plaintext
comment : "0.18 CMOS library for SNPS" ;
```

**current_conversion_factor Simple Attribute**

The `current_conversion_factor` attribute specifies the current resolution in the physical library database. For example, when you specify a value of 1000, all the current values are stored in the database as 1/1000 of the `current_unit` value.

**Syntax**

```plaintext
phys_library(library_nameid) {
  ...
  current_conversion_factor : valueint ;
  ...
}
```

**value**

Valid values are any multiple of 10.

**Example**

```plaintext
current_conversion_factor : 1000 ;
```

**current_unit Simple Attribute**

The `current_unit` attribute specifies the unit for current.

**Syntax**

```plaintext
phys_library(library_nameid) {
  ...
  current_unit : valueenum ;
  ...
}
```

**value**

Valid values are 1μA, 1mA, and 1A.

**Example**

```plaintext
current_unit : 1mA ;
```
**date Simple Attribute**

The `date` attribute specifies the library creation date.

**Syntax**

```
phys_library(library_nameid) {
  ...
  date : "value_string" ;
  ...
}
```

**value**

Any alphanumeric sequence.

**Example**

```
date : "1st Jan 2003" ;
```

**dist_conversion_factor Simple Attribute**

The `dist_conversion_factor` attribute specifies the distance resolution in the physical library database. For example, when you specify a value of 1000, all the distance values are stored in the database as 1/1000 of the `distance_unit` value.

**Syntax**

```
phys_library(library_nameid) {
  ...
  dist_conversion_factor : value_int ;
  ...
}
```

**value**

Valid values are any multiple of 10.

**Example**

```
dist_conversion_factor : 1000 ;
```

**distance_unit Simple Attribute**

The `distance` attribute specifies the linear distance unit.

**Syntax**

```
phys_library(library_nameid) {
  ...
  distance_unit : value_enum ;
  ...
}
```
value
Valid values are 1mm and 1um.

Example
distance_unit : 1mm ;

**frequency_conversion_factor Simple Attribute**

The `frequency_conversion_factor` attribute specifies the frequency resolution in the physical library database. For example, when you specify a value of 1000, all the frequency values are stored in the database as 1/1000 of the `frequency_unit` value.

**Syntax**

```plaintext
phys_library(library_nameid) {
  ...
  frequency_conversion_factor : value_int
  ...
}
```

value
Valid values are any multiple of 10.

Example

```plaintext
frequency_conversion_factor : 1 ;
```

**frequency_unit Simple Attribute**

The `frequency_unit` attribute specifies the frequency unit.

**Syntax**

```plaintext
phys_library(library_nameid) {
  ...
  frequency_unit : value_enum
  ...
}
```

value
The valid value is 1mhz.

Example

```plaintext
frequency_unit : 1mhz ;
```
has_wire_extension Simple Attribute

The `has_wire_extension` attribute specifies whether wires are extended by a half width at pins.

Syntax

```plaintext
phys_library(library_nameid) {
    ...
    has_wire_extension : valueBoolean ;
    ...
}
```

value

Valid values are TRUE (default) and FALSE.

Example

```plaintext
has_wire_extension : TRUE ;
```

inductance_conversion_factor Simple Attribute

The `inductance_conversion_factor` attribute specifies the inductance resolution in the physical library database. For example, when you specify a value of 1000, all the inductance values are stored in the database as 1/1000 of the `inductance_unit` value.

Syntax

```plaintext
phys_library(library_nameid) {
    ...
    inductance_conversion_factor : valueInt ;
    ...
}
```

value

Valid values are any multiple of 10.

Example

```plaintext
inductance_conversion_factor : 1000 ;
```

inductance_unit Simple Attribute

The `inductance_unit` attribute specifies the unit for inductance.

Syntax

```plaintext
phys_library(library_nameid) {
    ...
    inductance_unit : valueEnum ;
    ...
}
```
...}

value

Valid values are 1fh, 1ph, 1nh, 1uh, 1mh, and 1h.

Example

inductance_unit : 1ph ;

is_incremental_library Simple Attribute

The is_incremental_library attribute specifies whether this library is only a partial library which is meant to be used as an extension of a primary library.

Syntax

phys_library(library_name_id) {
    ...
    is_incremental_library : value Boolean ;
    ...
}

value

Valid values are TRUE (default) and FALSE.

Example

is_incremental_library : TRUE ;

manufacturing_grid Simple Attribute

The manufacturing_grid attribute defines the manufacture grid resolution in the physical library database. This is the smallest geometry size in this library for this process and uses the unit defined in the distance_unit attribute.

Syntax

phys_library(library_name_id) {
    ...
    manufacturing_grid : value float ;
    ...
}

value

Valid values are any positive floating-point number.

Example

manufacturing_grid : 100 ;
**power_conversion_factor Simple Attribute**

The `power_conversion_factor` attribute specifies the factor to use for power conversion.

**Syntax**

```
phys_library(library_nameid) {
    ...
    power_conversion_factor : value_int ;
    ...
}
```

**value**

Valid values are any positive integer.

**Example**

```
time_conversion_factor : 100 ;
```

---

**power_unit Simple Attribute**

The `power_unit` attribute specifies the unit for power.

**Syntax**

```
phys_library(library_nameid) {
    ...
    power_unit : value_enum ;
    ...
}
```

**value**

Valid values are 1uw, 10uw, 100uw, 1mw, 10mw, 100mw, and 1w.

**Example**

```
power_unit : 100 ;
```

---

**resistance_conversion_factor Simple Attribute**

The `resistance_conversion_factor` attribute specifies the resistance resolution in the physical library database. For example, when you specify a value of 1000, all the resistance values are stored in the database as 1/1000 of the `resistance_unit` value.

**Syntax**

```
phys_library(library_nameid) {
    ...
    resistance_conversion_factor : value_int ;
    ...
}
```
value

Valid values are any multiple of 10.

Example
resistance_conversion_factor : 1000 ;

resistance_unit Simple Attribute
The resistance_unit attribute specifies the unit for resistance.

Syntax
phys_library(library_nameid) {
  ...
  resistance_unit : valueenum ;
  ...
}

value

Valid values are 1mohm, 1ohm, 10ohm, 100ohm, 1kohm, and 1Mohm.

Example
resistance_unit : 1ohm ;

revision Simple Attribute
This optional attribute lets you specify the library revision number.

Syntax
phys_library(library_nameid) {
  ...
  revision : "valuestring ";
  ...
}

value

Any alphanumeric sequence.

Example
revision : "Revision 2.0.5" ;
SiO2_dielectric_constant Simple Attribute

Use the SiO2_dielectric_constant attribute to specify the relative permittivity of SiO2 that is to be used to calculate sidewall capacitance.

You determine the dielectric unit by dividing the unit for measuring capacitance by the unit for measuring distance. For example,

\[
\text{dielectric} = \frac{\text{capacitance unit}}{\text{distance unit}}
\]

Syntax

\[
\text{phys_library}(\text{library_name_id}) \{ \\
\ldots \\
\text{SiO2_dielectric_constant} : "value\_float"; \\
\ldots \\
\}
\]

value

A floating-point number representing the constant.

Example

SiO2_dielectric_constant : 3.9 ;

time_conversion_factor Simple Attribute

The time_conversion_factor attribute specifies the factor to use for time conversions.

Syntax

\[
\text{phys_library}(\text{library_name_id}) \{ \\
\ldots \\
\text{time_conversion_factor} : \text{value}\_int; \\
\ldots \\
\}
\]

value

Valid values are any positive integer.

Example

time_conversion_factor : 100 ;

time_unit Simple Attribute

The time_unit attribute specifies the unit for time.
Syntax

\[
\text{phys_library(library_nameid) }
\{ \\
\quad \text{time_unit : valueenum ;} \\
\quad \} \\
\]

value

Valid values are 1ns, 100ps, 10ps, and 1ps.

Example

time_unit : 100 ;

**voltage_conversion_factor Simple Attribute**

The `voltage_conversion_factor` attribute specifies the factor to use for voltage conversions.

Syntax

\[
\text{phys_library(library_nameid) }
\{ \\
\quad \text{voltage_conversion_factor : valueint ;} \\
\quad \} \\
\]

value

Valid values are any positive integer.

Example

voltage_conversion_factor : 100 ;

**voltage_unit Simple Attribute**

The `voltage_unit` attribute specifies the unit for voltage.

Syntax

\[
\text{phys_library(library_nameid) }
\{ \\
\quad \text{voltage_unit : valueenum ;} \\
\quad \} \\
\]

value

Valid values are 1mv, 10mv, 100mv, and 1v.
Example
voltage_unit : 100 ;

antenna_lut_template Group
The antenna_lut_template group defines the table template used to specify the antenna_ratio table. The antenna_ratio table is a one-dimensional template that accepts only antenna_diffusion_area limit as a valid value.

Syntax
phys_library(library_nameid) {
    ...
    antenna_lut_template (template_nameid) {
        ...description...
    }
    ...
}

template_name
The name of this lookup table template.

Example
antenna_lut_template (antenna_template_1) {
    ...
}

Simple Attribute
variable_1

Complex Attribute
index_1

variable_1 Simple Attribute
The variable_1 attribute specifies the antenna diffusion area.

Syntax
phys_library(library_nameid) {
    ...
    antenna_lut_template (template_nameid) {
        variable_1 : variable_nameid ;
        ...
    }
    ...
}
variable_name

The only valid value for variable_1 is antenna_diffusion_area.

Example
antenna_lut_template (antenna_template_1) {
    variable_1 : antenna_diffusion_area ;
}

index_1 Complex Attribute

The index_1 attribute specifies the default indexes.

Syntax
phys_library(library_name_id) {
    ...
    antenna_lut_template(template_name_id) {
        index_1 (value_float , value_float , value_float , ...);
        ...
    }
    ...
}

value, value, value, ...

Floating-point numbers that represent the default indexes.

Example
antenna_lut_template (antenna_template_1) {
    index_1 (0.0, 0.159, 0.16) ;
}

resistance_lut_template Group

The resistance_lut_template group defines the template referenced by the resistance_table group.

Syntax
phys_library(library_name_id) {
    ...
    resistance_lut_template (template_name_id) {
        ...
            description ...
        }
    ...
}

template_name

The name of this lookup table template.
Example
resistance_lut_template (resistance_template_1) {
  ...
}

Simple Attributes
variable_1
variable_2

Complex Attributes
index_1
index_2

variable_1 and variable_2 Simple Attributes
Use these attributes to specify whether the variable represents the routing width or the routing spacing.

Syntax
phys_library(library_name_id) {
  ...
  resistance_lut_template (template_name_id) {
    variable_1 : routing_type_id;
    variable_2 : routing_type_id;
    ...
  }
  ...
}

routing_type
Valid values are routing_width and routing_spacing. The values for variable_1 and variable_2 must be different.

index_1 and index_2 Complex Attributes
Use these attributes to specify the default indexes.

Syntax
phys_library(library_name_id) {
  ...
  resistance_lut_template (template_name_id) {
    ...
    index_1 (value_float, value_float, value_float, ...);
    index_2 (value_float, value_float, value_float, ...);
    ...
  }
}
Floating-point numbers that represent the default indexes.

**Example**

```plaintext
class resistance_lut_template (resistance_template_1) {
  variable_1 : routing_width;
  variable_2 : routing_spacing;
  index_1 (0.2, 0.4, 0.6, 0.8);
  index_2 (0.1, 0.3, 0.5, 0.7);
}
```

**shrinkage_lut_template Group**

The `shrinkage_lut_template` group defines the template referenced by the `shrinkage_table` group.

**Syntax**

```plaintext
class phys_library(library_nameid) {
  ...
  shrinkage_lut_template (template_nameid) {
    ...
description...
  }
  ...
}
```

**Template Name**

The name of this lookup table template.

**Example**

```plaintext
class shrinkage_lut_template (shrinkage_template_1) {
  ...
}
```

**Simple Attributes**

```plaintext
variable_1
variable_2
```

**Complex Attributes**

```plaintext
index_1
index_2
```
variable_1 and variable_2 Simple Attributes

Use these attributes to specify whether the variable represents the routing width or the routing spacing.

Syntax

```plaintext
phys_library(library_nameid) {  
   ...  
   shrinkage_lut_template (template_nameid) {  
      variable_1 : routing_typeid ;  
      variable_2 : routing_typeid ;  
   }  
   ... 
}  
```

`routing_type`

Valid values are `routing_width` and `routing_spacing`. The values for `variable_1` and `variable_2` must be different.

index_1 and index_2 Complex Attributes

Use these attributes to specify the default indexes.

Syntax

```plaintext
phys_library(library_nameid) {  
   ...  
   shrinkage_lut_template (template_nameid) {  
      ...  
      index_1 (valuefloat , valuefloat , valuefloat , ...);  
      index_2 (valuefloat , valuefloat , valuefloat , ...);  
   }  
   ... 
}  
```

`value, value, value, ...`

Floating-point numbers that represent the default indexes.

Example

```plaintext
shrinkage_lut_template (resistance_template_1) {  
   variable_1 : routing_width ;  
   variable_2 : routing_spacing ;  
   index_1 (0.3, 0.7, 0.8, 1.2);  
   index_2 (0.2, 0.4, 0.9, 1.1);  
}  
```
spacing_lut_template Group

The `spacing_lut_template` group defines the template referenced by the `spacing_table` group.

Syntax
```
phys_library(library_nameid) {

    ... spacing_lut_template (template_nameid) {
        ... description...
    }

    ...
}
```

`template_name`

The name of this lookup table template.

Example
```
spacing_lut_template (spacing_template_1) {

    ...
}
```

Simple Attributes

`variable_1`
`variable_2`
`variable_3`

Complex Attributes

`index_1`
`index_2`
`index_3`

`variable_1`, `variable_2`, and `variable_3` Simple Attributes

Use these attributes to specify whether the variable represents the routing width or the routing spacing.

Syntax
```
phys_library(library_nameid) {

    ... spacing_lut_template (template_nameid) {
        variable_1 : routing_typeid;
        variable_2 : routing_typeid;
        variable_3 : routing_typeid;

        ...
    }

    ...
}
```
The valid value for variable_1 is routing_width. The valid values for variable_2 are routing_width and routing_length. The valid value for variable_3 is routing_length.

**index_1, index_2, and index_3 Complex Attributes**

Use these attributes to specify the default indexes.

**Syntax**

```
phys_library(library_nameid) {
...
  spacing_lut_template (template_nameid) {
    ...
    index_1 (value_float, value_float, value_float, ...);
    index_2 (value_float, value_float, value_float, ...);
    index_3 (value_float, value_float, value_float, ...);
    ...
  }
...
}
```

Floating-point numbers that represent the default indexes.

**Example**

```
spacing_lut_template (resistance_template_1) {
  variable_1 : routing_width ;
  variable_2 : routing_width ;
  variable_3 : routing_length ;
  index_1 (0.3, 0.6, 0.9, 1.2);
  index_2 (0.3, 0.6, 0.9, 1.2);
  index_2 (1.2, 2.4, 3.8, 5.0);
}
```

**wire_lut_template Group**

The `wire_lut_template` group defines the template referenced by the `wire_extension_range_table` group.

**Syntax**

```
phys_library(library_nameid) {
  ...
}
```
wire_lut_template (template_nameid) {
  ...description...
  ...
}

template_name
The name of this lookup table template.

Example
wire_lut_template (wire_template_1) {
  ...
}

Simple Attributes
variable_1
variable_2
variable_3

Complex Attributes
index_1
index_2
index_3

variable_1, variable_2, and variable_3 Simple Attributes
Use these attributes to specify the routing widths and lengths.

Syntax
phys_library(library_nameid) {
  ...
  wire_lut_template (template_nameid) {
    variable_1 : routing_typeid ;
    variable_2 : routing_typeid ;
    variable_3 : routing_typeid ;
    ...
  }
  ...
}

routing_type
The valid values for variable_1 and variable_2 are routing_width, routing_length, top_routing_width, bottom_routing_width, extension_width, and extension_length. The valid values for variable_3 are routing_width, routing_length, extension_width, and extension_length.
**index_1, index_2, and index_3 Complex Attributes**

Use these attributes to specify the default indexes.

**Syntax**

```plaintext
phys_library(library_nameid) {
    ...
    wire_lut_template (template_nameid) {
        ...
        index_1 (valuefloat, valuefloat, valuefloat, ...);
        index_2 (valuefloat, valuefloat, valuefloat, ...);
        index_3 (valuefloat, valuefloat, valuefloat, ...);
        ...
    }
    ...
}
```

*value, value, value, ...*

Floating-point numbers that represent the default indexes.

**Example**

```plaintext
wire_lut_template (resistance_template_1) {
    variable_1 : routing_width ;
    variable_2 : routing_width ;
    variable_3 : routing_length ;
    index_1 (0.3, 0.6, 0.9, 1.2);
    index_2 (0.3, 0.6, 0.9, 1.2);
    index_2 (1.2, 2.4, 3.8, 5.0);
}
```
You use the resource group to specify the process architecture (standard cell or array) and to specify the layer information (such as routing or contact layer). The resource group is defined inside the phys_library group and must be defined before you model any cell.

The information in this chapter includes a description and syntax example for the attributes that you can define within the resource group.
Syntax for Attributes in the resource Group

The following sections describe the syntax for the attributes you need to define in the resource group. The syntax for the groups you can define within the resource group are described in Chapter 3.

resource Group

The resource group specifies the process architecture class. You must define a resource group before you define any macro group. Also, you can have only one resource group in a physical library.

Syntax

```
phys_library(library_nameid) {
 resource(architecture_enum) {
     ...
 }
}
```

architecture

Valid values are std_cell (standard cell technology) and array (gate array technology).

Example

```
resource(std_cell) {
     ...
 }
```

Complex Attributes

- contact_layer
- device_layer
- overlap_layer
- substrate_layer

Note:

You must specify the layer definition from the substrate out; that is, from the layer closest to the substrate out to the layer farthest from the substrate. You use the following attributes and groups to specify the layer definition:

Attributes: contact_layer, device_layer, and overlap_layer

Groups: poly_layer, and routing_layer.

Groups

- array
- cont_layer
implant_layer
ndiff_layer
pdiff_layer
poly_layer
routing_layer
routing_wire_model
site
tile
via

**contact_layer Complex Attribute**

The `contact_layer` attribute defines the contact cut layer that enables current to flow between the device and the first routing layer, or between any two routing layers.

**Syntax**

```plaintext
phys_library(library_name_id) {
  ...
  resource(architecture_enum) {
    ...
    contact_layer(layer_name_id) ;
    ...
  }
}
```

*layer_name*

The name of the contact layer.

**Example**

```plaintext
contact_layer(cut01) ;
```

**device_layer Complex Attribute**

The `device_layer` attribute specifies the layers that are fixed in the base array.

**Syntax**

```plaintext
phys_library(library_name_id) {
  ...
  resource(architecture_enum) {
    ...
    device_layer(layer_name_id) ;
    ...
  }
}
```

*layer_name*

The name of the device layer.
Example
device_layer(poly) ;

**overlap_layer Complex Attribute**
The `overlap_layer` attribute specifies a layer for describing a rectilinear footprint of a cell.

**Syntax**

```
phys_library(library_name_id) {
  ...
  resource(architecture_enum) {
    ...
    overlap_layer(layer_name_id) ;
    ...
  }
}
```

*layer_name*

The name of the overlap layer.

Example

```
overlap_layer(ovlp1) ;
```

**substrate_layer Complex Attribute**
The `substrate_layer` attribute specifies a substrate layer.

**Syntax**

```
phys_library(library_name_id) {
  ...
  resource(architecture_enum) {
    ...
    substrate_layer(layer_name_id) ;
    ...
  }
}
```

*layer_name*

The name of the substrate layer.

Example

```
substrate_layer(ovlp1) ;
```
You use the resource group to specify the process architecture (standard cell or array) and to specify the layer information (such as routing or contact layer). The resource group is defined inside the phys_library group and must be defined before you model any cell.

This chapter describes the following groups:

- array Group
- cont_layer Group
- implant_layer Group
- ndiff_layer Group
- pdiff_layer Group
- poly_layer Group
- routing_layer Group
- routing_wire_model Group
- site Group
- tile Group
- via Group
- via_array_rule Group
Syntax for Groups in the resource Group

The following sections describe the groups you define in the resource group.

array Group
Use this group to specify the base array for a gate array architecture.

Syntax
phys_library(library_nameid) {
resource(architectureenum) {
  array(array_nameid) {
    ...
  }
}
}

array_name
Specifies a name for the base array.

Note:
Standard cell technologies do not contain array definitions.

Example
array(ar1) {
  ...
}

Groups
floorplan
routing_grid
tracks

floorplan Group
Use this group to specify the arrangement of sites in your design.

Syntax
phys_library(library_nameid) {
resource(architectureenum) {
  array(array_nameid) {
    ...
  }
}
}
floorplan(floorplan_nameid) {
    ...
}
Complex Attribute
- iterate
- origin
- placement_rule

orientation Simple Attribute

The **orientation** attribute specifies the site orientation when placed on the floorplan.

**Syntax**

```
phys_library(library_nameid) {
  resource(architecture_enum) {
    array(array_nameid) {
      floorplan(floorplan_nameid) {
        site_array(site_nameid) {
          orientation : value_enum ;
          ...
        }
      }
    }
  }
}
```

**value**

Valid values are N (north), E (east), S (south), W (west), FN (flip north), FE (flip east), FS (flip south), and FW (flip west), as shown in **Figure 3-1**.

**Figure 3-1  Orientation Examples**

![Figure 3-1 Orientation Examples](image_url)
Example
orientation : E ;

iterate Complex Attribute
The iterate attribute specifies how many times to iterate the site from the specified origin.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    array(array_nameid) {
      floorplan(floorplan_nameid) {
        site_array(site_nameid) {
          iterate(num_x_int, num_y_int, space_x_float, space_y_float) ;
          ...
        }
        ...
      }
    }
  }
}

num_x, num_y
Floating-point numbers that represent the x and y iteration values.

space_x, space_y
Floating-point numbers that represent the spacing values.

Example
iterate(20, 40, 55.200, 16.100) ;

origin Complex Attribute
The origin attribute specifies the point in the floorplan where you can place the first instance of your array.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    array(array_nameid) {
      floorplan(floorplan_nameid) {
        site_array(site_nameid) {
          origin(num_x_float, num_y_float) ;
          ...
        }
      }
    }
  }
}
num_x, num_y

Floating-point numbers that specify the x- and y-coordinates for the starting point of your array.

Example

```
origin(-1.00, -1.00) ;
```

placement_rule Complex Attribute

The placement_rule attribute specifies whether you can place an instance on the specified site array.

Syntax

```
phys_library(library_nameid) {
resource(architecture_enum) {
    array(array_nameid) {
        floorplan(floorplan_nameid) {
            site_array(site_nameid) {
                placement_rule : value_enum ;
                ...
            }
        }
    }
}
```

value

Valid values are regular, can_place, and cannot_occupy.

where

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>Base array of sites occupying the floorplan.</td>
</tr>
<tr>
<td>can_place</td>
<td>Sites are available for placement.</td>
</tr>
<tr>
<td>cannot_occupy</td>
<td>Sites are not available for placement.</td>
</tr>
</tbody>
</table>

Example

```
placement_rule : can_place ;
```

**routing_grid Group**

Use this group to specify the global cell grid overlaying the array, as shown in Figure 3-2. If you do not specify a routing grid, the default grid is used.

*Figure 3-2  A Routing Grid*

```
Syntex
phys_library(library_nameid) {
 resource(architecture_enum) {
  array(array_nameid) {
   routing_grid() {
    routing_direction : value_enum ;
    grid_pattern(start_float, grids_int, space_float) ;
   }
  }
 }
}

Example
routing_grid() {
  ...
}
```

**Simple Attribute**

*routing_direction*

**Complex Attribute**

*grid_pattern*
routing_direction Simple Attribute

The `routing_direction` attribute specifies the preferred grid routing direction.

Syntax

```
phys_library(library_name_id) {
resource(architecture_enum) {
    array(array_name_id) {
        routing_grid() {
            routing_direction : value_enum ;
            ...
        }
    }
}
}
```

value

Valid values are `horizontal` and `vertical`.

Example

```
routing_direction : horizontal ;
```

grid_pattern Complex Attribute

The `grid_pattern` attribute specifies the global cell grid pattern.

Syntax

```
phys_library(library_name_id) {
resource(architecture_enum) {
    array(array_name_id) {
        routing_grid() {
            grid_pattern(start_float, grids_int, space_float) ;
            ...
        }
    }
}
}
```

start

A floating-point number that represents the grid starting point.

grids

A number that represents the number of grids in the specified routing direction.

space

A floating-point number that represents the spacing between the respective grids.
Example
grid pattern(1.0, 100, 2.0)

tracks Group
Use this group to specify the routing track grid for the gate array.

Syntax
phys_library(library_nameid) {
  resource(architectureenum) {
    array(array_nameid) {
      tracks() {
        ...
      }
    }
  }
}

Note:
You must define at least one track group for horizontal routing and one group for vertical routing.

Simple Attributes
layers
routing_direction

Complex Attribute
track_pattern

layers Simple Attribute
The layers attribute specifies a list of layers available for the tracks.

Syntax
phys_library(library_nameid) {
  resource(architectureenum) {
    array(array_nameid) {
      tracks() {
        layers: "layer1_nameid, layer2_nameid, ...
        ..., layern_nameid" ;
        ...
      }
    }
  }
}
layer1_name, layer2_name, ..., layern_name
A list of layer names.

Example
layers: "m1, m3" ;

routing_direction Simple Attribute
The routing_direction attribute specifies the track direction and the possible routing direction.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
array(array_nameid) {
tracks() {
...
routing_direction:value_enum ;
...
}
}
}

value
Valid values are horizontal and vertical.

Example
routing_direction: horizontal ;

track_pattern Complex Attribute
The track_pattern attribute specifies the track pattern.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
array(array_nameid) {
tracks() {
...
track_pattern(start_float, tracks_int, spacing_float) ;
}
}
}
}
**start, tracks, spacing**

Specifies the starting-point coordinate, the number of tracks, and the space between the tracks, respectively.

**Example**

```plaintext
track_pattern (1.40, 50, 10.5);
```

---

**cont_layer Group**

Use this group to specify values for the contact layer.

**Syntax**

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    cont_layer(layer_name_id) {
      ...
    }
  }
}
```

**layer_name**

The name of the contact layer.

**Example**

```plaintext
cont_layer() {
  ...
}
```

**Simple Attributes**

- corner_min_spacing
- max_stack_level
- spacing

**Groups**

- enclosed_via_rules
- max_current_ac_absavg
- max_current_ac_avg
- max_current_ac_peak
- max_current_ac_rms
- max_current_dc_avg
corner_min_spacing Simple Attribute

The `corner_min_spacing` attribute specifies the minimum spacing allowed between two vias when their corners point to each other; otherwise specifies the minimum edge-to-edge spacing.

Note:
The `corner_min_spacing` complex attribute in the `topological_design_rules` group specifies the minimum distance between two contact layers.

Syntax

```plaintext
phys_library(library_name_id) {
  ... 
  resource(architecture_enum) {
    cont_layer() {
      ...
      corner_min_spacing : value_float;
    }
  }
}
```

**value**

A positive floating-point number representing the spacing value.

Example

```plaintext
corner_min_spacing : 0.0;
```

max_stack_level Simple Attribute

The `max_stack_level` attribute specifies a value for the maximum number of stacked vias.

Syntax

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    cont_layer() {
      ...
      max_stack_level : value_int;
    }
  }
}
```

**value**

An integer representing the stack level.
Example
max_stack_level : 2 ;

**spacing Simple Attribute**
Defines the minimum separation distance between the edges of objects on the layer when the objects are on different nets.

**Syntax**

```plaintext
phys_library(library_nameid) {
  ...
  resource(architecture_enum) {
    cont_layer () {
      ...
      spacing : value float ;
      ...
    }
  }
}
```

**value**
A positive floating-point number representing the minimum spacing value.

Example

```plaintext
spacing : 0.0 ;
```

**enclosed_cut_rule Group**
Use this group to specify the rules for cuts in the middle of the cut array.

**Syntax**

```plaintext
phys_library(library_nameid) {
  resource(architecture_enum) {
    cont_layer () {
      ...
      enclosed_cut_rule() {
        ...
      }
    }
  }
}
```

**Simple Attributes**

- max_cuts
- max_neighbor_cut_spacing
- min_cuts
- min_enclosed_cut_spacing
min_neighbor_cut_spacing

**max_cuts Simple Attribute**

The `max_cuts` attribute specifies the maximum number of neighboring cuts allowed within a specified space (range).

**Syntax**

```plaintext
phys_library(library_nameid) {
  resource(architecture_enum) {
    cont_layer() {
      enclosed_cut_rule(layer_nameid) {
        max_cuts : value_float ;
        ...
      }
    }
  }
}
```

**value**

A floating-point number representing the number of cuts.

**Example**

```plaintext
max_cuts : 0.0 ;
```

**max_neighbor_cut_spacing Simple Attribute**

The `max_neighbor_cut_spacing` attribute specifies the spacing (range) around the cut on the perimeter of the array.

**Syntax**

```plaintext
phys_library(library_nameid) {
  resource(architecture_enum) {
    cont_layer() {
      enclosed_cut_rule(layer_nameid) {
        max_neighbor_cut_spacing : value_float ;
        ...
      }
    }
  }
}
```

**value**

A floating-point number representing the spacing.

**Example**

```plaintext
max_neighbor_cut_spacing : 0.0 ;
```
**min_cuts Simple Attribute**

The `min_cuts` attribute specifies the minimum number of neighboring cuts allowed within a specified space (range).

**Syntax**

```plaintext
phys_library(library_nameid) {
  resource(architecture_enum) {
    cont_layer () {
      enclosed_cut_rule(layer_nameid) {
        min_cuts : value_float ;
        ...
      }
    }
  }
}
```

**value**

A floating-point number representing the number of cuts.

**Example**

```plaintext
min_cuts : 0.0 ;
```

**min_enclosed_cut_spacing Simple Attribute**

The `min_enclosed_cut_spacing` attribute specifies the spacing (range) around the cut on the perimeter of the array.

**Syntax**

```plaintext
phys_library(library_nameid) {
  resource(architecture_enum) {
    cont_layer () {
      enclosed_cut_rule(layer_nameid) {
        min_enclosed_cut_spacing : value_float ;
        ...
      }
    }
  }
}
```

**value**

A floating-point number representing the spacing.

**Example**

```plaintext
min_enclosed_via_spacing : 0.0 ;
```
min_neighbor_cut_spacing Simple Attribute

The min_neighbor_cut_spacing attribute specifies minimum spacing around the

Syntax

```
phys_library(library_name_id) {
  resource(architecture_enum) {
    cont_layer() {
      enclosed_cut_rule(layer_name_id) {
        min_neighbor_via_spacing : value_float;
        ...}
      }
    }
  }
}
```

value

- A floating-point number representing the spacing around the cut on the perimeter of the array.

Example

- `min_neighbor_cut_spacing : 0.0 ;`

max_current_ac_absavg Group

Use this group to specify the absolute average value for the AC current that can pass through a cut.

Syntax

```
phys_library(library_name_id) {
  resource(architecture_enum) {
    cont_layer() {
      ... 
      max_current_ac_absavg(template_name_id) {
        ... 
      }
    }
  }
}
```

*template_name*

- The name of the contact layer.

Example

- `max_current_ac_absavg() { 
  ... 
  }`
Complex Attributes
index_1
index_2
index_3
values

max_current_ac_avg Group
Use this group to specify an average value for the AC current that can pass through a cut.

Syntax
phys_library(library_name_id) {
  resource(architecture_enum) {
    cont_layer () {
      ...
      max_current_ac_avg(template_name_id) {
        ...
      }
    }
  }
}

template_name
  The name of the contact layer.

Example
max_current_ac_avg() {
  ...
}

Complex Attributes
index_1
index_2
index_3
values

max_current_ac_peak Group
Use this group to specify a peak value for the AC current that can pass through a cut.

Syntax
phys_library(library_name_id) {
  resource(architecture_enum) {
    cont_layer () {
      ...
    }
  }
}
max_current_ac_peak(template_name_id) {  
    ...
}  
}  
}

template_name
The name of the contact layer.

Example
max_current_ac_peak() {  
    ...
}  
}

Complex Attributes
index_1
index_2
index_3
values

max_current_ac_rms Group
Use this group to specify a root mean square value for the AC current that can pass through a cut.

Syntax
phys_library(library_name_id) {  
    resource(architecture_enum) {  
        cont_layer() {  
            ...
        }  
    }  
    max_current_ac_rms(template_name_id) {  
        ...
    }
}  
}

template_name
The name of the contact layer.

Example
max_current_ac_rms() {  
    ...
}  
}
Complex Attributes
index_1
index_2
index_3
values

max_current_dc_avg Group
Use this group to specify an average value for the DC current that can pass through a cut.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
  cont_layer () {
    ...
  }
  max_current_dc_avg(template_nameid) {
    ...
  }
}
}

template_name
  The name of the contact layer.

Example
max_current_dc_avg() {
  ...
}

Complex Attributes
index_1
index_2
values

implant_layer Group
Use this group to specify the legal placement rules when mixing high drive and low drive cells in the detail placement.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
  implant_layer(layer_nameid) {
    ...
  }
}
}
layer_name

The name of the implant layer.

**Simple Attributes**

- `min_width`
- `spacing`

**Complex Attribute**

- `spacing_from_layer`

### `min_width` Simple Attribute

The `min_width` attribute specifies the minimum width of any dimension of an object on the layer.

**Syntax**

```plaintext
phys_library(library_nameid) {
    resource(architecture_enum) {
        implant_layer(layer_nameid) {
            min_width : value_float;
            . . .
        }
    }
}
```

**value**

A floating-point number representing the width.

**Example**

```plaintext
min_width : 0.0 ;
```

### `spacing` Simple Attribute

The `spacing` attribute specifies the separation distance between the edges of objects on the layer when the objects are on different nets.

**Syntax**

```plaintext
phys_library(library_nameid) {
    resource(architecture_enum) {
        implant_layer(layer_nameid) {
            spacing : value_float;
            . . .
        }
    }
}
```
value
A floating-point number representing the spacing.

Example
spacing : 0.0 ;

**spacing_from_layer Complex Attribute**

The *spacing_from_layer* attribute specifies the minimum allowable spacing between two geometries on the layer.

**Syntax**

```plaintext
phys_library(library_nameid) {
    resource(architecture_enum) {
        implant_layer(layer_nameid) {
            spacing_from_layer (value_float, name_id );
        }
    }
}
```

value
A floating-point number representing the spacing.

name
A layer name.

Example

```plaintext
spacing_from_layer () ;
```

---

**ndiff_layer Group**

Use the *ndiff_layer* group to specify the maximum current values for the n-diffusion layer.

**max_current_ac_absavg Group**

Use this group to specify the absolute average value for the AC current that can pass through a cut.
Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    ndiff_layer () {
      ...
    }
  }
  max_current_ac_absavg(template_nameid) {
    ...
  }
}

template_name
  The name of the contact layer.

Example
max_current_ac_absavg() {
  ...
}

Complex Attributes
  index_1
  index_2
  index_3
  values

max_current_ac_avg Group
Use this group to specify an average value for the AC current that can pass through a cut.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    ndiff_layer () {
      ...
    }
    max_current_ac_avg(template_nameid) {
      ...
    }
  }
}

template_name
  The name of the contact layer.

Example
max_current_ac_avg() {
max_current_ac_peak Group

Use this group to specify a peak value for the AC current that can pass through a cut.

Syntax

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    ndiff_layer () {
      ...
    }
    max_current_ac_peak(template_name_id) {
      ...
    }
  }
}
```

template_name

The name of the contact layer.

Example

```plaintext
max_current_ac_peak() {
  ...
}
```

max_current_ac_rms Group

Use this group to specify a root mean square value for the AC current that can pass through a cut.

Syntax

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    ...
  }
}
```
Chapter 3: Specifying Groups in the resource Group

Syntax for Groups in the resource Group

ndiff_layer () {
   ...
   max_current_ac_rms(template_name_id) {
      ...
   }
}
}

template_name

   The name of the contact layer.

Example

max_current_ac_rms() {
   ...
}

Complex Attributes

index_1
index_2
index_3
values

max_current_dc_avg Group

Use this group to specify an average value for the DC current that can pass through a cut.

Syntax

phys_library(library_name_id) {
   resource(architecture_enum) {
      ndiff_layer () {
         ...
         max_current_dc_avg(template_name_id) {
            ...
         }
      }
   }
}

template_name

   The name of the contact layer.

Example

max_current_dc_avg() {
   ...
}
Complex Attributes
  index_1
  index_2
  values

pdiff_layer Group
Use the pdiff_layer group to specify the maximum current values for the p-diffusion layer.

max_current_ac_absavg Group
Use this group to specify the absolute average value for the AC current that can pass through a cut.

Syntax
phys_library(library_name_id) {
resource(architecture_enum) {
  pdiff_layer () {
    ...
  }
  max_current_ac_absavg(template_name_id) {
    ...
  }
}
}

template_name
  The name of the contact layer.

Example
max_current_ac_absavg() {
  ...
}

Complex Attributes
  index_1
  index_2
  index_3
  values

max_current_ac_avg Group
Use this group to specify an average value for the AC current that can pass through a cut.
Syntax

```
phys_library(library_name_id) {
resource(architecture_enum) {
pdiff_layer() {
...
max_current_ac_avg(template_name_id) {
...
}
}
}
}
```

template_name

The name of the contact layer.

Example

```
max_current_ac_avg() {
...
}
```

Complex Attributes

index_1
index_2
index_3
values

max_current_ac_peak Group

Use this group to specify a peak value for the AC current that can pass through a cut.

Syntax

```
phys_library(library_name_id) {
resource(architecture_enum) {
pdiff_layer() {
...
max_current_ac_peak(template_name_id) {
...
}
}
}
```

template_name

The name of the contact layer.

Example

```
max_current_ac_peak() {
```
Complex Attributes
index_1
index_2
index_3
values

max_current_ac_rms Group
Use this group to specify a root mean square value for the AC current that can pass through a cut.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    pdiff_layer () {
    ...
      max_current_ac_rms(template_nameid) {
    ...
    }
  }
}
}

template_name
The name of the contact layer.

Example
max_current_ac_rms() {
  ...
}

Complex Attributes
index_1
index_2
index_3
values

max_current_dc_avg Group
Use this group to specify an average value for the DC current that can pass through a cut.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    pdiff_layer () {
    ...
      max_current_dc_avg(template_nameid) {
    ...
    }
  }
}
}
pdiff_layer() {
  ...
  max_current_dc_avg(template_name_id) {
    ...
  }
}

template_name
  The name of the contact layer.

Example
  max_current_dc_avg() {
    ...
  }

Complex Attributes
  index_1
  index_2
  values

---

poly_layer Group

Use this group to specify the poly layer name and properties.

Syntax
  phys_library(library_name_id) {
    resource(architecture_enum) {
      poly_layer(layer_name_id) {
        ...
      }
    }
  }

layer_name
  The name of the poly layer.

Example
  poly_layer() {
    ...
  }

Simple Attributes
  avg_lateral_oxide_permittivity
  avg_lateral_oxide_thickness
height
oxide_permittivity
oxide_thickness
res_per_sq
shrinkage
thickness

Complex Attributes
conformal_lateral_oxide
lateral_oxide

Groups
max_current_ac_absavg
max_current_ac_avg
max_current_ac_peak
max_current_ac_rms
max_current_dc_avg

avg_lateral_oxide_permittivity Simple Attribute
This attribute specifies a value representing the average lateral oxide permittivity.

Syntax
phys_library(library_name_id) {
  resource(architectureenum) {
    poly_layer(layer_name_id) {
      avg_lateral_oxide_permittivity : valuefloat ;
      ...
    }
  }
}

permittivity
A floating-point number that represents the lateral oxide permittivity.

Example
avg_lateral_oxide_permittivity (0.0 ) ;

avg_lateral_oxide_thickness Simple Attribute
This attribute specifies a value representing the average lateral oxide thickness.

Syntax
phys_library(library_name_id) {
  resource(architectureenum) {
    poly_layer(layer_name_id) {

avg_lateral_oxide_thickness : valuefloat ;
...
}
}

thickness
A floating-point number that represents the lateral oxide thickness.

Example
avg_lateral_oxide_thickness (0.0) ;

height Simple Attribute
The height attribute specifies the distance from the top of the substrate to the bottom of the routing layer.

Syntax
phys_library(library_nameid) {
resource(architecture Enum) {
poly_layer(layer_nameid) {
    height : type_name float ;
    ...
}
}
}

type_name
A floating-point number representing the distance.

Example
height : 1.0 ;

oxide_permittivity Simple Attribute
The oxide_permittivity attribute specifies the oxide permittivity for the layer.

Syntax
phys_library(library_nameid) {
resource(architecture Enum) {
poly_layer(layer_nameid) {
    oxide_permittivity : value float ;
    ...
}
}
}
value

A floating-point number representing the permittivity.

Example
oxide_permittivity : 3.9 ;

oxide_thickness Simple Attribute

The oxide_thickness attribute specifies the oxide thickness for the layer.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    poly_layer(layer_nameid) {
      oxide_thickness : value_float ;
      ...
    }
  }
}

float

A floating-point number representing the thickness.

Example
oxide_thickness : 2.0 ;

res_per_sq Simple Attribute

The res_per_sq attribute specifies the resistance unit area of a poly layer.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    poly_layer(layer_nameid) {
      res_per_sq : value_float ;
      ...
    }
  }
}

value

A floating-point number representing the resistance value.

Example
res_per_sq : 1.200e-01 ;
shrinkage Simple Attribute

The \texttt{shrinkage} attribute specifies the total distance by which the wire width on the layer shrinks or expands. The shrinkage parameter is a sum of the shrinkage for each side of the wire. The post-shrinkage wire width represents the final processed silicon width as calculated from the drawn silicon width in the design database.

Note:
Do not specify a value for the \texttt{shrinkage} attribute or \texttt{shrinkage_table} group if you specify a value for the \texttt{process_scale_factor} attribute.

Syntax

\begin{verbatim}
phys_library(library_name_id) {
    resource(architecture_enum) {
        poly_layer(layer_name_id) {
            shrinkage : value_float;
            ...
        }
    }
}
\end{verbatim}

\texttt{value}

A floating-point number representing the distance. A positive number represents shrinkage; a negative number represents expansion.

Example

\begin{verbatim}
shrinkage : 0.00046;
\end{verbatim}

thickness Simple Attribute

The \texttt{thickness} attribute specifies the thickness of the routing layer.

Syntax

\begin{verbatim}
phys_library(library_name_id) {
    resource(architecture_enum) {
        poly_layer(layer_name_id) {
            thickness : value_float;
            ...
        }
    }
}
\end{verbatim}

\texttt{value}

A floating-point number representing the thickness.
**Example**

```
thickness : 0.02 ;
```

**conformal_lateral_oxide Complex Attribute**

The `conformal_lateral_oxide` attribute specifies values for the thickness and permittivity of a layer.

**Syntax**

```
phys_library(library_nameid) {  
  resource(architecture_enum) {  
    poly_layer(layer_nameid) {  
      conformal_lateral_oxide(value_1float, value_2float, \  
        value_3float, value_4float ) ;  
    ...  
  }  
}  
```

- **value_1**
  - A floating-point number that represents the oxide thickness.
- **value_2**
  - A floating-point number that represents the topwall thickness.
- **value_3**
  - A floating-point number that represents the sidewall thickness.
- **value_4**
  - A floating-point number that represents the oxide permittivity.

**Example**

```
conformal_lateral_oxide (0.2, 0.3, 0.21, 3.5) ;
```

**lateral_oxide Complex Attribute**

The `lateral_oxide` attribute specifies values for the thickness and permittivity of a layer.

**Syntax**

```
phys_library(library_nameid) {  
  resource(architecture_enum) {  
    poly_layer(layer_nameid) {  
      lateral_oxide(thicknessfloat, permittivityfloat ) ;  
    ...  
  }  
}  
```
thickness
A floating-point number that represents the oxide thickness.

permittivity
A floating-point number that represents the oxide permittivity.

Example
lateral_oxide (0.024, 3.6);

max_current_ac_absavg Group
Use this group to specify the absolute average value for the AC current that can pass through a cut.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
pdiff () {
...
max_current_ac_absavg(template_nameid) {
  ...
  }
}
}

template_name
The name of the contact layer.

Example
max_current_ac_absavg() {
  ...
}

Complex Attributes
index_1
index_2
index_3
values
max_current_ac_avg Group

Use this group to specify an average value for the AC current that can pass through a cut.

Syntax

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    pdiff () {
      ...
      max_current_ac_avg(template_name_id) {
        ...
      }
    }
  }
}
```

**template_name**

The name of the contact layer.

**Example**

```plaintext
max_current_ac_avg() {
  ...
}
```

**Complex Attributes**

- index_1
- index_2
- index_3
- values

max_current_ac_peak Group

Use this group to specify a peak value for the AC current that can pass through a cut.

Syntax

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    pdiff () {
      ...
      max_current_ac_peak(template_name_id) {
        ...
      }
    }
  }
}
```
**template_name**

The name of the contact layer.

**Example**

```c
max_current_ac_peak() {
    ...
}
```

**Complex Attributes**

- index_1
- index_2
- index_3
- values

---

**max_current_ac_rms Group**

Use this group to specify a root mean square value for the AC current that can pass through a cut.

**Syntax**

```c
def phys_library(library_nameId) {
    def resource(architecture_enum) {
        pdiff () {
            ...
        }
        max_current_ac_rms(template_nameId) {
            ...
        }
    }
}
```

**template_name**

The name of the contact layer.

**Example**

```c
max_current_ac_rms() {
    ...
}
```

**Complex Attributes**

- index_1
- index_2
- index_3
- values
**max_current_dc_avg Group**

Use this group to specify an average value for the DC current that can pass through a cut.

**Syntax**

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    pdiff () {
      ...
    }
    max_current_dc_avg(template_name_id) {
      ...
    }
  }
}
```

*template_name*

The name of the contact layer.

**Example**

```plaintext
max_current_dc_avg() {
  ...
}
```

**Complex Attributes**

- index_1
- index_2
- values

---

**routing_layer Group**

Use this group to specify the routing layer name and properties.

**Syntax**

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      ...
    }
  }
}
```

*layer_name*

The name of the routing layer.
Example

```
routing_layer(m1) {
    ...
}
```

**Simple Attributes**

- `avg_lateral_oxide_permittivity`
- `avg_lateral_oxide_thickness`
- `baseline_temperature`
- `cap_multiplier`
- `cap_per_sq`
- `coupling_cap`
- `default_routing_width`
- `edgecapacitance`
- `field_oxide_permittivity`
- `field_oxide_thickness`
- `fill_active_spacing`
- `fringe_cap`
- `height`
- `inductance_per_dist`
- `max_current_density`
- `max_length`
- `max_observed_spacing_ratio_for_lpe`
- `max_width`
- `min_area`
- `min_enclosed_area`
- `min_enclosed_width`
- `min_fat_wire_width`
- `min_fat_via_width`
- `min_length`
- `min_width`
- `min_wire_split_width`
- `offset`
- `oxide_permittivity`
- `oxide_thickness`
- `pitch`
- `process_scale_factor`
- `res_temperature_coefficient`
- `routing_direction`
- `same_net_min_spacing`
- `shrinkage`
- `spacing`
- `thickness`
- `u_shaped_wire_spacing`
- `wire_extension`
- `wire_extension_range_check_connect_only`
- `wire_extension_range_check_corner_only`

**Complex Attribute**

- `conformal_lateral_oxide`
- `lateral_oxide`
min_extension_width
min_shape_edge
plate_cap
ranged_spacing
spacing_check_style
stub_spacing

Groups
end_of_line_spacing_rule
extension_via_rule
max_current_ac_absavg
max_current_ac_avg
max_current_ac_peak
max_current_ac_rms
max_current_dc_avg
min_edge_rule
min_enclosed_area_table
notch_rule
resistance_table
shrinkage_table
spacing_table
wire_extension_range_table

avg_lateral_oxide_permittivity Simple Attribute
This attribute specifies a value representing the average lateral oxide permittivity.

Syntax
phys_library(library_name_id) {
resource(architecture_enum) {
routing_layer(layer_name_id) {
    avg_lateral_oxide_permittivity : value_float ;
    ...
}
}
}

permittivity
A floating-point number that represents the lateral oxide permittivity.

Example
avg_lateral_oxide_permittivity (0.0 ) ;

avg_lateral_oxide_thickness Simple Attribute
This attribute specifies a value representing the average lateral oxide thickness.
Syntax

phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            avg_lateral_oxide_thickness : value_float;
            ...
        }
    }
}

thickness

A floating-point number that represents the lateral oxide thickness.

Example

avg_lateral_oxide_thickness (0.0) ;

baseline_temperature Simple Attribute

This attribute specifies a baseline operating condition temperature.

Syntax

phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            baseline_temperature : value_float;
            ...
        }
    }
}

value

A floating-point number representing the temperature.

Example

baseline_temperature : 60.0 ;

cap_multiplier Simple Attribute

Use the cap_multiplier attribute to specify a scaling factor for interconnect capacitance to account for changes in capacitance due to nearby wires.

Syntax

phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {

cap_multiplier : value_float;
...
}
}
}

value
A floating-point number representing the scaling factor.

Example
cap_multiplier : 2.0

cap_per_sq Simple Attribute
The cap_per_sq attribute specifies the substrate capacitance per unit area of a routing layer.

Syntax
phys_library(library_name_id) {
resource(architecture_enum) {
routing_layer(layer_name_id) {
    cap_per_sq : value_float;
    ...
}
}
}

value
A floating-point number that represents the capacitance for a square unit of wire, in picofarads per square distance unit.

Example
cap_per_sq : 5.909e-04 ;

coupling_cap Simple Attribute
The coupling_cap attribute specifies the coupling capacitance per unit length between parallel wires on the same layer.

Syntax
phys_library(library_name_id) {
resource(architecture_enum) {
routing_layer(layer_name_id) {
    coupling_cap : value_float;
    ...
}
value

A floating-point number that represents the capacitance value.

Example
coupling_cap: 0.000019 ;

default_routing_width Simple Attribute

The default_routing_width attribute specifies the minimal routing width (default) for wires on the layer.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
routing_layer(layer_nameid) {
    default_routing_width : value_float ;
    ...
}
}
}

value

A positive floating-point number representing the default routing width.

Example
default_routing : 4.400e-01 ;

dgac capacitance Simple Attribute

The edgecapacitance attribute specifies the total peripheral capacitance per unit length of a wire on the routing layer.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
routing_layer(layer_nameid) {
    edgecapacitance : value_float ;
    ...
}
}
}
value
A floating-point number that represents the capacitance per unit length value.

Example
edgecapacitance : 0.00065 ;

field_oxide_permittivity Simple Attribute
The field_oxide_permittivity attribute specifies the relative permittivity of the field oxide.

Syntax
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      field_oxide_permittivity : value_float ;
      ...
    }
  }
}

value
A positive floating-point number representing the relative permittivity.

Example
field_oxide_permittivity : 3.9 ;

field_oxide_thickness Simple Attribute
The field_oxide_thickness attribute specifies the field oxide thickness.

Syntax
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      field_oxide_thickness : value_float ;
      ...
    }
  }
}

value
A positive floating-point number in distance units.
Example
field_oxide_thickness : 0.5 ;

**fill_active_spacing Simple Attribute**

The `fill_active_spacing` attribute specifies the spacing between fill metal and active geometry.

**Syntax**

```plaintext
phys_library(value_float) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            fill_active_spacing : value_float;
            ...
        }
    }
}
```

**value**

A floating-point number that represents the spacing.

**Example**

fill_active_spacing : 0.0 ;

**fringe_cap Simple Attribute**

The `fringe_cap` attribute specifies the fringe (sidewall) capacitance per unit length of a routing layer.

**Syntax**

```plaintext
phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            fringe_cap : value_float;
            ...
        }
    }
}
```

**value**

A floating-point number that represents the capacitance value.

**Example**

fringe_cap : 0.00023 ;
**height Simple Attribute**

The `height` attribute specifies the distance from the top of the substrate to the bottom of the routing layer.

**Syntax**

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      height : value_float;
    }
  }
}
```

**value**

A floating-point number representing the distance.

**Example**

```plaintext
height : 1.0 ;
```

**inductance_per_dist Simple Attribute**

The `inductance_per_dist` attribute specifies the inductance per unit length of a routing layer.

**Syntax**

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      inductance_per_dist : value_float;
    }
  }
}
```

**value**

A floating-point number that represents the inductance value.

**Example**

```plaintext
inductance_per_dist : 0.0029 ;
```

**max_current_density Simple Attribute**

The `max_current_density` attribute specifies the maximum current density for a contact.
Syntax

phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      max_current_density : value_float ;
      ...
    }
  }
}

value

A floating-point number that represents, in amperes per centimeter, the maximum current density the contact can carry.

Example

max_current_density : 0.0 ;

max_length Simple Attribute

The max_length attribute specifies the maximum length of wire segments on the layer.

Syntax

phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      max_length : value_float ;
      ...
    }
  }
}

value

A floating-point number that represents wire segment length.

Example

max_length : 0.0 ;

max_observed_spacing_ratio_for_lpe Simple Attribute

This attribute specifies the maximum wire spacing for layer parasitic extraction (LPE) when calculating intracapacitance.

Use the true spacing value for calculating intracapacitance when the spacing between all wires reflects the following equation:

distances < spacing * max_observed_spacing_ratio_for_lpe
Use a calculated value as shown for calculating intracapacitance when the spacing between all wires reflects the following equation.

\[ \text{distances} > (\text{spacing} \times \text{max}\_\text{observed}\_\text{spacing}\_\text{ratio}\_\text{for}\_\text{lpe}) \]

**Syntax**

\[
\text{phys}\_\text{library}((\text{library}\_\text{name}_id) \{ \\
\text{resource}((\text{architecture}\_\text{enum}) \{ \\
\text{routing}\_\text{layer}((\text{layer}\_\text{name}_id) \{ \\
\text{max}\_\text{observed}\_\text{spacing}\_\text{ratio}\_\text{for}\_\text{lpe} : \text{value}_\text{float} ; \\
\text{...} \\
\} \\
) \\
) \\
\}) \\
\}
\]

**value**

A floating-point number that represents the distance.

**Example**

```
max\_\text{observed}\_\text{spacing}\_\text{ratio}\_\text{for}\_\text{lpe} : 3.0 ;
```

**max\_\text{width} Simple Attribute**

The **max\_\text{width}** attribute specifies the maximum width of wire segments on the layer for DRC.

**Syntax**

\[
\text{phys}\_\text{library}((\text{library}\_\text{name}_id) \{ \\
\text{resource}((\text{architecture}\_\text{enum}) \{ \\
\text{routing}\_\text{layer}((\text{layer}\_\text{name}_id) \{ \\
\text{max}\_\text{width} : \text{value}_\text{float} ; \\
\text{...} \\
\} \\
) \\
) \\
\}) \\
\}
\]

**value**

A floating-point number that represents wire segment width.

**Example**

```
max\_\text{width} : 0.0 ;
```

**min\_\text{area} Simple Attribute**

The **min\_\text{area}** attribute specifies the minimum metal area for the given routing layer.
Syntax

phys_library(library_nameid) {
    resource(architectureenum) {
        routing_layer(layer_nameid) {
            min_area : valuefloat;
            ...
        }
    }
}

value

A floating-point number that represents the minimum metal area.

Example

min_area : 0.0;

min_enclosed_area Simple Attribute

The min_enclosed_area attribute specifies the minimum metal area, enclosed by ring-shaped wires or vias, for the given routing layer.

Syntax

phys_library(library_nameid) {
    resource(architectureenum) {
        routing_layer(layer_nameid) {
            min_enclosed_area : valuefloat;
            ...
        }
    }
}

value

A floating-point number that represents the minimum metal area.

Example

min_enclosed_area : 0.14;

min_enclosed_width Simple Attribute

The min_enclosed_width attribute specifies the minimum metal width for the given routing layer.

Syntax

phys_library(library_nameid) {
    resource(architectureenum) {

routing_layer(layer_nameid) {
    min_enclosed_width : value float ;
    ...
}
}

value
A floating-point number that represents the minimum metal width.

Example
min_enclosed_width : 0.14 ;

min_fat_wire_width Simple Attribute
The min_fat_wire_width attribute specifies the minimal wire width that defines whether a wire is a fat wire.

Syntax
phys_library(library_nameid) {
    resource(architecture enum) {
        routing_layer(layer_nameid) {
            min_fat_wire_width : value float ;
            ...
        }
    }
}

value
A floating-point number that represents the minimal wire width.

Example
min_fat_wire_width : 0.0 ;

min_fat_via_width Simple Attribute
The min_fat_via_width attribute specifies a threshold value for using the fat wire spacing rule instead of the default spacing rule.

Syntax
phys_library(library_nameid) {
    resource(architecture enum) {
        routing_layer(layer_nameid) {
            min_fat_via_width : value float ;
            ...
        }
    }
}
value

A floating-point number that represents the threshold value.

Example

min_fat_via_width : 0.0 ;

min_length Simple Attribute

The min_length attribute specifies the minimum length of wire segments on the layer for DRC.

Syntax

phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      min_length : value_float;
      ...
    }
  }
}

value

A floating-point number that represents the minimum wire segment length.

Example

min_length : 0.202 ;

min_width Simple Attribute

The min_width attribute specifies the minimum width of wire segments on the layer for DRC.

Syntax

phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      min_width : value_float;
      ...
    }
  }
}
value

A floating-point number that represents the minimum wire segment width.

Example

min_width : 0.202 ;

min_wire_split_width Simple Attribute

This attribute specifies the minimum wire width for split wires.

Syntax

phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      min_wire_split_width : value_float ;
      ...
    }
  }
}

value

A floating-point number that represents the minimum wire split width.

Example

min_wire_split_width : 0.202 ;

offset Simple Attribute

The offset attribute specifies the offset distance from the placement grid to the routing grid. The default is one half the routing pitch value.

Syntax

phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      offset : value_float ;
      ...
    }
  }
}

value

A floating-point number representing the distance.
Example
offset : 0.0025 ;

oxide_permittivity Simple Attribute
The oxide_permittivity attribute specifies the permittivity for the layer.

Syntax
phys_library(library_nameid) {
resource(architectureenum) {
routing_layer(layer_nameid) {
   oxide_permittivity : valuefloat ;
   ...
}
}
}

value
A floating-point number representing the permittivity.

Example
oxide_permittivity : 3.9 ;

oxide_thickness Simple Attribute
The oxide_thickness attribute specifies the oxide thickness for the layer.

Syntax
phys_library(library_nameid) {
resource(architectureenum) {
routing_layer(layer_nameid) {
   oxide_thickness : valuefloat ;
   ...
}
}
}

value
A floating-point number representing the thickness.

Example
oxide_thickness : 1.33 ;
pitch Simple Attribute

The *pitch* attribute specifies the track distance (center point to center point) of the detail routing grid for a standard-cell routing layer.

**Syntax**

```plaintext
phys_library(library_nameid) {
    resource(architecture_enum) {
        routing_layer(layer_nameid) {
            pitch : value_float ;
        }
    }
}
```

**Value**

A floating-point number representing the specified distance.

**Example**

```plaintext
pitch : 8.400e-01 ;
```

process_scale_factor Simple Attribute

This attribute specifies the factor to use before RC calculation to scale the length, width, and spacing.

**Note:**

Do not specify a value for the *process_scale_factor* attribute if you specify a value for the *shrinkage* attribute or *shrinkage_table* group.

**Syntax**

```plaintext
phys_library(library_nameid) {
    resource(architecture_enum) {
        routing_layer(layer_nameid) {
            process_scale_factor : value_float ;
        }
    }
}
```

**Value**

A floating-point number representing the scaling factor.

**Example**

```plaintext
process_scale_factor : 0.95 ;
```
**res_per_sq Simple Attribute**

The `res_per_sq` attribute specifies the resistance unit area of a routing layer.

**Syntax**

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      res_per_sq : value_float ;
      ...
    }
  }
}
```

A floating-point number representing the resistance value.

**Example**

```plaintext
res_per_sq : 1.200e-01 ;
```

**res_temperature_coefficient Simple Attribute**

Use the `temperatureCoeff` attribute to define the coefficient of the first-order correction to the resistance per square when the operating temperature is not equal to the nominal temperature at which the resistance per square variables are defined.

**Syntax**

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      res_temperature_coefficient : value_float ;
      ...
    }
  }
}
```

A floating-point number representing the temperature coefficient.

**Example**

```plaintext
res_temperature_coefficient : 0.00 ;
```

**routing_direction Simple Attribute**

The `routing_direction` attribute specifies the preferred direction for routing wires.
Syntax

```
phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            routing_direction : value_enum ;
            ...
        }
    }
}
```

**value**

Valid values are `horizontal` and `vertical`.

**Example**

```
routing_direction : horizontal ;
```

**same_net_min_spacing Simple Attribute**

This attribute specifies a smaller spacing distance rule than the default rule for two shapes belonging to the same net.

Syntax

```
phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            same_net_min_spacing : value_float ;
            ...
        }
    }
}
```

**value**

A floating-point number representing the spacing distance.

**Example**

```
same_net_min_spacing : 0.04 ;
```

**shrinkage Simple Attribute**

The `shrinkage` attribute specifies the total distance by which the wire width on the layer shrinks or expands. The shrinkage parameter is a sum of the shrinkage for each side of the wire. The postshrinkage wire width represents the final processed silicon width as calculated from the drawn silicon width in the design database.
Chapter 3: Specifying Groups in the resource Group
Syntax for Groups in the resource Group

Note:
Do not specify a value for the shrinkage attribute or shrinkage_table group if you specify a value for the process_scale_factor attribute.

Syntax

phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      shrinkage : value@float ;
      ...
    }
  }
}

value
A floating-point number representing the distance. A positive number represents shrinkage; a negative number represents expansion.

Example
shrinkage : 0.00046 ;

spacing Simple Attribute

The spacing attribute specifies the minimal (default) value for different net (edge to edge) spacing for regular wiring on the layer. This spacing value applies to all routing widths unless overridden by the ranged_spacing attribute in the same routing_layer group or by the wire_rule group.

Syntax

phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      spacing : value@float ;
      ...
    }
  }
}

value
A floating-point number representing the minimal different net spacing value.

Example
spacing : 3.200e-01 ;
**thickness Simple Attribute**

The *thickness* attribute specifies the nominal thickness of the routing layer.

**Syntax**

```plaintext
phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            thickness : value_float ;
            ...
        }
    }
}
```

**value**

A floating-point number representing the thickness.

**Example**

```plaintext
thickness : 0.02 ;
```

**u_shaped_wire_spacing Simple Attribute**

The *u_shaped_wire_spacing* attribute specifies that a u-shaped notch requires more spacing between wires than the value of the *spacing* attribute allows.

**Syntax**

```plaintext
phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            u_shaped_wire_spacing : value_float ;
            ...
        }
    }
}
```

**value**

A floating-point number that represents the spacing value.

**Example**

```plaintext
u_shaped_wire_spacing : 0.0 ;
```

**wire_extension Simple Attribute**

The *wire_extension* attribute specifies the distance for extending wires at vias.
Syntax

phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_layer(layer_nameid) {
      wire_extension : value_float ;
      ...
    }
  }
}

value

A floating-point number that represents the wire extension value. A zero value specifies no wire extension. A nonzero value must be at least half the routing width for the layer.

Example

wire_extension : 0.025 ;

wire_extension_range_check_connect_only
Simple Attribute

This attribute specifies whether the projection length requires wide wire spacing.

Syntax

phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_layer(layer_nameid) {
      wire_extension_range_check_connect_only : Boolean ;
      ...
    }
  }
}

value

Valid values are true and false.

Example

wire_extension_range_check_connect_only : true ;

wire_extension_range_check_corner
Simple Attribute

This attribute specifies whether the projection length requires wide wire spacing.
Syntax

\[
\text{phys_library}(\text{library_nameid}) \ { }
\text{resource}(\text{architecture enum}) \ { }
\text{routing_layer}(\text{layer_nameid}) \ { }
\begin{subarray}
\text{wire_extension_range_check_corner} : \text{Boolean} ;
\end{subarray}
\ldots
\]

*Boolean*

Valid values are true and false.

**Example**

\[
\text{wire_extension_range_check_corner} : \text{true} ;
\]

**conformal_lateral_oxide Complex Attribute**

This attribute specifies values for the thickness and permittivity of a layer.

Syntax

\[
\text{phys_library}(\text{library_nameid}) \ { }
\text{resource}(\text{architecture enum}) \ { }
\text{routing_layer}(\text{layer_nameid}) \ { }
\text{conformal_lateral_oxide}(\text{value}_1\text{float}, \text{value}_2\text{float}, \text{value}_3\text{float}, \text{value}_4\text{float}) ;
\ldots
\]

*value_* 1

A floating-point number that represents the oxide thickness.

*value_* 2

A floating-point number that represents the topwall thickness.

*value_* 3

A floating-point number that represents the sidewall thickness.

*value_* 4

A floating-point number that represents the oxide permittivity.

**Example**

\[
\text{conformal_lateral_oxide} (0.2, 0.3, 0.21, 3.6) ;
\]
**lateral_oxide Complex Attribute**

The `lateral_oxide` attribute specifies values for the thickness and permittivity of a layer.

**Syntax**

```
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_layer(layer_nameid) {
      lateral_oxide(thicknessfloat, permittivityfloat) ;
      ...
    }
  }
}
```

**thickess**

A floating-point number that represents the oxide thickness.

**permittivity**

A floating-point number that represents the oxide permittivity.

**Example**

```
lateral_oxide (0.4, 3.9) ;
```

**min_extension_width Complex Attribute**

The `min_extension_width` attribute specifies the rules for a protrusion.

**Syntax**

```
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_layer(layer_nameid) {
      min_extension_width (value_1float, value_2float, value_3float);
      ...
    }
  }
}
```

**value_1**

A floating-point number that represents minimum wire width.

**value_2**

A floating-point number that represents the maximum extension length.
value_3
A floating-point number that represents the minimum extension width.

Example
min_extension_width () ;

min_shape_edge Complex Attribute
For a polygon, this attribute specifies the maximum number of edges of minimum edge length.

Syntax
phys_library(library_nameid) {
resource(architectureenum) {
routing_layer(layer_nameid) {
    min_shape_edge (lengthfloat, edgesint );
    ...
})
}

length
A floating-point number that represents the minimum length of a polygon edge.

edges
An integer that represents the maximum number of polygon edges.

Example
min_shape_edge(0.02, 3) ;

plate_cap Complex Attribute
The plate_cap attribute specifies the interlayer capacitance per unit area when a wire on the first routing layer overlaps a wire on the second routing layer.

Note:
The plate_cap statement must follow all the routing_layer statements and precede the routing_wire_model statements.

Syntax
phys_library(library_nameid) {
resource(architectureenum) {
routing_layer(layer_nameid) {
plate_cap(PCAP_la_lbf, PCAP_la_lbf, PCAP_ln-1_lnf) ;
}
PCAP_{la_{lb}}

Represents a floating-point number that specifies the plate capacitance per unit area between two routing layers, layer a and layer b. The number of PCAP values is determined by the number of previously defined routing layers. You must specify every combination of routing layer pairs based on the order of the routing layers. For example, if the layers are defined as substrate, layer1, layer2, and layer3, then the PCAP values are defined in PCAP_{1_2}, PCAP_{1_3}, and PCAP_{2_3}.

Example

The example shows a plate_cap statement for a library with four layers. The values are indexed by the routing layer order.

plate_cap(0.35, 0.06, 0.0, 0.25, 0.02, 0.15) ;
/* PCAP_{1_2}, PCAP_{1_3}, PCAP_{1_4}, PCAP_{2_3}, PCAP_{2_4}, PCAP_{3_4} */

ranged_spacing Complex Attribute

The ranged_spacing attribute specifies the different net spacing (edge to edge) for regular wiring on the layer. You can also use the ranged_spacing attribute to specify the minimal spacing for a particular routing width range of the metal. You can use more than one ranged_spacing attribute to specify spacings for different ranges.

Syntax

phys_library{library_nameid} {
  resource{architectureenum} {
    routing_layer{layer_nameid} {
      ranged_spacing(min_width{float}, max_width{float},
        spacing{float});
        ...
    }
  }
}

min_width, max_width

Floating-point numbers that represent the minimum and maximum routing width range.

spacing

A floating-point number that represents the spacing.
Example
ranged_spacing(2.5, 5.5, 1.3) ;

**spacing_check_style Complex Attribute**
The `spacing_check` attribute specifies the minimum distance.

**Syntax**
```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer(layer_name_id) {
      spacing_check_style : check_style_name_enum ;
      ...
    }
  }
}
```

*check_style_name*
Valid values are `manhattan` and `diagonal`.

Example
```
spacing_check_style : diagonal ;
```

**stub_spacing Complex Attribute**
The `stub_spacing` attribute specifies the distances required between the edges of two objects on a layer when the distance that the objects run parallel to each other is less than or equal to a specified threshold.

**Syntax**
```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    stub_spacing(layer_name_id) {
      stub_spacing (spacing_float, max_length_threshold_float, min_wire_width_float, max_wire_width_float);
      ...
    }
  }
}
```

*spacing*
A floating-point number that is less than the minimum spacing value specified for the layer.
max_length_threshold
A floating-point number that represents the maximum distance that two objects on the layer can run parallel to each other.

min_wire_width
A floating-point number that represents the minimum spacing to a neighbor wire (optional).

max_wire_width
A floating-point number that represents the maximum spacing to a neighbor wire (optional).

Example
stub_spacing(1.05, 0.08)

end_of_line_spacing_rule Group
Use the end_of_line_spacing_rule attribute to specify the spacing between a stub wire and other wires.

Syntax
phys_library(library_name_id) {
resource(architecture_enum) {
routing_layer(layer_name_id) {
    end_of_line_spacing_rule() {
        ...
    }
}
}

Simple Attributes
end_of_line_corner_keepout_width
end_of_line_edge_checking
end_of_line_metal_max_width
end_of_line_min_spacing
max_wire_width

Example
end_of_line_spacing_rule () {
    ...
}

end_of_line_corner_keepout_width Simple Attribute
This attribute specifies the corner keepout width.
Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_layer(layer_nameid) {
      end_of_line_spacing_rule() {
        end_of_line_corner_keepout_width : value_boolean ;
        ...
      }
    }
  }
}

value
  Valid values are 1 and 0.

Example
end_of_line_corner_keepout_width : 0.0 ;

end_of_line_edge_checking Simple Attribute
This attribute specifies the number of edges to check.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_layer(layer_nameid) {
      end_of_line_spacing_rule() {
        end_of_line_edge_checking : value_enum ;
        ...
      }
    }
  }
}

value
  Valid values are one_edge, two_edges, and three_edges.

Example
end_of_line_edge_checking

end_of_line_metal_max_width Simple Attribute
The maximum distance between two objects on a layer.

Syntax
phys_library(library_nameid) {
resource(architecture$enum$) {
  routing_layer(layer_name$id$) {
    end_of_line_spacing_rule() {
      end_of_line_metal_max_width : value$float$ ;
      ... ...
    }
  }
  } 
}

value
  A floating-point number representing the width.

Example
end_of_line_metal_max_width

end_of_line_min_spacing Simple Attribute
This attribute specifies the minimum distance required between the parallel edges of two objects on the layer.

Syntax
phys_library(library_name$id$) { 
  resource(architecture$enum$) { 
    routing_layer(layer_name$id$) { 
      end_of_line_spacing_rule() { 
        end_of_line_min_spacing : value$float$ ;
        ... ...
      }
    }
  }
}

value
  A floating-point number representing the spacing.

Example
end_of_line_min_spacing : 0.0 ;

max_wire_width Simple Attribute
Use this attribute to specify the maximum wire width for the spacing rule.

Syntax
phys_library(library_name$id$) { 
  resource(architecture$enum$) {

routing_layer(layer_name_id) {
    end_of_line_spacing_rule() {
        max_wire_width : value float;
        ...
    }
}
}

value

A floating-point number representing the width.

**Example**

code

max_wire_width

text

**extension_via_rule Group**

Use this group to define specific via and minimum cut numbers for a given fat metal width and extension range.

**Syntax**

code

phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            extension_via_rule() {
                ...
            }
        }
    }
}

text

**Simple Attribute**

code

related_layer

text

**Groups**

code

min_cuts_table
reference_cut_table

text

**Example**

code

extension_via_rule() {
    ...
}

text

**related_layer**

The related_layer attribute specifies the contact layer to which this rule applies.
**Syntax**

phys_library(library_nameid) {
resource(architectureenum) {
routing_layer(layer_nameid) {
extension_via_rule() {
    related_layer : layer_nameid ;
    ...
}
}
}
}

layer_name

A string value representing the layer name.

**Example**

related_layer : ;

**min_cuts_table Group**

Use this group to specify the minimum number of vias.

**Syntax**

phys_library(library_nameid) {
resource(architectureenum) {
routing_layer(layer_nameid) {
    extension_via_rule() {
        min_cuts_table(template_nameid) {
    index_1(“value_float, value_float, ...”) ;
    index_2(“value_float, value_float, ...”) ;
        values (”value_float, value_float, ...”) ;
    }
    }
}
}

wire_lut_template_name

The wire_lut_template name.

**Complex Attributes**

index_1
index_2
values
index_1 and index_2 Complex Attributes

These attributes specify the default indexes.

Syntax

```plaintext
phys_library(library_nameid) {
resource(architectureenum) {
routing_layer(layer_nameid) {
    extension_via_rule() { 
        min_cuts_table(wire_lut_template_nameid) {
            index_1 ("valuefloat, valuefloat, ...");
            index_2 ("valuefloat, valuefloat, ...");
            values ("valuefloat, valuefloat, ...");
        }
    }
}
}
}
```

Example

```plaintext
extension_via_rule (template_name) {
    index_1 ("0.6, 0.8, 1.2");
    index_2 ("0.6, 0.8, 1.0");
    values ("0.07, 0.08, 0.09");
}
```

reference_cut_table Group

Use this group to specify a table of predefined via values.

Syntax

```plaintext
phys_library(library_nameid) {
resource(architectureenum) {
routing_layer(layer_nameid) {
extension_via_rule(via_array_lut_template_nameid) {
    reference_cut_table (wire_lut_template_nameid) {
        index_1("valuefloat, valuefloat, ...");
        index_2("valuefloat, valuefloat, ...");
        values ("valuefloat, valuefloat, ...");
    }
}
}
}
```

via_array_lut_template_name

The via_array_lut_template name.
Complex Attributes

index_1
index_2
values

index_1 and index_2 Complex Attributes
These attributes specify the default indexes.

Syntax
phys_library(library_nameId) {
  resource(architectureEnum) {
    routing_layer(layer_nameId) {
      extension_via_rule() {
        index_1 ("valuefloat, valuefloat, valuefloat, ...") ;
        index_2 ("valuefloat, valuefloat, valuefloat, ...") ;
        values ("valuefloat, valuefloat, valuefloat, ...") ;
      }
    }
  }
}

Example
extension_via_rule (template_name) {
  index_1 ("0.6, 0.8, 1.2") ;
  index_2 ("0.6, 0.8, 1.0") ;
  values ("0.07, 0.08, 0.09") ;
}

max_current_ac_absavg Group
Use this group to specify the absolute average value for the AC current that can pass through a cut.

Syntax
phys_library(library_nameId) {
  resource(architectureEnum) {
    routing_layer() {
      ...
    max_current_ac_absavg(template_nameId) {
      ...
    }
  }
}

template_name
The name of the contact layer.
Example
max_current_ac_absavg() {
  ...
}

Complex Attributes
index_1
index_2
index_3
values

max_current_ac_avg Group
Use this group to specify an average value for the AC current that can pass through a cut.

Syntax
phys_library(library_name_id) {
  resource(architecture_enum) {
    routing_layer () {
      ...
      max_current_ac_avg(template_name_id) {
        ...
      }
    }
  }
}

template_name
  The name of the contact layer.

Example
max_current_ac_avg() {
  ...
}

Complex Attributes
index_1
index_2
index_3
values

max_current_ac_peak Group
Use this group to specify a peak value for the AC current that can pass through a cut.

Syntax
phys_library(library_name_id) {

resource(architecture enum) {
    routing_layer () {
        ...
    }
    max_current_ac_peak(template_name id) {
        ...
    }
    }
} 

*template_name*

The name of the contact layer.

**Example**

```java
max_current_ac_peak() {
    ...
}
```

**Complex Attributes**

- `index_1`
- `index_2`
- `index_3`
- `values`

### max_current_ac_rms Group

Use this group to specify a root mean square value for the AC current that can pass through a cut.

**Syntax**

```java
phys_library(library_name id) {
    resource(architecture enum) {
        routing_layer () {
            ...
        }
        max_current_ac_rms(template_name id) {
            ...
        }
    }
}
```

*template_name*

The name of the contact layer.

**Example**

```java
max_current_ac_rms() {
    ...
```
Complex Attributes

index_1
index_2
index_3
values

max_current_dc_avg Group

Use this group to specify an average value for the DC current that can pass through a cut.

Syntax

phys_library(library_nameid) {
  resource(architecture enum) {
    routing_layer() {
      ...
      max_current_dc_avg(template_nameid) {
        ...
      }
    }
  }
}

template_name

The name of the contact layer.

Example

max_current_dc_avg() {
  ...
}

Complex Attributes

index_1
index_2
values

min_edge_rule Group

Use the min_edge_rule group to specify the minimum edge length rules.

Syntax

phys_library(library_nameid) {
  resource(architecture enum) {
    routing_layer(layer_nameid) {
      min_edge_rule() {
        ...
      }
    }
  }
}
Example
min_edge_rule () {
...
}

Simple Attributes
concave_corner_required
max_number_of_min_edges
max_total_edge_length
min_edge_length

concave_corner_required Simple Attribute
This attribute specifies whether a concave corner triggers a violation of the minimum edge length rules.

Syntax
phys_library(library_name\_id) { resource(architecture\_enum) { routing_layer(layer_name\_id) { min_edge_rule() { concave_corner_required : valueBoolean ; ... } } } }

value
Valid values are TRUE and FALSE.

Example
concave_corner_required : TRUE ;

max_number_of_min_edges Simple Attribute
This attribute specifies the maximum number of consecutive short (minimum) edges.

Syntax
phys_library(library_name\_id) { resource(architecture\_enum) { routing_layer(layer_name\_id) { } } }
min_edge_rule() {
    max_number_of_min_edges : value_int;
    ...
}
}
}
)value

An integer value representing the number of edges.

Example
max_number_of_min_edges : 1;

max_total_edge_length Simple Attribute
This attribute specifies the maximum allowable total edge length.

Syntax
phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            min_edge_rule() {
                max_total_edge_length : value_float;
                ...
            }
        }
    }
}
)value

A floating-point number representing the edge length.

Example
max_total_edge_length : 0.0;

min_edge_length Simple Attribute
The min_edge_length attribute specifies the length for defining short edges.

Syntax
phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_layer(layer_name_id) {
            min_edge_rule() {
                min_edge_length : value_float;
                ...
            }
        }
    }
}
term

A floating-point number representing the edge length.

Example

\[
\text{min\_edge\_length : 0.0 ;}
\]

**min\_enclosed\_area\_table Group**

Use this group to specify a range of values for an enclosed area.

**Syntax**

\[
\text{phys\_library(\text{library\_name}id) \{}
\text{resource(architecture\_enum) \{}
\text{routing\_layer(\text{layer\_name}id) \{}
\text{min\_enclosed\_area\_table(\text{wire\_lut\_template\_name}id) \{}
\ldots
\}
\}
\}
\]

**wire\_lut\_template\_name**

The **wire\_lut\_template** name.

Example

\[
\text{min\_enclosed\_area\_table ( ) ( ) ( } \ldots
\]

**Complex Attributes**

**index\_1**

values

**Index\_1 Complex Attribute**

The **index\_1** attribute specifies the default indexes.

**Syntax**

\[
\text{phys\_library(\text{library\_name}id) \{}
\text{resource(architecture\_enum) \{}
\text{routing\_layer(\text{layer\_name}id) \{}
\ldots
\}
\}
\]


min_enclosed_area_table(wire_lut_template_nameid) {
  index_1 ("valuefloat, valuefloat, valuefloat, ...")
  index_2 ("valuefloat, valuefloat, valuefloat, ...")
  values ("valuefloat, valuefloat, valuefloat, ...")
}

Example
min_enclosed_area_table (template_name) {
  index_1 ("0.6, 0.8, 1.2")
  values ("0.07, 0.08, 0.09")

notch_rule Group
Use the notch_rule group to specify the notch rules.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_layer(layer_nameid) {
      notch_rule() {
        ...
      }
    }
  }
}

Example
notch_rule () {
  ...
}

Simple Attributes
min_notch_edge_length
min_notch_width

min_notch_edge_length Simple Attribute
This attribute specifies the notch height.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_layer(layer_nameid) {
      notch_rule() {

min_notch_edge_length : valuefloat ;

...
value

A floating-point number representing the wire width.

Example
min_wire_width : 0.26 ;

resistance_table Group

Use this group to specify an array of values for sheet resistance.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
routing_layer(layer_nameid) {
resistance_table(template_nameid) {
  ...
  }
  }
}
}

template_name

The name of a resistance_lut_template defined at the phys_library level.

Example
resistance_table ( ) {
  ...
}

Complex Attributes

index_1
index_2
values

index_1 and index_2 Complex Attributes

These attributes specify the default indexes.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
routing_layer(layer_nameid) {

resistance_table(template_nameid) {
    index_1 ("valuefloat, valuefloat, valuefloat, ...")
    index_2 ("valuefloat, valuefloat, valuefloat, ...")
    values("valuefloat, valuefloat, valuefloat, ...")
}

Example
resistance_table (template_name) {
    index_1 ("0.6, 0.8, 1.2")
    index_2 ("0.6, 0.8, 1.0")
    values ("0.07, 0.08, 0.09")
}

shrinkage_table Group
Use this group to specify a lookup table template.

Syntax
phys_library(library_nameid) {
    resource(architectureenum) {
        routing_layer(layer_nameid) {
            shrinkage_table(template_nameid) {
                ...
            }
        }
    }
}

template_name
The name of a shrinkage_lut_template defined at the phys_library level.

Example
shrinkage_table (shrinkage_lut) {
    ...
}

Complex Attributes
index_1
index_2
values

index_1 and index_2 Complex Attributes
These attributes specify the default indexes.
Syntax

phys_library(library_nameid) {
    ...
    shrinkage_table (template_nameid) {
        index_1 (value_float, value_float, value_float, ...);
        index_2 (value_float, value_float, value_float, ...);
        values ("value_float, value_float, value_float", "...", "...");
        ...
    }
    ...
}

value, value, value, ...

Floating-point numbers that represent the indexes for this shrinkage table and the shrinkage table values.

Example

shrinkage_table (shrinkage_template_name) {
    values ("0.02, 0.03, 0.04", "0.0,1 0.02, 0.03" );
}

spacing_table Group

Use this group to specify a lookup table template.

Syntax

phys_library(library_nameid) {
    resource(architecture_enum) {
        routing_layer(layer_nameid) {
            spacing_table(template_nameid) {
                ...
            }
        }
    }
}

template_name

The name of a spacing_lut_template defined at the phys_library level.

Example

spacing_table (spacing_template_1) {
    ...
}

Complex Attributes

index_1
index_2
index_3
values

index_1, index_2, index_3, and values Complex Attributes

These attributes specify the indexes and values for the spacing table.

Syntax

```
phys_library(library_nameid) {
  ...
  spacing_table (template_nameid) {
    index_1 (valuefloat, valuefloat, valuefloat, ...);
    index_2 (valuefloat, valuefloat, valuefloat, ...);
    index_3 (valuefloat, valuefloat, valuefloat, ...);
    values ("valuefloat", "valuefloat", "...");
  }
  ...
}
```

`value, value, value, ...

Floating-point numbers that represent the indexes and spacing table values.

Example

```
spacing_table (spacing_template_1) {
  index_1 (0.0, 0.0, 0.0, 0.0);
  index_2 (0.0, 0.0, 0.0, 0.0);
  index_3 (0.0, 0.0, 0.0, 0.0);
  values (0.0, 0.0, 0.0, 0.0);
}
```

**wire_extension_range_table Group**

Use this group to specify the length of a wire extension where the wide wire spacing must be observed. A wire extension is a piece of thin or fat metal extended out from a wide wire.

Syntax

```
phys_library(library_nameid) {
  resource(architectureenum) {
    routing_layer(layer_nameid) {
      wire_extension_range_table(template_nameid) {
        ...
      }
    }
  }
}
```
**template_name**

The name of a wire_lut_template defined at the phys_library level.

Example

```plaintext
wire_extension_range_table (wire_template_1) {
    ...
}
```

Complex Attributes

**index_1**

values

**index_1 and values** Complex Attributes

These attributes specify the wire width values and corresponding wire_extension_range values.

Syntax

```plaintext
phys_library(library_name_id) {
    ...
    wire_extension_range_table (template_name_id) {
        index_1 (value_float, value_float, value_float, ...);
        values ("value_float, value_float, value_float", "...", "...");
    }
    ...
}
```

**value, value, value, ...**

Floating-point numbers.

Example

```plaintext
wire_extension_range_table (wire_template_1) {
    index_1 (0.4, 0.6, 0.8, 1.0);
    values ( "0.1, 0.2, 0.3, 0.4" ) ;
}
```

**routing_wire_model Group**

A predefined routing wire ratio model that represents an estimation on interconnect topology.

Syntax

```plaintext
phys_library(library_name_id) {
    resource(architecture_enum) {
        routing_wire_model(model_name_id) {
```
model_name

Specifies the name of the predefined routing wire model.

Example
routing_wire_model(mod1) {
  ...
}

Simple Attributes
wire_length_x
wire_length_y

Complex Attributes
adjacent_wire_ratio
overlap_wire_ratio
wire_ratio_x
wire_ratio_y

wire_length_x Simple Attribute

The wire_length_x attribute specifies the estimated average horizontal wire length in the direction for a net.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_wire_model(model_nameid) {
      ...
      wire_length_x : value float ;
      ...
    }
  }
}

value

A floating-point number that represents the average horizontal length.

Example
wire_length_x : 305.4 ;
wire_length_y Simple Attribute

The `wire_length_y` attribute specifies the estimated average vertical wire lengths in the direction for a net.

**Syntax**

```
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_wire_model(model_nameid) {
      ...
      wire_length_y : value_float ;
      ...
    }
  }
}
```

**value**

A floating-point number that represents the average vertical length.

**Example**

```
wire_length_y : 260.35 ;
```

adjacent_wire_ratio Complex Attribute

This attribute specifies the percentage of wiring on a layer that can run adjacent to wiring on the same layer and still maintain the minimum spacing.

**Syntax**

```
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_wire_model(model_nameid) {
      ...
      adjacent_wire_ratio(value_float, value_float, ...) ;
      ...
    }
  }
}
```

**value**

Floating-point numbers that represent the percentage value. For example, two parallel adjacent wires with the same length would have an `adjacent_wire_ratio` value of 50.0 percent. For a library with `n` routing layers, the `adjacent_wire_ratio` attribute has `n` floating values representing the ratio on each routing layer.
Example

In the case of a library with four routing layers:

\[
\text{adjacent\_wire\_ratio}(35.6, 2.41, 19.8, 25.3) ;
\]

\section*{overlap\_wire\_ratio Complex Attribute}

This attribute specifies the percentage of the wiring on the first layer that overlaps the second layer.

The following syntax example shows the order for the 20 entries required for a library with five routing layers.

\section*{Syntax}

\begin{verbatim}
phys_library(library_nameid) {
    resource(architectureenum) {
        routing_wire_model(model_nameid) {
            overlap_wire_ratio(
                V_1_2float, V_1_3float, V_1_4float, V_1_5float,
                V_2_1float, V_2_3float, V_2_4float, V_2_5float,
                V_3_1float, V_3_2float, V_3_4float, V_3_5float,
                V_4_1float, V_4_2float, V_4_3float, V_4_5float,
                V_5_1float, V_5_2float, V_5_3float, V_5_4float) ;
            ...
        }
    }
}
\end{verbatim}

\[V_{a\_b}\]

The overlap ratio that represents how much of the reference layer (a) is overshadowed by another layer (b). The value of each \(V_{a\_b}\) is a floating-point number from 0 to 100.0. The sum of all \(V_{a\_n}\) ratios must be less than or equal to 100.0. The order of \(V_{a\_b}\) is significant; it must be iteratively listed from the routing layer closest to the substrate.

Example

In the case of a library with five routing layers:

\begin{verbatim}
overlap_wire_ratio( 5, 15.5, 7.5, 10, \ 6.5, 16, 8.5, 10.5, \ 15, 5.5, 5, 15.5, \ 7.5, 10, 6.5, 16, \ 8.5, 10.5, 15, 5.5) ;
\end{verbatim}
wire_ratio_x Complex Attribute

The wire_ratio_x attribute specifies the percentage of total wiring in the horizontal direction that you estimate to be on each layer.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_wire_model(model_nameid) {
      ...
      wire_ratio_x(value_1float, value_2float, value_3float, ...) ;
      ...
    }
  }
}

value_1, value_2, value_3, ...

An array of floating-point numbers following the order of the routing layers, starting from the one closest to the substrate. Each example is a floating-point number value from 0 to 100.0. For example, if there are four routing layers, then there are four floating-point numbers.

Note:
The sum of the floating-point numbers must be 100.0.

Example
wire_ratio_x(25.0, 25.0, 25.0, 25.0) ;

wire_ratio_y Complex Attribute

The wire_ratio_y attribute specifies the percentage of total wiring in the vertical direction that you estimate to be on each layer.

Syntax
phys_library(library_nameid) {
  resource(architecture_enum) {
    routing_wire_model(model_nameid) {
      ...
      wire_ratio_y(value_1float, value_2float, value_3float, ...) ;
      ...
    }
  }
}
value_1, value_2, value_3, ...

An array of floating-point numbers following the order of the routing layers, starting from the one closest to the substrate. Each example is a floating-point number value from 0 to 100.0. For example, if there are four routing layers, then there are four floating-point numbers.

Note:
The sum of the floating-point numbers must be 100.0.

Example
wire_ratio_y(25.0, 25.0, 25.0, 25.0) ;

---

**site Group**

Defines the placement grid for macros.

Note:
Define a *site* group or a *tile* group, but not both.

**Syntax**

```plaintext
phys_library(library_nameid) {
  resource(architecture_enum) {
    site(site_nameid) {
      ...
    }
  }
}
```

**site_name**
The name of the site.

**Example**

```plaintext
site(core) {
  ...
}
```

**Simple Attributes**

- `on_tile`
- `site_class`
- `symmetry`

**Complex Attribute**

`size`
on_tile Simple Attribute

The on_tile attribute specifies an associated tile name.

Syntax

```plaintext
phys_library(library_nameid) {
    resource(architecture_enum) {
        site(site_nameid) {
            on_tile : tile_nameid 
            ... 
        }
    }
}
```

tile_name

The name of the tile.

Example

```plaintext
on_tile : ;
```

site_class Simple Attribute

The site_class attribute specifies what type of devices can be placed on the site.

Syntax

```plaintext
phys_library(library_nameid) {
    resource(architecture_enum) {
        site(site_nameid) {
            site_class : value_enum ; 
            ... 
        }
    }
}
```

value

Valid values are pad and core (default).

Example

```plaintext
site_class : pad ;
```

core

symmetry Simple Attribute

The symmetry attribute specifies the site symmetry. A site is considered asymmetrical, unless explicitly specified otherwise.
Syntax
phys_library(library_name_id) {
    resource(architecture_enum) {
        site(site_name_id) {
            symmetry : value_enum ;
            ...
        }
    }
}

value
Valid values are \( r, x, y, xy, \) and \( rxy. \)

where
x
Specifies symmetry about the x-axis

y
Specifies symmetry about the y-axis

r
Specifies symmetry in 90 degree counterclockwise rotation

xy
Specifies symmetry about the x-axis and the y-axis

rxy
Specifies symmetry about the x-axis and the y-axis and in 90 degree counterclockwise rotation increments

Example
symmetry : r ;

size Complex Attribute
The size attribute specifies the site dimension in normal orientation.

Syntax
phys_library(library_name_id) {
    resource(architecture_enum) {
        site(site_name_id) {
            size(x_size_float, y_size_float) ;
            ...
        }
    }
}
x_size, y_size

Floating-point numbers that specify the bounding rectangle size. The bounding rectangle size must be a multiple of the placement grid.

Example

size(0.9, 7.2) ;

tile Group

Use this group to define the placement grid for macros.

Note:
Define a site group or a tile group, but not both.

Syntax

phys_library(library_nameid) {
  resource(architecture_enum) {
    tile(tile_nameid) {
      ...
    }
  }
}

tile_name

The name of the tile.

Simple Attribute

tile_class

Complex Attribute

size

tile_class Simple Attribute

The tile_class attribute specifies the tile class.

Syntax

phys_library(library_nameid) {
  resource(architecture_enum) {
    tile(site_nameid) {
      tile_class : value_enum ;
      ...
    }
  }
}
value

Valid values are pad and core (default).

Example
tile_class : pad ;

size Complex Attribute

The size attribute specifies the site dimension in normal orientation.

Syntax
phys_library(library_nameid) {
    resource(architecture_enum) {
        tile (site_nameid) {
            size(x_sizefloat, y_sizefloat) ;
            ...
        }
    }
}

x_size, y_size

Floating-point numbers that specify the bounding rectangle size. The bounding rectangle size must be a multiple of the placement grid.

Example
size(0.9, 7.2) ;

via Group

Use this group to specify a via. You can use the via group to specify vias with any number of layers.

Syntax
phys_library(library_nameid) {
    resource(architecture_enum) {
        via(via_nameid) {
            ...
        }
    }
}
**via_name**

The name of the via.

**Example**

```plaintext
via(via12) {
  ...
}
```

**Simple Attributes**

- capacitance
- inductance
- is_default
- is_fat_via
- resistance
- res_temperature_coefficient
- top_of_stack_only
- via_id

**Groups**

- foreign
- via_layer

**capacitance Simple Attribute**

The **capacitance** attribute specifies the capacitance per cut.

**Syntax**

```plaintext
phys_library(library_nameid) {
  resource(architectureenum) {
    via(via_nameid) {
      capacitance : valuefloat ;
      ...
    }
  }
}
```

**value**

A floating-point number that represents the capacitance value.

**Example**

```plaintext
capacitance : 0.2 ;
```

**inductance Simple Attribute**

The **inductance** attribute specifies the inductance per cut.
Syntax

```
phys_library(library_nameid) {
  resource(architecture_enum) {
    via(via_nameid) {
      inductance : value_float;
      ...
    }
  }
}
```

**value**

A floating-point number that represents the inductance value.

**Example**

```
inductance : 0.5 ;
```

**is_default Simple Attribute**

The `is_default` attribute specifies the via as the default for the given layers.

Syntax

```
phys_library(library_nameid) {
  resource(architecture_enum) {
    via(via_nameid) {
      is_default : value_Boolean ;
      ...
    }
  }
}
```

**value**

Valid values are `TRUE` and `FALSE` (default).

**Example**

```
is_default : TRUE ;
```

**is_fat_via Simple Attribute**

The `is_fat_via` attribute specifies that fat wire contacts are required when the wire width is equal to or greater than the threshold specified. Specifies that this via is used by wide wires.

Syntax

```
phys_library(library_nameid) {
  resource(architecture_enum) {
    ...
  }
}
```
via(via_name_id) {
    is_fat_via : valueBoolean ;
    ...
}
}

value

Valid values are TRUE and FALSE (default).

Example
is_fat_via : TRUE ;

resistance Simple Attribute

The resistance attribute specifies the aggregate resistance per contact rectangle.

Syntax
phys_library(library_name_id) {
    resource(architecture_enum) {
        via(via_name_id) {
            resistance : valuefloat ;
            ...
        }
    }
}

value

A floating-point number that represents the resistance value.

Example
resistance : 0.0375 ;

res_temperature_coefficient Simple Attribute

This attribute specifies the coefficient of the first-order correction to the resistance per square when the operating temperature does not equal the nominal temperature.

Syntax
phys_library(library_name_id) {
    resource(architecture_enum) {
        via(via_name_id) {
            res_temperature_coefficient : valuefloat ;
            ...
        }
    }
}
value

A floating-point number that represents the coefficient.

Example
res_temperature_coefficient : 0.03 ;

**top_of_stack_only Simple Attribute**

This attribute specifies to use the via only on top of a via stack.

**Syntax**

```
phys_library(library_nameid) {
    resource(architecture_name) {
        via(via_nameid) {
            top_of_stack_only : valueBoolean ;
            ...
        }
    }
}
```

**value**

Valid values are **TRUE** and **FALSE** (default).

Example
top_of_stack_only : FALSE ;

**via_id Simple Attribute**

Use the *via_id* attribute to specify a number that identifies a device.

**Syntax**

```
phys_library(library_nameid) {
    resource(architecture_name) {
        via(via_nameid) {
            via_id : value_int ;
            ...
        }
    }
}
```

**value**

Valid values are any integer between 1 and 255.
Example

\texttt{via\_id : 255 ;}

**foreign Group**

Use this group to specify which GDSII structure (model) to use when placing an instance of this via.

\textbf{Note:} Only one foreign reference is allowed for each via.

**Syntax**

\begin{verbatim}
phys_library(library_nameid) {
 resource(architectureenum) {
   via(via_nameid) {
     foreign(foreign_object_nameid) {
       ...
     }
   }
 }
}
\end{verbatim}

*foreign_object_name*

The name of the corresponding GDSII via (model).

**Example**

\begin{verbatim}
foreign(via34) {
  ...
}
\end{verbatim}

**Simple Attribute**

*orientation*

**Complex Attribute**

*origin*

**orientation Simple Attribute**

The *orientation* attribute specifies how you place the foreign GDSII object.

**Syntax**

\begin{verbatim}
phys_library(library_nameid) {
 resource(architectureenum) {
   via(via_nameid) {
     foreign(foreign_object_nameid) {
       orientation : valueenum ;
     }
   }
 }
}
\end{verbatim}
value

Valid values are $N$ (north), $E$ (east), $S$ (south), $W$ (west), $FN$ (flip north), $FE$ (flip east), $FS$ (flip south), and $FW$ (flip west), as shown in Figure 3-3.

**Figure 3-3 Orientation Examples**

![Orientation Examples](image-url)

**Example**

`orientation : FN ;`

**origin Complex Attribute**

The `origin` attribute specifies the via origin with respect to the GDSII structure (model). In the physical library, the origin of a via is its center; in GDSII, the origin is 0,0.

**Syntax**

```plaintext
phys_library(library_name_id) {
  resource(architecture_enum) {
    via(via_name_id) {
      foreign(foreign_object_name_id) {
        ...
        origin(num_x_float, num_y_float) ;
      }
    }
  }
}
```
num_x, num_y

Numbers that specify the x- and y-coordinates.

Example

\texttt{origin(-1, -1) ;}

\textbf{via\textunderscore layer Group}

Use this group to specify layer geometries on one layer of the via.

\textbf{Syntax}

\footnotesize

\begin{verbatim}
phys_library\{library\_nameid\} {
resource\{architecture\_enum\} {
    via\{via\_nameid\} {
    via\_layer\{layer\_nameid\} {
        ...
    }
    }
}
}
\end{verbatim}

\end{footnotesize}

\textit{layer\_name}

Specifies the layer on which the geometries are located.

Example

\footnotesize

\begin{verbatim}
via\_layer\{m1\} {
    ...
}
\end{verbatim}

\end{footnotesize}

\textbf{Simple Attributes}

\begin{itemize}
    \item max\_wire\_width
    \item min\_wire\_width
\end{itemize}

\textbf{Complex Attributes}

\begin{itemize}
    \item contact\_spacing
    \item contact\_array\_spacing
    \item enclosure
    \item max\_cuts
    \item min\_cuts
    \item rectangle
    \item rectangle\_iterate
\end{itemize}

\textbf{max\_wire\_width Simple Attribute}

Use this attribute along with the \texttt{min\_wire\_width} attribute to define the range of wire widths.
Syntax

phys_library(library_nameid) {
  resource(architectureenum) {
    via(via_nameid) {
      via_layer(layer_nameid) {
        max_wire_width : value float ;
        ...
      }
    }
  }
}

value

A floating-point number representing the wire width.

Example

max_wire_width : 0.0 ;

min_wire_width Simple Attribute

Use this attribute along with the max_wire_width attribute to define the range of wire widths.

Syntax

phys_library(library_nameid) {
  resource(architectureenum) {
    via(via_nameid) {
      via_layer(layer_nameid) {
        min_wire_width : value float ;
        ...
      }
    }
  }
}

value

A floating-point number representing the wire width.

Example

min_wire_width : 0.0 ;

contact_array_spacing Complex Attribute

This attribute specifies the edge-to-edge spacing on a contact layer.
Syntax

phys_library(library_nameid) {
resource(architecture_enum) {
    via(via_nameid) {
via_layer(layer_nameid) {
    contact_array_spacing(value_xfloat, value_yfloat);
    
    ...  
    
}  
}
}
}

value_x, value_y

Floating-point numbers that represent the horizontal and vertical spacing between two abutting contact arrays.

Example

contact_array_spacing (0.0, 0.0) ;

contact_spacing Complex Attribute

The contact_spacing attribute specifies the center-to-center spacing for generating an array of contact cuts in the via.

Syntax

phys_library(library_nameid) {
resource(architecture_enum) {
    via(via_nameid) {
via_layer(layer_nameid) {
    contact_spacing(value_xfloat, value_yfloat);
    
    ...  
    
}  
}
}
}

x, y

Floating-point numbers that represent the spacing value in terms of the x distance and y distance between the centers of two contact cuts.

Example

contact_spacing (0.0, 0.0) ;

closure Complex Attribute

The enclosure attribute specifies an enclosure on a metal layer.
Syntax
phys_library(library_nameid) {
    resource(architectureenum) {
        via(via_nameid) {
            via_layer(layer_nameid) {
                enclosure(value_xfloat, value_Yfloat ) ;
                ...
            }
        }
    }
}

value_x, value_y
Floating-point numbers that represent the enclosure.

Example
enclosure (0.0, 0.0) ;

max_cuts Complex Attribute
The max_cuts attribute specifies the maximum number of cuts on a contact layer.

Syntax
phys_library(library_nameid) {
    resource(architectureenum) {
        via(via_nameid) {
            via_layer(layer_nameid) {
                max_cuts(value_xfloat, value_Yfloat ) ;
                ...
            }
        }
    }
}

value_x, value_y
Floating-point numbers that represent the maximum number of cuts in the horizontal and vertical directions of a contact array.

Example
max_cuts (0.0, 0.0) ;

min_cuts Complex Attribute
The min_cuts attribute specifies the minimum number of neighboring cuts allowed within a specified space (range).
Syntax

phys_library(library_nameid) {
resource(architectureenum) {
    via(via_nameid) {
        via_layer(layer_nameid) {
            min_cuts(value_xfloat, value_Yfloat) ;
            ...;
        }
    }
}
}

value_x, value_y

Floating-point numbers that represent the minimum number of cuts in the horizontal and vertical directions of a contact array.

Example

min_cuts (0.0, 0.0) ;

rectangle Complex Attribute

The rectangle attribute specifies a rectangular shape for the via.

Syntax

phys_library(library_nameid) {
resource(architectureenum) {
    via(via_nameid) {
        via_layer(layer_nameid) {
            rectangle(x1float, y1float, x2float, y2float) ;
            ...;
        }
    }
}
}

x1, y1, x2, y2

Floating-point numbers that specify the coordinates for the diagonally opposite corners of the rectangle.

Example

rectangle(-0.3. -0.3, 0.3, 0.3) ;

rectangle_iterate Complex Attribute

The rectangle_iterate attribute specifies an array of rectangles in a particular pattern.
Syntax

phys_library(library_nameid) {
resource(architecture enum) {
    via(via_nameid) {
    via_layer(layer_nameid) {
        rectangle_iterate(num_xint, num_Yint,
            space_xfloat, space_yfloat,
            x1float, y1float, x2float, y2float)
        ...
    }
}
}

\textit{num}_x, \textit{num}_y

Integer numbers that represent the number of columns and rows in the array, respectively.

\textit{space}_x, \textit{space}_y

Floating-point numbers that specify the value for spacing around the rectangles.

\textit{x1}, \textit{y1}; \textit{x2}, \textit{y2}

Floating-point numbers that specify the coordinates for the diagonally opposite corners of the rectangles.

Example

rectangle_iterate(2, 2, 2.000, 4.000, 175.500, 1417.360, 176.500, 1419.140) ;

\underline{via\textunderscore array\textunderscore rule Group}

Defines the specific via and minimum cut number for the different fat metal wire widths on contact layer.

Syntax

phys_library(library_nameid) {
resource(architecture enum) {
    via_array_rule () {
        
        ...
    }
}
}

Groups

min_cuts_table
reference_cut_table

min_cuts_table Group

Use this group to specify the values for the lookup table.

Note:
Only one foreign reference is allowed for each via.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
    via_array_rule() {
        min_cuts_table (template_nameid) {
            ...
        }
    }
}
}

template_name
The via_array_lut_template name.

Example
min_cuts_table (via34) {
    ...
}

Complex Attribute
index_1
index_2
values

index Complex Attribute
The index attribute specifies the default indexes.

Syntax
phys_library(library_nameid) {
resource(architecture_enum) {
    via_array_rule() {
        min_cuts_table (template_nameid) {
            ...
            index(num_xfloat, num_yfloat) ;
            ...
        }
    }
}
}
\textit{num}_x, \textit{num}_y

Numbers that specify the x- and y-coordinates.

\textbf{Example}
index (-1, -1) ;

\textbf{reference\_cut\_table Group}
Use this group to specify values for the lookup table.

\textbf{Syntax}
\begin{verbatim}
phys_library\{library\_nameid\} { 
  resource\{architecture\_enum\} { 
    via\_array\_rule() { 
      reference\_cut\_table\{template\_nameid\} { 
        ... 
      } 
    } 
  } 
}
\end{verbatim}

\textbf{template\_name}
The \texttt{via\_array\_lut\_template} name.

\textbf{Example}
reference\_cut\_table\{via34\} { 
  ... 
}

\textbf{Complex Attribute}
\begin{itemize}
  \item \textit{index\_1}
  \item \textit{index\_2}
  \item \textit{values}
\end{itemize}

\textbf{index\ Complex\ Attribute}
The \textit{index} attribute specifies the default indexes.

\textbf{Syntax}
\begin{verbatim}
phys_library\{library\_nameid\} { 
  resource\{architecture\_enum\} { 
    via\_array\_rule() { 
      reference\_cut\_table\{template\_nameid\} { 
        ... 
        index\{num\_x\_float, num\_y\_float\} ; 
      } 
    } 
  } 
}
\end{verbatim}


} }

num_x, num_y

Numbers that specify the x- and y-coordinates.

Example

index (-1, -1) ;
You use the `topological_design_rules` group to specify the design rules for the technology (such as minimum spacing and width).

The information in this chapter includes a description and syntax example for the attributes that you can define within the `topological_design_rules` group.
Syntax for Attributes in the topological_design_rules Group

This chapter describes the attributes that you define in the topological_design_rules group. The groups that you can define in the topological_design_rules group are described in Chapter 5.

topological_design_rules Group

Defines all the design rules that apply to the physical library.

Syntax

phys_library(library_nameid) {
    topological_design_rules() {
        ...
    }
}

Note:

A name is not required for the topological_design_rules group.

Example

topological_design_rules() {
    ...
}

Simple Attributes

antenna_inout_threshold
antenna_input_threshold
antenna_output_threshold
min_enclosed_area_table_surrounding_metal

Complex Attributes

contact_min_spacing
corner_min_spacing
diff_net_min_spacing
end_of_line_enclosure
min_enclosure
min_generated_via_size
min_overhang
same_net_min_spacing

Group

extension_wire_spacing_rule
antenna_inout_threshold Simple Attribute

Use this attribute to specify the default (maximum) threshold (cumulative) value for the antenna effect on inout pins. Use this attribute for parameter-based calculations only; that is, it is not required when your library contains an antenna_rule group.

Syntax

```
phys_library(library_name_id) {
    topological_design_rules() {
        antenna_inout_threshold : value float ;
        ...
    }
}
```

value

A floating-point number that represents the global pin value.

Example

```
antenna_inout_threshold : 0.0;
```

antenna_input_threshold Simple Attribute

Use this attribute to specify the default (maximum) threshold (cumulative) value for the antenna effect on input pins. Use this attribute for parameter-based calculations only; that is, it is not required when your library contains an antenna_rule group.

Syntax

```
phys_library(library_name_id) {
    topological_design_rules() {
        antenna_input_threshold : value float ;
        ...
    }
}
```

value

A floating-point number that represents the global pin value.

Example

```
antenna_input_threshold : 0.0;
```

antenna_output_threshold Simple Attribute

Use this attribute to specify the default (maximum) threshold (cumulative) value for the antenna effect on output pins. Use this attribute for parameter-based calculations only; that is, it is not required when your library contains an antenna_rule group.
Syntax

```
phys_library(library_nameid) {
   topological_design_rules() {
      antenna_output_threshold : valuefloat;
      ...
   }
}
```

_value_

A floating-point number that represents the global pin value.

**Example**

```
antenna_output_threshold : 0.0;
```

**min_enclosed_area_table_surrounding_metal**

Simple Attribute

Use this attribute to specify the minimum enclosed area.

Syntax

```
phys_library(library_nameid) {
   topological_design_rules() {
      min_enclosed_area_table_surrounding_metal(valueenum);
      ...
   }
}
```

_value_

Valid values are `all_fat_wires` and `at_least_one_fat_wire`.

**Example**

```
min_enclosed_area_table_surrounding_metal : all_fat_wires;
```

**contact_min_spacing** Complex Attribute

The `contact_min_spacing` attribute specifies the minimum spacing required between two different contact layers on different nets.

Syntax

```
phys_library(library_nameid) {
   topological_design_rules() {
      contact_min_spacing(layer1_nameid, layer2_nameid, valuefloat);
      ...
   }
}
```
layer1_name, layer2_name

Specify the two contact layers. The layers can be equivalent or different.

value

A floating-point number that represents the spacing value.

Example

contact_min_spacing(cut01, cut12, 1)

corner_min_spacing Complex Attribute

The corner_min_spacing attribute specifies the spacing between two different contact layers.

Note:
The corner_min_spacing simple attribute in the cont_layer group specifies the minimum distance between two vias.

Syntax

phys_library(library_nameid) {
  topological_design_rules() {
    corner_min_spacing(layer1_nameid, layer2_nameid, value_float) ;
    ...
  }
}

layer1_name, layer2_name

Specify the two contact layers.

value

A floating-point number that represents the spacing value.

Example

corner_min_spacing () ;

det_of_line_enclosure Complex Attribute

The end_of_line_enclosure attribute defines an enclosure size to specify the end-of-line rule for routing wire segments.

Syntax

phys_library(library_nameid) {
  topological_design_rules() {
    end_of_line_enclosure(layer1_nameid, layer2_nameid, value_float) ;
    ...
  }
}
layer1_name, layer2_name
Specify the metal layer and a contact layer, respectively.

value
A floating-point number that represents the spacing value.

Example
end_of_line_enclosure () ;

**min_enclosure Complex Attribute**

The `min_enclosure` attribute defines the minimum distance at which a layer must enclose another layer when the two layers overlap.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        min_enclosure(layer1_nameid, layer2_nameid, valuefloat) ;
        ...
    }
}
```

layer1_name, layer2_name
Specify the metal layer and a contact layer, respectively.

value
A floating-point number that represents the spacing value.

Example
min_enclosure () ;

**diff_net_min_spacing Complex Attribute**

The `diff_net_min_spacing` attribute specifies the minimum spacing between a metal layer and a contact layer.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        diff_net_min_spacing(layer1_nameid, layer2_nameid, valuefloat) ;
        ...
    }
}
```
layer1_name, layer2_name
Specify the metal layer and a contact layer, respectively.

value
A floating-point number that represents the spacing value.

Example
diff_net_min_spacing () ;

min_generated_via_size Complex Attribute
Use this attribute to specify the minimum size for the generated via. All edges of a via must lie on the grid defined by the x- and y-coordinates.

Syntax
phys_library(library_nameid) {
topological_design_rules() {
    min_generated_via_size(num_xfloat, num_yfloat) ;
...
}
}

num_x, num_y
Floating-point numbers that represent the minimum size for the x and y dimensions.

Example
min_generated_via_size(0.01, 0.01) ;

min_overhang Complex Attribute
Use this attribute to specify the minimum overhang for the generated via.

Syntax
phys_library(library_nameid) {
topological_design_rules() {
    min_overhang(layer1_string, layer2_string, valuefloat) ;
...
}
}

layer1, layer2
The names of the two overhanging layers.
value

A floating-point number that represents the minimum overhang value.

Example
min_overhang(0.01, 0.01);

same_net_min_spacing Complex Attribute

The same_net_min_spacing attribute specifies the minimum spacing required between wires on a layer or on two layers in the same net.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {
    same_net_min_spacing(layer1_nameid, layer2_nameid, \space float, is_stack Boolean) ;

    ...
  }
}

layer1_name, layer2_name
Specify the two routing layers, which can be different layers or the same layer.

space
A floating-point number representing the spacing value.

is_stack
Valid values are TRUE and FALSE. Set the value to TRUE to allow stacked vias at the routing layer. When set to TRUE, the same_net_min_spacing value can be 0 (complete overlap) or the value held by the min_spacing attribute; otherwise the value reflects the rule.

Example
same_net_min_spacing(m2, m2, 0.4, FALSE)
You use the \texttt{topological\_design\_rules} group to specify the design rules for the technology (such as minimum spacing and width).

This chapter describes the following groups:

- \texttt{antenna\_rule} Group
- \texttt{density\_rule} Group
- \texttt{extension\_wire\_spacing\_rule} Group
- \texttt{stack\_via\_max\_current} Group
- \texttt{via\_rule} Group
- \texttt{via\_rule\_generate} Group
- \texttt{wire\_rule} Group
- \texttt{wire\_slotting\_rule} Group
Syntax for Groups in the topological_design_rules Group

The following sections describe the groups you can define in the topological_design_rules group:

antenna_rule Group

Use this group to specify the methods for calculating the antenna effect.

Syntax

```
phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      ...
    }
  }
}
```

`antenna_rule_name`

The name of the antenna_rule group.

Example

```
antenna_rule (antenna_metal3_only) {
  ...description...
}
```

Simple Attributes

- adjusted_gate_area_calculation_method
- adjusted_metal_area_calculation_method
- antenna_accumulation_calculation_method
- antenna_ratio_calculation_method
- apply_to
- geometry_calculation_method
- pin_calculation_method
- routing_layer_calculation_method

Complex Attribute

- layer_antenna_factor

Groups

- adjusted_gate_area
- adjusted_metal_area
- antenna_ratio
- metal_area_scaling_factor
**adjusted_gate_area_calculation_method**

Simple Attribute

Use this attribute to specify a factor to apply to the gate area.

Syntax

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    antenna_rule(antenna_rule_name_id) {
      adjusted_gate_area_calculation_method : valueenum ;
      ...
    }
  }
}
```

**value**

Valid values are `max_diffusion_area` and `total_diffusion_area`.

Example

```plaintext
adjusted_gate_area_calculation_method : max_diffusion_area;
```

**adjusted_metal_area_calculation_method**

Simple Attribute

Use this attribute to specify a factor to apply to the metal area.

Syntax

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    antenna_rule(antenna_rule_name_id) {
      adjusted_metal_area_calculation_method : valueenum ;
      ...
    }
  }
}
```

**value**

Valid values are `max_diffusion_area` and `total_diffusion_area`.

Example

```plaintext
adjusted_metal_area_calculation_method :
max_diffusion_area ;
```
antenna_accumulation_calculation_method Simple Attribute

Use this attribute to specify a method for calculating the antenna.

Syntax

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    antenna_rule(antenna_rule_name_id) {
      antenna_accumulation_calculation_method: valueenum;
    }
  }
}
```

value

Valid values are single_layer, accumulative_ratio, and accumulative_area.

Example

```plaintext
antenna_accumulation_calculation_method : ;
```

antenna_ratio_calculation_method Simple Attribute

Use this attribute to specify a method for calculating the antenna.

Syntax

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    antenna_rule(antenna_rule_name_id) {
      antenna_ratio_calculation_method : valueenum;
    }
  }
}
```

value

Valid values are infinite_antenna_ratio, max_antenna_ratio, and total_antenna_ratio.

Example

```plaintext
antenna_ratio_calculation_method : total_antenna_ratio ;
```

apply_to Simple Attribute

The apply_to attribute specifies the type of pin geometry that the rule applies to.
Syntax

phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      apply_to : valueenum ;
      ...
    }
  }
}

value

The valid values are gate_area, gate_perimeter, and diffusion_area.

Example

apply_to : gate_area ;

geometry_calculation_method Simple Attribute

Use this attribute with the pin_calculation_method attribute to specify which geometries are applied to which pins. See Table 5-1 for a matrix of the options.

Syntax

phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      ...
      geometry_calculation_method : valueenum ;
      pin_calculation_method : valueenum ;
      ...
    }
  }
}

value

The valid values are all_geometries and connected_only.

Table 5-1 Calculating Geometries on Pins

<table>
<thead>
<tr>
<th>geometry_calculation_method values</th>
<th>pin_calculation_method values</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_pins</td>
<td>each_pin</td>
</tr>
</tbody>
</table>
Table 5-1  Calculating Geometries on Pins (continued)

<table>
<thead>
<tr>
<th>all_geometries</th>
<th>All the geometries are applied to all pins. The connectivity analysis is not performed. Pins share antennas.</th>
<th>All the geometries of the net are applied to every pin on the net separately. The connectivity analysis is not performed. Antennas are not shared by connected pins. This is the most pessimistic calculation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>connected_only</td>
<td>Considers connected geometries as well as sharing. This is the most accurate calculation.</td>
<td>Only the geometries connected to the pin are considered. Sharing of antennas is not allowed.</td>
</tr>
</tbody>
</table>

Example
gallery_calculation_method : connected_only ;
pin_calculation_method : all_pins ;

metal_area_scaling_factor_calculation_method
Simple Attribute
Use this attribute to specify which diffusion area to use for scaling the metal area.

Syntax
phys_library(library_nameid) {  
topological_design_rules() {  
  antenna_rule(antenna_rule_nameid) {  
    ...  
    metal_area_scaling_factor_calculation_method : valueenum ;  
    ...  
  }  
}  
}

value
The valid values are max_diffusion_area and total_diffusion_area.

Example
metal_area_scaling_factor_calculation_method : total_diffusion_area ;
**pin_calculation_method Simple Attribute**

Use this attribute with the geometry_calculation_method attribute to specify which geometries are applied to which pins. See Table 5-1 for a matrix of the options.

**Syntax**

```plaintext
phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      ...
      geometry_calculation_method : value_enum ;
      pin_calculation_method : value_enum ;
      ...
    }
  }
}
```

**value**

The valid values are all_pins and each_pin.

**Example**

```plaintext
geometry_calculation_method : connected_only ;
pin_calculation_method : all_pins ;
```

**routing_layer_calculation_method Simple Attribute**

Use this attribute to specify which property of the routing segments to use to calculate antenna contributions.

**Syntax**

```plaintext
phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      ...
      routing_layer_calculation_method : value_enum ;
      ...
    }
  }
}
```

**value**

The valid values are side_wall_area, top_area, side_wall_and_top_area, segment_length, and segment_perimeter.

**Example**

```plaintext
routing_layer_calculation_method : top_area ;
```
layer_antenna_factor Complex Attribute

The layer_antenna_factor attribute specifies a factor in each routing or contact layer that is multiplied to either the area or the length of the routing segments to determine their contribution.

Syntax

\[
\text{phys_library(library_nameid) \{ }
\quad \text{topological_design_rules() \{ }
\quad \quad (antenna_rule_nameid) \{
\quad \quad \quad \ldots
\quad \quad \quad \quad \text{layer_antenna_factor(layer_namestring, antenna_factorfloat) ;}
\quad \quad \quad \ldots
\quad \quad \}
\quad \}
\]

layer_name
  Specifies the layer that contains the factor.

antenna_factor
  A floating-point number that represents the factor.

Example

layer_antenna_factor (m1_m2, 1) ;

adjusted_gate_area Group

Use this group to specify gate area values.

Syntax

\[
\text{phys_library(library_nameid) \{ }
\quad \text{topological_design_rules() \{ }
\quad \quad \text{antenna_rule(antenna_rule_nameid) \{ }
\quad \quad \quad \ldots
\quad \quad \quad \quad \text{adjusted_gate_area(antenna_lut_template_nameid) \{ }
\quad \quad \quad \quad \quad \ldots
\quad \quad \quad \quad \}
\quad \quad \}
\quad \}
\]

template_name
  The name of the template.
Example

adjusted_gate_area () {
    ...description...
}

Complex Attributes

index_1
values

adjusted_metal_area Group

Use this group to specify metal area values.

Syntax

phys_library(library_nameid) {
    topological_design_rules() {
        antenna_rule(antenna_rule_nameid) {
            ...
            adjusted_metal_area(antenna_lut_template_nameid) {
                ...
                ...

            }
        }
    }
}

template_name

The name of the template.

Example

adjusted_metal_area () {
    ...description...
}

Complex Attributes

index_1
values

antenna_ratio Group

Use this group to specify the piecewise linear table for antenna calculations.

Syntax

phys_library(library_nameid) {
    topological_design_rules() {
        antenna_rule(antenna_rule_nameid) {
            ...
            antenna_ratio (template_nameid) {
                ...description...
            }
        }
    }
}

Example
antenna_ratio (antenna_template_1) {
  ...
}

**Complex Attributes**

**index_1 Complex Attribute**

Use this optional attribute to specify, in ascending order, each diffusion area limit.

**Syntax**

```plaintext
phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      ...
      antenna_ratio (template_nameid) {
        index_1(value_float, value_float, value_float, ...) ;
      }
      ...
    }
  }
}
```

**values Complex Attribute**

The *values* attribute specifies the table ratio.

**Syntax**

```plaintext
phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      ...
    }
}
```
antenna_ratio (template_nameid) {
    values (valuefloat, valuefloat, valuefloat, ...) ;
}
}

value, value, value, ...

Floating-point numbers that represent the ratio to apply.

Example
antenna_ratio (antenna_template_1) {
    values (10, 100, 1000) ;
}

Example 5-1 shows the attributes and group in an antenna rule group.

Example 5-1  An antenna_rule Group
antenna_rule (antenna_metal3_only) {
    apply_to : gate_area
    geometry_calculation_method : connected_only
    pin_calculation_method : all_pins ;
    routing_layer_calculation_method : side_wall_area ;
    layer_antenna_factor (m1_m2, 1) ;
    antenna_ratio (antenna_template_1) {
        values (10, 100, 1000) ;
    }
    metal_area_scaling_factor () {
        ...
    }
}

metal_area_scaling_factor Group
Use this group to specify the piecewise linear table for antenna calculations.

Syntax
phys_library(library_nameid) {
    topological_design_rules() {
        antenna_rule(antenna_rule_nameid) {
            ...
            metal_area_scaling_factor (template_nameid) {
                ...description...
            }
        }
    }
}
Example
antenna_ratio (antenna_template_1) {
  ...
}

Complex Attributes
index_1
values

index_1 Complex Attribute
Use this optional attribute to specify, in ascending order, each diffusion area limit.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      ...
      antenna_ratio (template_nameid) {
        index_1(value_float, value_float, value_float, ...) ;
      } ...
    }
  }
}

value, value, value, ...
Floating-point numbers that represent diffusion area limits in ascending order.

Example
antenna_ratio (antenna_template_1) {
  index_1 (*0, 2.4, 4.8") ;
}

values Complex Attribute
The values attribute specifies the table ratio.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {
    antenna_rule(antenna_rule_nameid) {
      ...
      antenna_ratio (template_nameid) {
        values (value_float, value_float, value_float, ...) ;
      }
    }
  }
}
value, value, value, ...

Floating-point numbers that represent the ratio to apply.

**Example**

```plaintext
antenna_ratio (antenna_template_1) {
    values (10, 100, 1000);
}
```

---

**default_via_generate Group**

Use the `default_via_generate` group to specify default horizontal and vertical layer information.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        default_via_generate (name) {
            via_routing_layer( layer_name ) {
                overhang (float, float); /*horizontal and vertical*/
                end_of_line_overhang : float;
            }
            via_contact_layer(layer_name) {
                rectangle (float, float, float, float);
                resistance : float;
            }
        }
    }
    ...
}
```

---

**density_rule Group**

Use this group to specify the metal density rule for the layer.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        density_rule(routing_layer_nameid) {
            ...
        }
    }
}
```

`routing_layer_name`
Example
density_rule () {
    ...
}

Complex Attributes
check_step
check_window_size
density_range

check_step Complex Attribute
The check_step attribute specifies the stepping distance in distance units.

Syntax
phys_library(library_nameid) {
    topological_design_rules() {
        density_rule(routing_layer_nameid) {
            check_step (value_1float, value_2float )
                ...
        }
    }
}

value_1, value_2
Floating-point numbers representing the stepping distance.

Example
check_step (0.0. 0.0);

cHECK_WINDOW_SIZE Complex Attribute
The check_window_size attribute specifies the check window dimensions.

Syntax
phys_library(library_nameid) {
    topological_design_rules() {
        density_rule(routing_layer_nameid) {
            check_window_size (x_valuefloat, y_valuefloat )
                ...
        }
    }
}

x_value, y_value
Floating-point numbers representing the window size.
Example
check_window_size (0.5, 0.5);

density_range Complex Attribute
The density_range attribute specifies density percentages.

Syntax
phys_library(library_name_id) {
    topological_design_rules() {
        density_rule(routing_layer_name_id) {
            density_range (min_valuefloat, max_valuefloat)
            ...
        }
    }
}

min_value, max_value
Floating-point numbers representing the minimum and maximum density percentages.

Example
density_range (0.0, 0.0);

extension_wire_spacing_rule Group
The extension_wire_spacing_rule group specifies the extension range for connected wires.

Syntax
phys_library(library_name_id) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            ...
        }
    }
}

Example
extension_wire_spacing_rule() {
    ...
}

Groups
extension_wire_qualifier
min_total_projection_length_qualifier
spacing_check_qualifier
extension_wire_qualifier Group

The extension_wire_qualifier group defines an extension wire.

Syntax

phys_library(library_name_id) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            extension_wire_qualifier () {
                ...
            }
        }
    }
}

Simple Attributes

connected_to_fat_wire
corner_wire
not_connected_to_fat_wire

connected_to_fat_wire Simple Attribute

The connected_to_fat_wire attribute specifies whether a wire connected to a fat wire within the fat wire's extension range is an extension wire.

Syntax

phys_library(library_name_id) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            extension_wire_qualifier () {
                connected_to_fat_wire : valueBoolean;
                ...
            }
        }
    }
}

value

Valid values are TRUE and FALSE.

Example

connected_to_fat_wire : ;

corner_wire Simple Attribute

The corner_wire attribute specifies whether a wire located in the corner of a fat wire's extension range is an extension wire.
Syntax

phys_library(library_nameid) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            extension_wire_qualifier() {
                corner_wire : valueBoolean;
                ...
            }
        }
    }
}

value

Valid values are TRUE and FALSE.

Example

corner_wire : ;

not_connected_to_fat_wire Simple Attribute

The not_connected_to_fat_wire attribute specifies whether a wire that is not within a fat wire's extension range is an extension wire.

Syntax

phys_library(library_nameid) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            extension_wire_qualifier() {
                not_connected_to_fat_wire : valueBoolean;
                ...
            }
        }
    }
}

value

Valid values are TRUE and FALSE.

Example

not_connected_to_fat_wire : ;

min_total_projection_length_qualifier Group

The min_total_projection_length_qualifier group defines the projection length.

Syntax

phys_library(library_nameid) {
    topological_design_rules() {
    ....
    }
}
extension_wire_spacing_rule() {
    min_total_projection_length_qualifier () {
        ...
    }
}

**Simple Attributes**

non_overlapping_projection
overlapping_projection
parallel_length

**non_overlapping_projection Simple Attribute**

The `non_overlapping_projection` attribute specifies whether the extension wire spacing rule includes the non-overlapping projection length between non-overlapping extension wires.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            extension_wire_qualifier () {
                non_overlapping_projection : valueBoolean ;
            }
        }
    }
}
```

**value**

Valid values are TRUE and FALSE.

**Example**

```plaintext
non_overlapping_projection : ;
```

**overlapping_projection Simple Attribute**

The `overlapping_projection` attribute specifies whether the extension wire spacing rule includes the overlapping projection length between non-overlapping extension wires.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            extension_wire_qualifier () {
                overlapping_projection : valueBoolean ;
            }
        }
    }
}
```
value

Valid values are TRUE and FALSE.

Example
overlapping_projection : ;

parallel_length Simple Attribute

The parallel_length attribute specifies whether the extension wire spacing rule includes the parallel length between extension wires.

Syntax
phys_library(library_name_id) {
  topological_design_rules() {
    extension_wire_spacing_rule() {
      extension_wire_qualifier() {
        parallel_length : valueBoolean ;
        ...
      }
    }
  }
}

value

Valid values are TRUE and FALSE.

Example
parallel_length : ;

spacing_check_qualifier Group

The spacing_check_qualifier group specifies...

Syntax
phys_library(library_name_id) {
  topological_design_rules() {
    extension_wire_spacing_rule() {
      spacing_check_qualifier() {
        ...
      }
    }
  }
}
Simple Attributes

- corner_to_corner
- non_overlapping_projection_wire
- overlapping_projection_wires
- wires_to_check

**corner_to_corner Simple Attribute**

The `corner_to_corner` attribute specifies whether the extension wire spacing rule includes the corner-to-corner spacing between two extension wires.

**Syntax**

```plaintext
phys_library(library_name_id) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            extension_wire_qualifier() {
                corner_to_corner : value Boolean ;
                ...
            }
        }
    }
}
```

**value**

Valid values are TRUE and FALSE.

**Example**

```plaintext
corner_to_corner : TRUE ;
```

**non_overlapping_projection_wire Simple Attribute**

The `non_overlapping_projection_wire` attribute specifies whether the extension wire spacing rule includes the spacing between two non-overlapping extension wires.

**Syntax**

```plaintext
phys_library(library_name_id) {
    topological_design_rules() {
        extension_wire_spacing_rule() {
            extension_wire_qualifier() {
                non_overlapping_projection_wire : value Boolean ;
                ...
            }
        }
    }
}
```

**value**

Valid values are TRUE and FALSE.
Example
non_overlapping_projection_wire : TRUE ;

overlapping_projection_wires Simple Attribute

The overlapping_projection_wires attribute specifies whether the extension wire spacing rule includes the spacing between two overlapping extension wires.

Syntax
phys_library(library_nameid) {
   topological_design_rules() {
      extension_wire_spacing_rule() {
         extension_wire_qualifier () {
            overlapping_projection_wires : valueBoolean ;
            ...
         }
      }
   }
}

value
Valid values are TRUE and FALSE.

Example
overlapping_projection_wires : TRUE ;

wires_to_check Simple Attribute

The wires_to_check attribute specifies whether the extension wire spacing rule includes the spacing between any two wires or only between extension wires.

Syntax
phys_library(library_nameid) {
   topological_design_rules() {
      extension_wire_spacing_rule() {
         extension_wire_qualifier () {
            wires_to_check : valueenum ;
            ...
         }
      }
   }
}

value
Valid values are all_wires and extension_wires.

Example
wires_to_check : all_wires ;
stack_via_max_current Group

Use the stack_via_max_current group to define the values for current passing through a via stack.

Syntax

\[
\text{phys_library(library_nameid)} \{ \\
\quad \text{topological_design_rules()} \{ \\
\quad\quad \text{stack_via_max_current (nameid)} \{ \\
\quad\quad\quad \ldots \\
\quad\quad \} \\
\quad \} \\
\}
\]

name

Specifies a stack name.

Example

\[
\text{stack_via_max_current( ) } \{ \\
\quad \ldots \\
\}
\]

Simple Attributes

bottom_routing_layer
top_routing_layer

Groups

max_current_ac_absavg
max_current_ac_avg
max_current_ac_peak
max_current_ac_rms
max_current_dc_avg

bottom_routing_layer Simple Attribute

The attribute specifies the bottom_routing_layer.

Syntax

\[
\text{phys_library(library_nameid)} \{ \\
\quad \ldots \\
\quad \text{topological_design_rules()} \{ \\
\quad\quad \text{stack_via_max_current (nameid)} \{ \\
\quad\quad\quad \text{bottom_routing_layer : layer_nameid ;} \\
\quad\quad\quad \ldots \\
\quad\quad \} \\
\quad \} \\
\}
\]
layer_name

A string value representing the routing layer name.

Example

bottom_routing_layer : ;

top_routing_layer Simple Attribute

The top_routing_layer attribute specifies the top_routing_layer.

Syntax

phys_library(library_nameid) {
  ... 
  topological_design_rules() {
    stack_via_max_current (nameid) {
      ...
      top_routing_layer : layer_nameid ;
      ...
    }
    ...
  }
}

layer_name

A string value representing the routing layer name.

Example

top_routing_layer : ;

max_current_ac_absavg Group

Use this group to specify the absolute average value for the AC current that can pass through a cut.

Syntax

phys_library(library_nameid) {
  topological_design_rules() {
    stack_via_max_current (nameid) {
      ...
      max_current_ac_absavg(template_nameid) {
        ...
      }
      ...
    }
  }
}
**template_name**

The name of the contact layer.

**Example**

```plaintext
max_current_ac_absavg() {
    ...
}
```

**Complex Attributes**

- index_1
- index_2
- index_3
- values

**max_current_ac_avg Group**

Use this group to specify an average value for the AC current that can pass through a cut.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        stack_via_max_current(nameid) {
            ...
            max_current_ac_avg(template_nameid) {
                ...
            }
        }
    }
}
```

**template_name**

The name of the contact layer.

**Example**

```plaintext
max_current_ac_avg() {
    ...
}
```

**Complex Attributes**

- index_1
- index_2
- index_3
- values

**max_current_ac_peak Group**

Use this group to specify a peak value for the AC current that can pass through a cut.
Syntax

phys_library(library_nameid) {
  topological_design_rules() {
    stack_via_max_current (nameid) {
      ...
      max_current_ac_peak(template_nameid) {
        ...
      }
      ...
    }
  }
}

template_name

The name of the contact layer.

Example

max_current_ac_peak() {
  ...
}

Complex Attributes

index_1
index_2
index_3
values

max_current_ac_rms Group

Use this group to specify a root mean square value for the AC current that can pass through a cut.

Syntax

phys_library(library_nameid) {
  topological_design_rules() {
    stack_via_max_current (nameid) {
      ...
      max_current_ac_rms(template_nameid) {
        ...
      }
      ...
    }
  }
}

template_name

The name of the contact layer.

Example

max_current_ac_rms() {
Complex Attributes

index_1
index_2
index_3
values

max_current_dc_avg Group

Use this group to specify an average value for the DC current that can pass through a cut.

Syntax

phys_library(library_nameid) {
    topological_design_rules() {
        stack_via_max_current (nameid) {
            ...
            max_current_dc_avg(template_nameid) {
                ...
            }
        }
    }
}

template_name

    The name of the contact layer.

Example

max_current_dc_avg() {
    ...
}

Complex Attributes

index_1
index_2
values

via_rule Group

Use this group to define vias used at the intersection of special wires. You can have multiple via_rule groups for a given layer pair.

Syntax

phys_library(library_nameid) {
    topological_design_rules() {
        via_rule(via_rule_nameid) {
            ...
        }
    }
}
via_rule_name

Specifies a via rule name.

Example

```plaintext
via_rule(crossm1m2) {
  ...
}
```

Simple Attribute
via_list

Group
routing_layer_rule

via_list Simple Attribute

The `via_list` attribute specifies a list of vias. The router selects the first via that satisfies the routing layer rules.

Syntax

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    via_rule(via_rule_name_id) {
      via_list : "via_name1_id; ...
    }
  }
}
```

`via_name1, ..., via_nameN`

Specify the via values used in the selection process.

Example

```plaintext
via_list : "via12, via23";
```

routing_layer_rule Group

Use this group to specify the criteria for selecting a via from a list you specify with the `vias` attribute.
Syntax

```
phys_library(library_name_id) {
    topological_design_rules() {
        via_rule(via_rule_name_id) {
            routing_layer_rule(layer_name_id) {
                ...
            }
        }
    }
}
```

`layer_name`

Specifies the name of a routing layer that the via connects to.

Example

```
routing_layer_rule(metal1) {
    ...
}
```

Simple Attributes

- `contact_overhang`
- `max_wire_width`
- `min_wire_width`
- `metal_overhang`
- `routing_direction`

`contact_overhang` Simple Attribute

The `contact_overhang` attribute specifies the amount of metal (wire) between a contact and a via edge in the specified routing direction on all routing layers.

Syntax

```
phys_library(library_name_id) {
    topological_design_rules() {
        via_rule(via_rule_name_id) {
            routing_layer_rule(layer_name_id) {
                contact_overhang : value_float ;
                ...
            }
        }
    }
}
```

`value`

A floating-point number that represents the value of the overhang.
Chapter 5: Specifying Groups in the topological_design_rules Group

Syntax for Groups in the topological_design_rules Group 5-29

Example
contact_overhang : 9.000e-02 ;

max_wire_width Simple Attribute

Use this attribute along with the min_wire_width attribute to define the range of wire widths subject to these via rules.

Syntax
phys_library(library_nameid) {  
topological_design_rules() {  
via_rule(via_rule_nameid) {  
routing_layer_rule(layer_nameid) {  
max_wire_width : value_float ;  
...  
}  
}  
}  
}

value
A floating-point number that represents the value for the maximum wire width.

Example
max_wire_width : 1.2 ;

min_wire_width Simple Attribute

Use this attribute along with the max_wire_width attribute to define the range of wire widths subject to these via rules.

Syntax
phys_library(library_nameid) {  
topological_design_rules() {  
via_rule(via_rule_nameid) {  
routing_layer_rule(layer_nameid) {  
min_wire_width : value_float ;  
...  
}  
}  
}  
}

value
A floating-point number that represents the value for the minimum wire width.
Example

min_wire_width : 0.4 ;

**metal_overhang Simple Attribute**

The `metal_overhang` attribute specifies the amount of metal (wire) at the edges of wire intersection on all routing layers of the via_rule in the specified routing direction.

**Syntax**

```liberty
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule(via_rule_nameid) {
            routing_layer_rule(layer_nameid) {
                metal_overhang : value<float> ;
            }
        }
    }
}
```

**value**

A floating-point number that represents the value of the overhang.

Example

metal_overhang : 0.0 ;

**routing_direction Simple Attribute**

The `routing_direction` attribute specifies the preferred routing direction for metal that extends to make the overhang and metal overhang on all routing layers.

**Syntax**

```liberty
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule(via_rule_nameid) {
            routing_layer_rule(layer_nameid) {
                routing_direction : value<enum> ;
            }
        }
    }
}
```

**value**

Valid values are `horizontal` and `vertical`. 
Example

```
routing_direction : horizontal ;
```

**via_rule_generate Group**

Use this group to specify the formula for generating vias when they are needed in the case of special wiring. You can have multiple `via_rule_generate` groups for a given layer pair.

**Syntax**

```
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule_generate(via_rule_generate_nameid) {
            ...
        }
    }
}
```

**via_rule_generate_name**

The name for the `via_rule_generate` group.

Example

```
via_rule_generate(via12gen) {
    ...
}
```

**Simple Attributes**

- capacitance
- inductance
- resistance
- res_temperature_coefficient

**Groups**

- contact_formula
- routing_layer_formula

**capacitance Simple Attribute**

The capacitance attribute specifies the capacitance per cut.

**Syntax**

```
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule_generate(via_nameid) {
            capacitance : value_float ;
            ...
        }
    }
}
```
value
A floating-point number that represents the capacitance value.

Example
capacitance : 0.02 ;

inductance Simple Attribute
The inductance attribute specifies the inductance per cut.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {
    via_rule_generate(via_nameid) {
      inductance : value float;
    }
    ...
  }
}
value
A floating-point number that represents the inductance value.

Example
inductance : 0.03 ;

resistance Simple Attribute
The resistance attribute specifies the aggregate resistance per contact rectangle.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {
    via_rule_generate(via_nameid) {
      resistance : value float;
    }
    ...
  }
}
value
A floating-point number that represents the resistance value.
Example
resistance : 0.0375 ;

res_temperature_coefficient Simple Attribute
The res_temperature_coefficient attribute specifies the first-order correction to the resistance per square when the operating temperature does not equal the nominal temperature.

Syntax
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule_generate(via_nameid) {
            res_temperature_coefficient : valuefloat ;
        }
    }
}

value
A floating-point number that represents the coefficient.

Example
res_temperature_coefficient : 0.0375 ;

contact_formula Group
Use this group to specify the contact-layer geometry-generation formula for the generated via.

Syntax
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule_generate(via_rule_generate_nameid) {
            contact_formula(contact_layer_nameid) {
                ... 
            }
        }
    }
}

contact_layer_name
The name of the associated contact layer.

Example
contact_formula(cut23) {
    ...
Simple Attributes
max_cut_rows_current_direction
min_number_of_cuts
resistance
routing_direction

Complex Attributes
contact_array_spacing
contact_spacing
max_cuts
rectangle

max_cut_rows_current_direction Simple Attribute
Use this attribute to specify the maximum number of rows of cuts, in the current routing direction, in a non-turning via for global wire (power and ground).

Syntax
phys_library(library_nameid) {
   topological_design_rules() {
      via_rule_generate(via_rule_generate_nameid) {
         contact_formula(contact_layer_nameid) {
            max_cut_rows_current_direction : valueint ;
         }
      }
   }
}

value
An integer representing the maximum number of rows of cuts in a via.

Example
max_cut_rows_current_direction : 3 ;

min_number_of_cuts Simple Attribute
Use this attribute to specify attribute specifies the minimum number of cuts.

Syntax
phys_library(library_nameid) {
   topological_design_rules() {
      via_rule_generate(via_rule_generate_nameid) {
         contact_formula(contact_layer_nameid) {
            min_number_of_cuts : valueint ;
         }
      }
   }
}

value
value

An integer representing the minimum number of cuts.

Example

min_number_of_cuts : 2;

resistance Simple Attribute

The resistance attribute specifies the aggregate resistance per contact cut.

Syntax

phys_library(library_nameid) { 
  topological_design_rules() { 
    via_rule_generate(via_rule_generate_nameid) { 
      contact_formula(contact_layer_nameid) 
        resistance : valuefloat ;
      ... 
    }
  }
}

value

A floating-point number representing the aggregate resistance.

Example

resistance : 1.0 ;

routing_direction Simple Attribute

The routing_direction attribute specifies the preferred routing direction, which serves as the direction of extension for contact_overlap and metal_overhang on all of the generated via routing layers.

Syntax

phys_library(library_nameid) { 
  topological_design_rules() { 
    via_rule_generate(via_rule_generate_nameid) { 
      contact_formula(contact_layer_nameid) 
        routing_direction : valueenum ;
      ... 
    }
  }
}
value

Valid values are \texttt{horizontal} and \texttt{vertical}.

Example

\begin{verbatim}
routing_direction : vertical ;
\end{verbatim}

contact_array_spacing Complex Attribute

The \texttt{contact_array} attribute specifies the spacing between two contact arrays.

Syntax

\begin{verbatim}
phys_library\{library_name_id\} { 
  topological_design_rules() { 
    via_rule_generate\{via_rule_generate_name_id\} { 
      contact_formula\{contact_layer_name_id\} { 
        contact_array_spacing\{xfloat, yfloat\} ; 
      } 
    } 
  } 
}
\end{verbatim}

\texttt{x, y}

Floating-point numbers that represent the spacing value.

Example

\begin{verbatim}
contact_array_spacing\{ 0.0 \} ;
\end{verbatim}

contact_spacing Complex Attribute

The \texttt{contact_spacing} attribute specifies the center-to-center spacing for generating an array of contact cuts in the generated via.

Syntax

\begin{verbatim}
phys_library\{library_name_id\} { 
  topological_design_rules() { 
    via_rule_generate\{via_rule_generate_name_id\} { 
      contact_formula\{contact_layer_name_id\} { 
        contact_spacing\{xfloat, yfloat\} ; 
      } 
    } 
  } 
}
\end{verbatim}
$x, y$

Floating-point numbers that represent the spacing value in terms of the $x$ distance and $y$ distance between the centers of two contact cuts.

**Example**

```plaintext
contact_spacing(0.84, 0.84);
```

**max_cuts Complex Attribute**

The `max_cuts` attribute specifies the maximum number of cuts.

**Syntax**

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    via_rule_generate(via_rule_generate_name_id) {
      contact_formula(contact_layer_name_id) {
        max_cuts(x_int, y_int);
        ...
      }
    }
  }
}
```

$x, y$

Integer numbers that represent the number of cuts.

**Example**

```plaintext
max_cuts();
```

**rectangle Complex Attribute**

The `rectangle` attribute specifies the dimension of the contact cut.

**Syntax**

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    via_rule_generate(via_rule_generate_name_id) {
      contact_formula(contact_layer_name_id) {
        rectangle(x1_float, y1_float, x2_float, y1_float);
        ...
      }
    }
  }
}
```
$x_1, y_1, x_2, y_2$

Floating-point numbers that specify the coordinates for the diagonally opposite corners of the rectangle.

**Example**

```plaintext
rectangle(-0.3, -0.3, 0.3, 0.3);
```

**routing_formula Group**

Use this group to specify properties for the routing layer. You must specify a `routing_formula` group for each routing layer associated with a via; typically, two routing layers are associated with a via.

**Syntax**

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    via_rule_generate(via_rule_generate_name_id) {
      routing_formula(layer_name_id) {
        ...
      }
    }
  }
}
```

**layer_name**

The name of the associated routing layer.

**Example**

```plaintext
routing_formula(metal1) {
  ...
}
routing_formula(metal2) {
  ...
}
```

**Simple Attributes**

- contact_overhang
- max_wire_width
- min_wire_width
- metal_overhang
- routing_direction

**Complex Attribute**

enclosure
**contact_overhang Simple Attribute**

The `contact_overhang` attribute specifies the minimum amount of metal (wire) extension between a contact and a via edge in the specified direction.

**Syntax**

```plaintext
phys_library(library_name_id) {
    topological_design_rules() {
        via_rule_generate(via_rule_generate_name_id) {
            routing_formula(layer_name_id) {
                contact_overhang : valuefloat;
                ...
            }
        }
    }
}
```

**value**

A floating-point number representing the amount of contact overhang.

**Example**

```plaintext
contact_overhang : 9.000e-01;
```

**max_wire_width Simple Attribute**

Use this attribute along with the `min_wire_width` attribute to define the range of wire widths subject to these via generation rules.

**Syntax**

```plaintext
phys_library(library_name_id) {
    topological_design_rules() {
        via_rule_generate(via_rule_generate_name_id) {
            routing_formula(layer_name_id) {
                max_wire_width : valuefloat;
                ...
            }
        }
    }
}
```

**value**

A floating-point number representing the maximum wire width.

**Example**

```plaintext
max_wire_width : 2.4;
```
min_wire_width Simple Attribute

Use this attribute along with the max_wire_width attribute to define the range of wire widths subject to these via generation rules.

Syntax

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule_generate(via_rule_generate_nameid) {
            routing_formula(layer_nameid) {
                min_wire_width : valuefloat ;
                ...
            }
        }
    }
}
```

value

A floating-point number representing the minimum wire width.

Example

```plaintext
min_wire_width : 1.4 ;
```

metal_overhang Simple Attribute

The metal_overhang attribute specifies the minimum amount of metal overhang at the edges of wire intersections in the specified direction.

Syntax

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule_generate(via_rule_generate_nameid) {
            routing_formula(layer_nameid) {
                metal_overhang : valuefloat ;
                ...
            }
        }
    }
}
```

value

A floating-point number representing the amount of metal overhang.

Example

```plaintext
metal_overhang : 0.1 ;
```
**routing_direction Simple Attribute**

The `routing_direction` attribute specifies the preferred routing direction, which serves as the direction of extension for `contact_overlap` and `metal_overhang` on all of the generated via routing layers.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule_generate(via_rule_generate_nameid) {
            routing_formula(layer_nameid) {
                routing_direction : valueenum;
                ...
            }
        }
    }
}
```

**value**

Valid values are `horizontal` and `vertical`.

**Example**

```plaintext
routing_direction : vertical;
```

**enclosure Complex Attribute**

The `enclosure` attribute specifies the dimensions of the routing layer enclosures.

**Syntax**

```plaintext
phys_library(library_nameid) {
    topological_design_rules() {
        via_rule_generate(via_rule_generate_nameid) {
            routing_formula(layer_nameid) {
                enclosure(value_1float, value_2float);
                ...
            }
        }
    }
}
```

**value_1, value_2**

Floating-point number representing the enclosure dimensions.

**Example**

```plaintext
enclosure (0.0, 0.0);
```
**wire_rule Group**

Use this group to specify the nondefault wire rules for regular wiring.

**Syntax**

```
phys_library(library_name_id) {
    topological_design_rules() {
        wire_rule(wire_rule_name_id) {
            ...
        }
    }
}
```

*wire_rule_name*

The name of the wire rule group.

**Example**

```
wire_rule(rule1) {
    ...
}
```

**Groups**

**layer_rule**

via

**layer_rule Group**

Use this group to specify properties for each routing layer. The width and spacing specifications in this group override the default values defined in the `routing_layer` group in the `resource` group. If the extension is not specified or if the extension has a nonzero value less than half the routing width, then a default extension of half the routing width for the layer is used.

**Syntax**

```
phys_library(library_name_id) {
    topological_design_rules() {
        wire_rule(wire_rule_name_id) {
            layer_rule(layer_name_id) {
                ...
            }
        }
    }
}
```

*layer_name*

The name of the layer defined in the wire rule.
Example
layer_rule(metal1) {
  ...
}

Simple Attributes
min_spacing
wire_extension
wire_width

Complex Attribute
same_net_min_spacing

min_spacing Simple Attribute
The min_spacing attribute specifies the minimum spacing for regular wires that are on the specified layer, subject to the wire rule, and belonging to different nets.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {
    wire_rule(wire_rule_nameid) {
      layer_rule(layer_nameid) {
        min_spacing : valuefloat ;
        ...
      }
    }
  }
}

value
A floating-point number representing the spacing value.

Example
min_spacing : 0.4 ;

wire_extension Simple Attribute
The wire_extension attribute specifies a default distance value for extending wires at vias for regular wires on this layer subject to the wire rule. A value of 0 indicates no wire extension. If the value is less than half the wire_width value, the router uses half the value of the wire_width attribute as the wire extension value. If the wire_width attribute is not defined, the router uses the default value declared in the routing_layer group.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {

wire_rule(wire_rule_nameid) {
    layer_rule(layer_nameid) {
        wire_extension : value_float;
        ...
    }
}

value
A floating-point number that represents the wire extension value.

Example
wire_extension : 0.25;

wire_width Simple Attribute
The wire_width attribute specifies the wire width for regular wires that are on the specified layer and are subject to the wire rule. The wire_width value must be equivalent to or more than the default_wire_width value defined in the layer group.

Syntax
phys_library(library_nameid) {
    topological_design_rules() {
        wire_rule(wire_rule_nameid) {
            layer_rule(layer_nameid) {
                wire_width : value_float;
                ...
            }
        }
    }
}

value
A floating-point number representing the width value.

Example
wire_width : 0.4;

same_net_min_spacing Complex Attribute
The same_net_min_spacing attribute specifies the minimum spacing required between wires on a layer or on two layers in the same net.

Syntax
phys_library(library_nameid) {
    topological_design_rules() {


wire_rule(wire_rule_nameid) {
    layer_rule(layer_nameid) {
        ...  
        same_net_min_spacing(layer1_nameid,  
                               layer2_nameid,  
                               spacefloat,  
                               is_stackBoolean) ;  
    }  
}  
}

layer1_name, layer2_name  
Specify two routing layers. To specify spacing between wires on the same layer, use the same name for both layer1_name and layer2_name.

space  
A floating-point number representing the minimum spacing.

is_stack  
Valid values are TRUE and FALSE. Set the value to TRUE to allow stacked vias at the routing layer. When set to TRUE, the same_net_min_spacing value can be 0 (complete overlap) or the value held by the min_spacing attribute.

Example  
same_net_min_spacing(m2, m2, 0.4, false);

via Group  
Use this group to specify the via that the router uses for this wire rule.

Syntax  
phys_library(library_nameid) {  
    topological_design_rules() {  
        wire_rule(wire_rule_nameid) {  
            via(via_nameid) {  
                ...  
            }  
        }  
    }  
}

via_name  
Specifies the via name.

Example  
via(non_default_via12) {
    ...
}
Simple Attributes

- capacitance
- inductance
- res_temperature_coefficient
- resistance

Complex Attribute

- same_net_min_spacing

Groups

- foreign
- via_layer

capacitance Simple Attribute

The `capacitance` attribute specifies the capacitance per cut.

Syntax

```plaintext
phys_library(library_nameid) {
  topological_design_rules() {
    wire_rule(wire_rule_nameid) {
      via(via_nameid) {
        capacitance : valuefloat ;
        ...
      }
    }
  }
}
```

`value`

A floating-point number that represents the capacitance per cut.

Example

```plaintext
capacitance : 0.2 ;
```

inductance Simple Attribute

The `inductance` attribute specifies the inductance per cut.

Syntax

```plaintext
phys_library(library_nameid) {
  topological_design_rules() {
    wire_rule(wire_rule_nameid) {
      via(via_nameid) {
        inductance : valuefloat ;
        ...
      }
    }
  }
}
```
value

A floating-point number that represents the inductance per cut.

Example
inductance : 0.03 ;

res_temperature_coefficient Simple Attribute

Use this attribute to specify the first-order temperature coefficient for the resistance.

Syntax
phys_library(library_name_id) {
   topological_design_rules() {
      wire_rule(wire_rule_name_id) {
         via(via_name_id) {
            res_temperature_coefficient : value_float ;
            ...
         }
      }
   }
}
value

A floating-point number that represents the temperature coefficient.

Example
res_temperature_coefficient : 0.0375 ;

resistance Simple Attribute

The resistance attribute specifies the aggregate resistance per contact cut.

Syntax
phys_library(library_name_id) {
   topological_design_rules() {
      wire_rule(wire_rule_name_id) {
         via(via_name_id) {
            resistance : value_float ;
            ...
         }
      }
   }
}
value

A floating-point number representing the resistance.

Example
resistance : 1.000e+00 ;

same_net_min_spacing Complex Attribute

The same_net_min_spacing attribute specifies the minimum spacing required between wires on a layer or on two layers in the same net.

Syntax
phys_library(library_name_id) {  
topological_design_rules() {  
wire_rule(wire_rule_name_id) {  
via(via_name_id) {  
    same_net_min_spacing(layer1_name_id,  
                        layer2_name_id,  
                        space_float,  
                        is_stack_Boolean) ;  
  }  
}  
}
}

layer1_name, layer2_name

Specify two routing layers. To specify spacing between wires on the same layer, use the same name for both layer1_name and layer2_name.

space

A floating-point number representing the minimum spacing.

is_stack

Valid values are TRUE and FALSE. Set the value to TRUE to allow stacked vias at the routing layer. When set to TRUE, the same_net_min_spacing value can be 0 (complete overlap) or the value held by the min_spacing attribute.

Example
same_net_min_spacing(m2, m2, 0.4, false);

foreign Group

The foreign attribute specifies which GDSII structure (model) to use when an instance of a via is placed.

Note:
Only one foreign group is allowed for each via.
Syntax

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    wire_rule(wire_rule_name_id) {
      via(via_name_id) {
        foreign(foreign_object_name_id) {
          ...
        }
      }
    }
  }
}
```

**foreign_object_name**

The name of a GDSII structure (model).

**Example**

```plaintext
foreign(fdesf2a6) {
  ...
}
```

**Simple Attribute**

orientation

**Complex Attribute**

origin

**orientation Simple Attribute**

The `orientation` attribute specifies the orientation of a foreign object.

**Syntax**

```plaintext
phys_library(library_name_id) {
  topological_design_rules() {
    wire_rule(wire_rule_name_id) {
      via(via_name_id) {
        foreign(foreign_object_name_id) {
          orientation : valueenum ;
          ...
        }
      }
    }
  }
}
```

**value**

Valid values are `N` (north), `E` (east), `S` (south), `W` (west), `FN` (flip north), `FE` (flip east), `FS` (flip south), and `FW` (flip west), as shown in Figure 5-1.


**Figure 5-1 Orientation Examples**

![Orientation Examples](image)

**Example**

```
orientation : FN ;
```

**origin Complex Attribute**

The `origin` attribute specifies the equivalent coordinates for the origin of a placed foreign object.

**Syntax**

```
phys_library(library_nameid) {
    topological_design_rules() {
        wire_rule(wire_rule_nameid) {
            via(via_nameid) {
                foreign(foreign_object_nameid) {
                    ...,
                    origin(num_xfloat, num_yfloat) ;
                }
            }
        }
    }
}
```

`num_x, num_y`

Floating-point numbers that specify the coordinates where the foreign object is placed.

**Example**

```
origin(-1, -1) ;
```
via_layer Group

Use this group to specify a via layer. A via can have one or more via_layer groups.

Syntax

```
phys_library(library_nameid) {
  topological_design_rules() {
    wire_rule(wire_rule_nameid) {
      via(via_nameid) {
        via_layer(via_layerid) {
          ...
        }
      }
    }
  }
}
```

via_layer

A predefined layer name.

Example

```
via_layer(via23) {
  ...
}
```

Complex Attribute

rectangle

rectangle Complex Attribute

The rectangle attribute specifies the geometry of the via on the layer.

Syntax

```
phys_library(library_nameid) {
  topological_design_rules() {
    wire_rule(wire_rule_nameid) {
      via(via_nameid) {
        via_layer(via_layerid) {
          rectangle(x1float, y1float, x2float, y2float) ;
        }
      }
    }
  }
}
```

x1, y1, x2, y2

Floating-point numbers that specify the coordinates for the diagonally opposite corners of the rectangle.
Example
rectangle(-0.3, -0.3, 0.3, 0.3);

wire_slotting_rule Group

Use this group to specify the wire slotting rules to satisfy the maximum metal density design rule.

Syntax
phys_library(library_name_id) {
  topological_design_rules() {
    wire_slotting_rule(routing_layer_name_id) {
      ...
    }
  }
}

Simple Attributes
max_metal_density
min_length
min_width

Complex Attributes
slot_length_range
slot_length_side_clearance
slot_length_wise_spacing
slot_width_range
slot_width_side_clearance
slot_width_wise_spacing

max_metal_density Simple Attribute

Use this attribute to specify the maximum metal density for a slotted layer, as a percentage of the layer.

Syntax
phys_library(library_name_id) {
  topological_design_rules() {
    wire_slotting_rule(routing_layer_name_id) {
      max_metal_density : value_float;
    }
  }
}

value

A floating-point number that represents the percentage.
**Example**

max\_metal\_density : 0.70 ;

**min\_length Simple Attribute**

The `min\_length` attribute specifies the the minimum geometry length threshold that triggers slotting. Slotting is triggered when the thresholds specified by the `min\_length` and `min\_width` attributes are both surpassed.

**Syntax**

```liberty
phys_library(library\_name) {
  topological\_design\_rules() {
    wire\_slotting\_rule(routing\_layer\_name) {
      min\_length : value\_float;
    }
  }
}
```

**value**

A floating-point number that represents the minimum geometry length threshold.

**Example**

min\_length : 0.5 ;

**min\_width Simple Attribute**

The `min\_width` attribute specifies the the minimum geometry length threshold that triggers slotting. Slotting is triggered when the thresholds specified by the `min\_length` and `min\_width` attributes are both surpassed.

**Syntax**

```liberty
phys_library(library\_name) {
  topological\_design\_rules() {
    wire\_slotting\_rule(routing\_layer\_name) {
      min\_width : value\_float;
    }
  }
}
```

**value**

A floating-point number that represents the minimum geometry width threshold.

**Example**

min\_width : 0.4 ;
**slot_length_range Complex Attribute**

The `slot_length` attribute specifies the allowable range for the length of a slot.

**Syntax**

```plaintext
phys_library(library_name_id) {
    topological_design_rules() {
        wire_slotting_rule(routing_layer_name_id) {
            slot_length_range (min_value, max_value);
        }
    }
}
```

*min_value, max_value*

Floating-point numbers that represent the minimum and maximum range values.

**Example**

```plaintext
slot_length_range (0.2, 0.3);
```

---

**slot_length_side_clearance Complex Attribute**

Use this attribute to specify the spacing from the end edge of a wire to its outermost slot.

**Syntax**

```plaintext
phys_library(library_name_id) {
    topological_design_rules() {
        wire_slotting_rule(routing_layer_name_id) {
            slot_length_side_clearance (min_value, max_value);
        }
    }
}
```

*min_value, max_value*

Floating-point numbers that represent the minimum and maximum spacing values.

**Example**

```plaintext
slot_length_side_clearance (0.2, 0.4);
```

---

**slot_length_wise_spacing Complex Attribute**

Use this attribute to specify the minimum spacing between adjacent slots in a direction perpendicular to the wire (current flow) direction.

**Syntax**

```plaintext
phys_library(library_name_id) {
    topological_design_rules() {
        wire_slotting_rule(routing_layer_name_id) {
```
slot_length_wise_spacing(min_valuefloat, max_valuefloat);
}
}

min_value, max_value
Floating-point numbers that represent the minimum and maximum spacing distance values.

Example
slot_length_wise_spacing(0.2, 0.3);

slot_width_range Complex Attribute
Use this attribute to specify the allowable range for the width of a slot.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {
    wire_slotting_rule(routing_layer_nameid) {
      slot_width_range(min_valuefloat, max_valuefloat);
    }
  }
}

min_value, max_value
Floating-point numbers that represent the minimum and maximum range values.

Example
slot_width_range(0.2, 0.3);

slot_width_side_clearance Complex Attribute
Use this attribute to specify the spacing from the side edge of a wire to its outermost slot.

Syntax
phys_library(library_nameid) {
  topological_design_rules() {
    wire_slotting_rule(routing_layer_nameid) {
      slot_width_side_clearance(min_valuefloat, max_valuefloat);
    }
  }
}

min_value, max_value

Floating-point numbers that represent the minimum and maximum spacing distance values.

Example

slot_width_side_clearance (0.2, 0.3) ;

slot_width_wise_spacing Complex Attribute

Use this attribute to specify the minimum spacing between slots in a direction perpendicular to the wire (current flow) direction.

Syntax

phys_library(library_name_id) {
  topological_design_rules() {
    wire_slotting_rule(routing_layer_name_id) {
      slot_width_wise_spacing (min_value_float, max_value_float) ;
    }
  }
}

min_value, max_value

Floating-point numbers that represent the minimum and maximum spacing distance values.

Example

slot_width_wise_spacing (0.2, 0.3) ;
You use the `process_resource` group to specify various process corners in a particular process. The `process_resource` group is defined inside the `phys_library` group and must be defined before you model any cell. Multiple `process_resource` groups are allowed in a physical library.

The information in this chapter includes the following:

- Syntax for Attributes in the `process_resource` Group
- Syntax for Groups in the `process_resource` Group
Syntax for Attributes in the process_resource Group

This section describes the attributes that you define in the process_resource group.

**Simple Attributes**

- baseline_temperature
- field_oxide_thickness
- process_scale_factor

**Complex Attribute**

- plate_cap

---

**baseline_temperature Simple Attribute**

Defines a baseline operating condition temperature.

**Syntax**

```plaintext
phys_library(library_name_id) {  
  process_resource(architecture_enum) {  
    ...  
    baseline_temperature : value_float ;  
    ...  
  }  
}
```

**value**

A floating-point number representing the baseline temperature.

**Example**

```plaintext
baseline_temperature : 0.5 ;
```

---

**field_oxide_thickness Simple Attribute**

Specifies the field oxide thickness.

**Syntax**

```plaintext
phys_library(library_name_id) {  
  process_resource(architecture_enum) {  
    ...  
    field_oxide_thickness : value_float ;  
    ...  
  }  
}
```
value
   A positive floating-point number in distance units.

Example
field_oxide_thickness : 0.5 ;

process_scale_factor Simple Attribute
Specifies the factor to describe the process shrinkage factor to scale the length, width, and spacing geometries.

Note:
   Do not specify a value for the process_scale_factor attribute if you specify a value for the shrinkage attribute or shrinkage_table group.

Syntax
phys_library(library_name-id) {
   process_resource(architecture enum) {
      ...
      process_scale_factor : value float ;
      ...
   }
}

value
   A floating-point number representing the scaling factor.

Example
process_scale_factor : 0.96 ;

plate_cap Complex Attribute
Specifies the interlayer capacitance per unit area when a wire on the first routing layer overlaps a wire on the second routing layer.

Note:
   The plate_cap statement must follow all the routing_layer statements and precede the routing_wire_model statements.

Syntax
phys_library(library_name-id) {
   process_resource(architecture enum) {
      ...
      routing_layer(layer_name-id) {
         ...
... } plate_cap(PCAP_l1_l2_float, PCAP_l1_l3_float, PCAP_ln-1_ln_float); routing_wire_model(model_name_id) { ... } }

PCAP_la_lb

Represents a floating-point number that specifies the plate capacitance per unit area between two routing layers, layer a and layer b. The number of PCAP values is determined by the number of previously defined routing layers. You must specify every combination of routing layer pairs based on the order of the routing layers. For example, if the layers are defined as substrate, layer1, layer2, and layer3, then the PCAP values are defined in PCAP_l1_l2, PCAP_l1_l3, and PCAP_l2_l3.

Example

The example shows a plate_cap statement for a library with four layers. The values are indexed by the routing layer order.

plate_cap(0.35, 0.06, 0.0, 0.25, 0.02, 0.15); /* PCAP_1_2, PCAP_1_3, PCAP_1_4, PCAP_2_3, PCAP_2_4, PCAP_3_4 */

---

Syntax for Groups in the process_resource Group

This section describes the groups that you define in the process_resource group.

Groups

- process_cont_layer
- process_routing_layer
- process_via
- process_via_rule_generate
- process_wire_rule

process_cont_layer Group

Specifies values for the process contact layer.

Syntax

phys_library(library_name_id) {
    process_resource(architecture_enum) {
        process_cont_layer(layer_name_id) {

Chapter 6: Specifying Attributes and Groups in the process_resource Group
Layer Name

The name of the contact layer.

Example

```plaintext
process_cont_layer(m1) {
  ...
}
```

---

**process_routing_layer Group**

Use a `process_routing_layer` group to define operating-condition-specific routing layer attributes.

**Syntax**

```plaintext
phys_library(library_nameid) {
  process_resource(architecture_enum) {
    process_routing_layer(layer_nameid) {
      ...
    }
  }
}
```

Layer Name

The name of the scaled routing layer.

**Example**

```plaintext
process_routing_layer(m1) {
  ...
}
```

**Simple Attributes**

- `cap_multiplier`
- `cap_per_sq`
- `coupling_cap`
- `edgecapacitance`
- `fringe_cap`
- `height`
- `inductance_per_dist`
- `lateral_oxide_thickness`
- `oxide_thickness`
- `res_per_sq`
- `shrinkage`
Chapter 6: Specifying Attributes and Groups in the process_resource Group

Syntax for Groups in the process_resource Group 6-6

thickness

Complex Attributes
conformal_lateral_oxide
lateral_oxide

Groups
resistance_table
shrinkage_table

cap_multiplier Simple Attribute
Specifies a scaling factor for interconnect capacitance to account for changes in capacitance due to nearby wires.

Syntax
phys_library(library_name_id) {
    process_resource(architecture_enum) {
        process_routing_layer(layer_name_id) {
            cap_multiplier : valuefloat ;
            ...
        }
    }
}

value
A floating-point number representing the scaling factor.

Example
cap_multiplier : 2.0

cap_per_sq Simple Attribute
Specifies the substrate capacitance per square unit area of a process routing layer.

Syntax
phys_library(library_name_id) {
    process_resource(architecture_enum) {
        process_routing_layer(layer_name_id) {
            cap_per_sq : valuefloat ;
            ...
        }
    }
}
value

A floating-point number that represents the capacitance for a square unit of wire, in picofarads per square distance unit.

Example
cap_per_sq : 5.909e-04 ;

coupling_cap Simple Attribute
Specifies the coupling capacitance per unit length between parallel wires on the same layer.

Syntax
phys_library(library_nameid) {
    process_resource(architectureenum) {
        process_routing_layer(layer_nameid) {
            coupling_cap : valuefloat ;
            ...
        }
    }
}

value

A floating-point number that represents the coupling capacitance.

Example
coupling_cap: 0.000019 ;

dgecapacitance Simple Attribute
Specifies the total peripheral capacitance per unit length of a wire on the process routing layer.

Syntax
phys_library(library_nameid) {
    process_resource(architectureenum) {
        process_routing_layer(layer_nameid) {
            edgecapacitance : valuefloat ;
            ...
        }
    }
}

value

A floating-point number that represents the capacitance per unit length value.
Example
edgecapacitance : 0.00065 ;

fringe_cap Simple Attribute
Specifies the fringe (sidewall) capacitance per unit length of a process routing layer.

Syntax
phys_library(library_nameid) {
  process_resource(architectureenum) {
    process_routing_layer(layer_nameid) {
      fringe_cap : valuefloat ;
    }
  }
}

value
A floating-point number that represents the fringe capacitance.

Example
fringe_cap : 0.00023 ;

height Simple Attribute
Specifies the distance from the top of the substrate to the bottom of the routing layer.

Syntax
phys_library(library_nameid) {
  process_resource(architectureenum) {
    process_routing_layer(layer_nameid) {
      height : valuefloat ;
    }
  }
}

value
A floating-point number representing the distance unit of measure.

Example
height : 1.0 ;

inductance_per_dist Simple Attribute
Specifies the inductance per unit length of a process routing layer.
Syntax
phys_library(library_nameid) {
    process_resource(architecture_enum) {
        process_routing_layer(layer_nameid) {
            inductance_per_dist : valuefloat ;
            ...
        }
    }
}

value
A floating-point number that represents the inductance.

Example
inductance_per_dist : 0.0029 ;

lateral_oxide_thickness Simple Attribute
Specifies the lateral oxide thickness for the layer.

Syntax
phys_library(library_nameid) {
    process_resource(architecture_enum) {
        process_routing_layer(layer_nameid) {
            lateral_oxide_thickness : valuefloat ;
            ...
        }
    }
}

value
A floating-point number that represents the lateral oxide thickness.

Example
lateral_oxide_thickness : 1.33 ;

oxide_thickness Simple Attribute
Specifies the oxide thickness for the layer.

Syntax
phys_library(library_nameid) {
    process_resource(architecture_enum) {
        process_routing_layer(layer_nameid) {
            oxide_thickness : valuefloat ;
            ...
        }
    }
}
value

A floating-point number that represents the oxide thickness.

Example
oxide_thickness : 1.33 ;

res_per_sq Simple Attribute
Specifies the substrate resistance per square unit area of a process routing layer.

Syntax
phys_library(library_nameid) {
  process_resource(architecture_enum) {
    process_routing_layer(layer_nameid) {
      res_per_sq : valuefloat ;
    ...}
  }
}

value

A floating-point number representing the resistance.

Example
res_per_sq : 1.200e-01 ;

shrinkage Simple Attribute
Specifies the total distance by which the wire width on the layer shrinks or expands. The shrinkage parameter is a sum of the shrinkage for each side of the wire. The post-shrinkage wire width represents the final processed silicon width as calculated from the drawn silicon width in the design database.

Note:

Do not specify a value for the shrinkage attribute or shrinkage_table group if you specify a value for the process_scale_factor attribute.

Syntax
phys_library(library_nameid) {
  process_resource(architecture_enum) {
    process_routing_layer(layer_nameid) {
      shrinkage : valuefloat ;
    ...}
  }
}
value
A floating-point number representing the distance unit of measure. A positive number represents shrinkage; a negative number represents expansion.

Example
shrinkage : 0.00046 ;

**thickness Simple Attribute**
Specifies the thickness of the user units of objects process routing layer.

**Syntax**
```plaintext
phys_library(library_nameid) {
    process_resource(architecture_enum) {
        process_routing_layer(layer_nameid) {
            thickness : value<float> ;
            ...
        }
    }
}
```

value
A floating-point number representing the thickness of the routing layer.

Example
thickness : 0.02 ;

**conformal_lateral_oxide Complex Attribute**
Specifies the substrate capacitance per unit area of a process routing layer.

**Syntax**
```plaintext
phys_library(library_nameid) {
    process_resource(architecture_enum) {
        process_routing_layer(layer_nameid) {
            conformal_lateral_oxide : value<float> ;
            ...
        }
    }
}
```
value

A floating-point number that represents the capacitance for a square unit of wire, in picofarads per square distance unit.

Example

conformal_lateral_oxide : 5.909e-04 ;

lateral_oxide Complex Attribute

Specifies the lateral oxide thickness.

Syntax

phys_library(library_nameid) {
    process_resource(architecture enum) {
        process_routing_layer(layer_nameid) {
            lateral_oxide : value float ;
            ...
        }
    }
}

value

A floating-point number representing the lateral oxide thickness.

Example

lateral_oxide : 5.909e-04

resistance_table Group

Use this group to specify an array of values for sheet resistance.

Syntax

phys_library(library_nameid) {
    process_resource(architecture enum) {
        process_routing_layer(layer_nameid) {
            resistance_table(template_nameid) {
                ...
            }
        }
    }
}

Example

resistance_table ( ) {
    ...
}
Complex Attributes
index_1
index_2
values

index_1 and index_2 Complex Attributes
Specifies the default indexes.

Syntax
phys_library(library_name_id) {
  process_resource(architecture_enum) {
    process_routing_layer(layer_name_id) {
      resistance_table(template_name_id) {
        index_1 (value_float, value_float, value_float, ...)
        index_2 (value_float, value_float, value_float, ...)
      }
    }
  }
}

Example
resistance_table (template_name) {
  index_1 ( ) ;
  index_2 ( ) ;
  values ( ) ;

shrinkage_table Group
Specifies a lookup table template.

Syntax
phys_library(library_name_id) {
  process_resource(architecture_enum) {
    process_routing_layer(layer_name_id) {
      shrinkage_table(template_name_id) {
        ...
      }
    }
  }
}

template_name
The name of a shrinkage_lut_template defined at the phys_library level.

Example
shrinkage_table (shrinkage_template_1) {
  ...

Complex Attributes

index_1
index_2
values

index_1 and index_2 Complex Attributes

Specify the default indexes.

Syntax

phys_library(library_nameid) {
  ...
  shrinkage_table(template_nameid) {
    index_1 (value_float, value_float, value_float, ...);
    index_2 (value_float, value_float, value_float, ...);
    ...
  }
  ...
}

value, value, value, ...

Floating-point numbers that represent the default indexes.

Example

shrinkage_lut_template(resistance_template_1) {
  index_1 (0.0, 0.0, 0.0, 0.0);
  index_2 (0.0, 0.0, 0.0, 0.0);
}

process_via Group

Use a process_via group to define an operating-condition-specific resistance value for a via.

Syntax

phys_library(library_nameid) {
  process_resource(architecture_enum) {
    process_via(via_nameid) {
      ...
    }
  }
}

via_name
   The name of the via.

Example
via(via12) {
   ...
}

Simple Attributes
   capacitance
   inductance
   resistance
   res_temperature_coefficient

capacitance Simple Attribute
Specifies the capacitance contact in a cell instance (or over a macro).

Syntax
phys_library(library_name_id) {
   process_resource(architecture_enum) {
      process_via(via_name_id) {
         capacitance : value_float ;
         ...
      }
   }
}

value
   A floating-point number that represents the capacitance.

Example
capacitance : 0.05 ;

inductance Simple Attribute
Specifies the inductance per cut.

Syntax
phys_library(library_name_id) {
   process_resource(architecture_enum) {
      process_via(via_name_id) {
         inductance : value_float ;
      }
   }
}
value
A floating-point number that represents the inductance value.

Example
inductance : 0.03 ;

resistance Simple Attribute
Specifies the aggregate resistance per contact rectangle.

Syntax
phys_library(library_nameid) {
  process_resource(architectureenum) {
    process_via(via_nameid) {
      resistance : value\text{\textit{float}} ;
    }
    ...
  }
}

value
A floating-point number that represents the resistance value.

Example
resistance : 0.0375 ;

res\_temperature\_coefficient Simple Attribute
The res\_temperature\_coefficient attribute specifies the coefficient of the first-order correction to the resistance per square when the operating temperature does not equal the nominal temperature.

Syntax
phys_library(library_nameid) {
  process_resource(architectureenum) {
    process_via(via_nameid) {
      res\_temperature\_coefficient : value\text{\textit{float}} ;
    }
    ...
  }
}

value
A floating-point number that represents the temperature coefficient.
Example
res_temperature_coefficient : 0.03 ;

process_via_rule_generate Group
Use a process_via_rule_generate group to define an operating-condition-specific resistance value for a via.

Syntax
phys_library(library_nameid) {
    process_resource(architectureenum) {
        process_via_rule_generate(via_nameid) {
            
            
        }
    }
}

via_name
The name of the via.

Example
via(via12) {
    
    
}

Simple Attributes
capacitance
inductance
resistance
res_temperature_coefficient

capacitance Simple Attribute
Specifies the capacitance per cut.

Syntax
phys_library(library_nameid) {
    process_resource(architectureenum) {
        process_via_rule_generate(via_nameid) {
            capacitance : valueenum ;
            
        }
    }
}

Chapter 6: Specifying Attributes and Groups in the process_resource Group
Syntax for Groups in the process_resource Group
value

A floating-point number that represents the capacitance value.

Example

capacitance : 0.05 ;

inductance Simple Attribute

Specifies the inductance per cut.

Syntax

phys_library(library_name_id) {
  process_resource(architecture_enum) {
    process_via_rule_generate(via_name_id) {
      inductance : value_float ;
    }
    ...
  }
}

value

A floating-point number that represents the inductance value.

Example

inductance : 0.03 ;

resistance Simple Attribute

Specifies the aggregate resistance per contact rectangle.

Syntax

phys_library(library_name_id) {
  process_resource(architecture_enum) {
    process_via_rule_generate(via_name_id) {
      resistance : value_float ;
    }
    ...
  }
}

value

A floating-point number that represents the resistance.

Example

resistance : 0.0375 ;
**res_temperature_coefficient Simple Attribute**

Specifies the first-order temperature coefficient for the resistance.

**Syntax**

```plaintext
phys_library(library_name_id) {
    process_resource(architecture_enum) {
        process_via_rule_generate(via_name_id) {
            res_temperature_coefficient : value_float ;
        ...}
    }
}
```

**value**

A floating-point number that represents the temperature coefficient.

**Example**

```plaintext
res_temperature_coefficient : 0.0375 ;
```

---

**process_wire_rule Group**

Use this group to define an operating-condition-specific value for a nondefault regular via defined within a wire_rule group.

**Syntax**

```plaintext
phys_library(library_name_id) {
    process_resource() {
        process_wire_rule(wire_rule_name_id) {
            ...
        }
    }
}
```

**wire_rule_name**

The name of the wire rule group.

**Example**

```plaintext
process_wire_rule(rule1) {
    ...
}
```

**Group**

```plaintext
process_via
```
process_via Group

Specifies the via that the router uses for this wire rule.

Syntax

```plaintext
phys_library(library_nameid) {
    process_resource() {
        process_wire_rule(wire_rule_nameid) {
            process_via(via_nameid) {
                ...
            }
        }
    }
}

via_name

Specifies the via name.

Example

```plaintext
process_via(non_default_via12) {
    ...
}
```

Simple Attributes

capacitance
inductance
resistance
res_temperature_coefficient

capacitance Simple Attribute

Specifies the capacitance per cut.

Syntax

```plaintext
phys_library(library_nameid) {
    process_resource() {
        process_wire_rule(wire_rule_nameid) {
            process_via(via_nameid) {
                capacitance : valueenum;
                ...
            }
        }
    }
}

value

A floating-point number that represents the capacitance value.
Example

\[
\text{capacitance} : 0.0 ;
\]

**inductance Simple Attribute**

Specifies the inductance per cut.

**Syntax**

\[
\text{phys_library(} \text{library_name_id)} \{ \\
\quad \text{process_resource(} \{ \\
\quad\quad \text{process_wire_rule(} \text{wire_rule_name_id)} \{ \\
\quad\quad\quad \text{process_via(} \text{via_name_id)} \{ \\
\quad\quad\quad\quad \text{inductance} : \text{value}_{\text{float}} ; \\
\quad\quad\quad\quad \ldots \\
\quad\quad \} \\
\quad\} \\
\} \\
\}
\]

*value*

A floating-point number that represents the inductance value.

**Example**

\[
\text{inductance} : 0.0 ;
\]

**res_temperature_coefficient Simple Attribute**

Specifies the first-order temperature coefficient for the resistance unit area of a routing layer.

**Syntax**

\[
\text{phys_library(} \text{library_name_id)} \{ \\
\quad \text{process_resource(} \{ \\
\quad\quad \text{process_wire_rule(} \text{wire_rule_name_id)} \{ \\
\quad\quad\quad \text{process_via(} \text{via_name_id)} \{ \\
\quad\quad\quad\quad \text{res_temperature_coefficient} : \text{value}_{\text{float}} ; \\
\quad\quad\quad\quad \ldots \\
\quad\quad \} \\
\quad\} \\
\} \\
\}
\]

*value*

A floating-point number that represents the coefficient value.

**Example**

\[
\text{res_temperature_coefficient} : 0.0375 ;
\]
**resistance Simple Attribute**

Specifies the aggregate resistance per contact cut.

**Syntax**

```
phys_library(library_nameid) {
    process_resource() {
        process_wire_rule(wire_rule_nameid) {
            process_via(via_nameid) {
                resistance : valuefloat ;
                ...
            }
        }
    }
}
```

**value**

A floating-point number representing the resistance value.

**Example**

```
resistance : 1.000e+00 ;
```
Specifying Attributes and Groups in the macro Group

For each cell, you use the macro group to specify the macro-level information and pin information. Macro-level information includes such properties as symmetry, size and obstruction. Pin information includes such properties as geometry and position.

This chapter describes the attributes and groups that you define in the macro group, with the exception of the pin group, which is described in Chapter 9.
macro Group

Use this group to specify the physical aspects of the cell.

Syntax
phys_library(library_name) {
  macro(cell_name) {
    ...
  }
}

cell_name
   Specifies the name of the cell.
   Note: This name must be identical to the name of the logical cell_name that you define in the library.

Example
macro(and2) {
  ...
}

Simple Attributes
cell_type
create_full_pin_geometry
eq_cell
extract_via_region_within_pin_area
in_site
in_tile
leq_cell
source
symmetry

Complex Attributes
extract_via_region_from_cont_layer
obs_clip_box
origin
size

Groups
foreign
obs
site_array
pin
**cell_type Simple Attribute**

Use this attribute to specify the cell type.

**Syntax**

```plaintext
phys_library(library_nameid) {
    macro(cell_nameid) {
        cell_type : valueenum ;
        ...
    }
}
```

**value**

See Table 7-1 for value definitions.

**Example**

```plaintext
cell_type : block ;
```

**Table 7-1  cell_type Values**

<table>
<thead>
<tr>
<th>Cell type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>antennadiode_core</td>
<td>Dissipates a manufacturing charge from a diode.</td>
</tr>
<tr>
<td>areaio_pad</td>
<td>Area I/O driver</td>
</tr>
<tr>
<td>blackbox_block</td>
<td>Subclass of block</td>
</tr>
<tr>
<td>block</td>
<td>Predefined macro used in hierarchical design</td>
</tr>
<tr>
<td>bottomleft_endcap</td>
<td>I/O cell placed at bottom-left corner</td>
</tr>
<tr>
<td>bottomright_endcap</td>
<td>I/O cell placed at bottom-right corner</td>
</tr>
<tr>
<td>bump_cover</td>
<td>Subclass of cover</td>
</tr>
<tr>
<td>core</td>
<td>Core cell</td>
</tr>
<tr>
<td>cover</td>
<td>A cover cell is fixed to the floorplan</td>
</tr>
<tr>
<td>feedthru_core</td>
<td>Connects to another cell.</td>
</tr>
<tr>
<td>inout_pad</td>
<td>Bidirectional pad cell</td>
</tr>
<tr>
<td>input_pad</td>
<td>Input pad cell</td>
</tr>
</tbody>
</table>
create_full_pin_geometry Simple Attribute

Use this attribute to specify the full pin geometry.

**Syntax**

```plaintext
phys_library(library_name_id) {
  macro(cell_name_id) {
    create_full_pin_geometry : valueBoolean ;
    ...  
  }
}
```

**value**

Valid values are TRUE and FALSE.

---

**Table 7-1  cell_type Values (continued)**

<table>
<thead>
<tr>
<th>Cell type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_pad</td>
<td>Output pad cell</td>
</tr>
<tr>
<td>pad</td>
<td>I/O cell</td>
</tr>
<tr>
<td>post_endcap</td>
<td>Cell placed at the left or top end of core rows to connect with the power ring</td>
</tr>
<tr>
<td>power_pad</td>
<td>Power pad</td>
</tr>
<tr>
<td>pre_endcap</td>
<td>Cell placed at the right or bottom end of core rows to connect with the power ring</td>
</tr>
<tr>
<td>ring</td>
<td>Blocks that can cut prerouted special nets and connect to these nets with ring pins</td>
</tr>
<tr>
<td>spacer_core</td>
<td>Fills space between regular core cells.</td>
</tr>
<tr>
<td>spacer_pad</td>
<td>Spacer pad</td>
</tr>
<tr>
<td>tiehigh_core</td>
<td>Connects I/O terminals to the power or ground.</td>
</tr>
<tr>
<td>tielow_core</td>
<td>Connects I/O terminals to the power or ground.</td>
</tr>
<tr>
<td>topleft_endcap</td>
<td>I/O cell placed at top-left corner</td>
</tr>
<tr>
<td>topright_endcap</td>
<td>I/O cell placed at top-right corner</td>
</tr>
</tbody>
</table>
Example
create_full_pin_geometry : TRUE ;

eq_cell Simple Attribute
Use this attribute to specify an electrically equivalent cell that has the same functionality, pin positions, and electrical characteristics (such as timing and power) as a previously defined cell.

Syntax
phys_library(library_nameId) {
    macro(cell_nameId) {
        eq_cell : eq_cell_nameId ;
        ...  
    }
}

eq_cell_name
The name of the equivalent cell previously defined in the phys_library group.

Example
eq_cell : and2a ;

extract_via_region_within_pin_area Simple Attribute
Use this attribute to whether to extract via region information from within the pin area only.

Syntax
phys_library(library_nameId) {
    macro(cell_nameId) {
        extract_via_region_within_pin_area : valueBoolean ;
        ...  
    }
}

value
Valid values are TRUE and FALSE (default).

Example
extract_via_region_within_pin_area : TRUE ;
in_site Simple Attribute

Use this attribute to specify the site associated with a cell. The site class and symmetry must match the cell class and symmetry.

Note:

You can use this attribute only with standard cell libraries.

Syntax

phys_library(library_nameid) {
    macro(cell_nameid) {
        in_site : site_nameid ;
        ...
    }
}

site_name

The name of the associated site.

Example

in_site : core ;

in_tile Simple Attribute

The in_tile attribute specifies the tile associated with a cell.

Syntax

phys_library(library_nameid) {
    macro(cell_nameid) {
        in_tile : tile_nameid ;
        ...
    }
}

value

The name of the associated tile.

Example

in_tile : ;
**leq_cell Simple Attribute**

Use this attribute to specify a logically equivalent cell that has the same functionality and pin interface as a previously defined cell. Logically equivalent cells need not have the same electrical characteristics, such as timing and power.

**Syntax**

```plaintext
phys_library(library_nameid) {
    macro(cell_nameid) {
        leq_cell : leq_cell_nameid ;
    }
    ...
}
```

**leq_cell_name**

The name of the equivalent cell previously defined in the `phys_library` group.

**Example**

```plaintext
leq_cell : and2x2 ;
```

**source Simple Attribute**

Use this attribute to specify the source of a cell.

**Syntax**

```plaintext
phys_library(library_nameid) {
    macro(cell_nameid) {
        source : value_enum ;
    }
    ...
}
```

**value**

Valid values are `user` (for a regular cell), `generate` (for a parametric cell), and `block` (for a block cell).

**Example**

```plaintext
source : user ;
```

**symmetry Simple Attribute**

Use this attribute to specify the acceptable orientation for the macro. The cell symmetry must match the associated site symmetry. When the attribute is not specified, a cell is
considered asymmetric. The allowable orientations of the cell are derived from the symmetry.

**Syntax**

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        symmetry : value_enum ;
        ...
    }
}
```

**value**

Valid values are $r$, $x$, $y$, $xy$, and $rxy$.

where

- $r$
  - Specifies symmetry in 90 degree counterclockwise rotation
- $x$
  - Specifies symmetry about the x-axis
- $y$
  - Specifies symmetry about the y-axis
- $xy$
  - Specifies symmetry about the x-axis and the y-axis
- $rxy$
  - Specifies symmetry about the x-axis and the y-axis and in 90 degree counterclockwise rotation increments

**Example**

```
symmetry : r ;
```

---

**extract_via_region_from_cont_layer Complex Attribute**

Use this attribute to extract via region information from contact layers.

**Syntax**

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        extract_via_region_from_cont_layer(cont_layer_nameid,
```
cont_layer_name_id, ... } ;
...
}
}

cont_layer_name
A list of one or more string values representing the contact layer names.

Example
extract_via_region_from_cont_layer () ;

obs_clip_box Complex Attribute
Use this attribute to specify a rectangular area of a cell layout in which connections are not allowed or not desired. The resulting rectangle becomes an obstruction. Use this attribute at the macro group level to customize the rectangle size for a cell. The values you specify at the macro group level override the values you set in the pseudo_phys_library group.

Syntax
phys_library(library_name_id) {
    macro(cell_name_id) {
        obs_clip_box( top\_float, right\_float, 
                      bottom\_float, left\_float) ;
        ...
    }
}

top, right, bottom, left
Floating-point numbers that specify the coordinates for the corners of the rectangular area.

Example
obs_clip_box(165000, 160000, 160000, 160000) ;

origin Complex Attribute
Use this attribute to specify the origin of a cell, which is the lower-left corner of the bounding box.

Syntax
phys_library(library_name_id) {
    macro(cell_name_id) {
        origin(num\_X\_float, num\_Y\_float) ;
        ...
    }
}
num_x, num_y

Floating-point numbers that specify the origin coordinates.

Example
group(origin(0.0, 0.0) ;

---

size Complex Attribute

Use this attribute to specify the size of a cell. This is the minimum bounding rectangle for the cell. Set this to a multiple of the placement grid.

Syntax

phys_library(library_name_id) {
  macro(cell_name_id) {
    size(num_xfloat, num_yfloat) ;
    ...
  }
}

num_x, num_y

Floating-point numbers that represent the cell bounding box dimension. For standard cells, the height should be equal to the associated site height and the width should be a multiple of the site width.

Example

group(size(0.9, 7.2) ;

---

foreign Group

Use this group to specify the associated GDSII structure (model) of a macro. Used for GDSII input and output to adjust the coordinate and orientation variations between GDSII and the physical library.

Note:

Only one foreign group is allowed in a macro group.

Syntax

phys_library(library_name_id) {
  macro(cell_name_id) {
    foreign(foreign_object_name_id) {
      
    }
  }
}
foreign_object_name

The name of the corresponding GDSII cell (model).

Example
foreign(and12a) {
  ...
}

Simple Attribute
orientation

Complex Attribute
origin

orientation Simple Attribute
Use this attribute to specify the orientation of the GDSII foreign cell.

Syntax
phys_library(library_nameid) {
  macro(cell_nameid) {
    foreign(foreign_object_nameString) {
      orientation : value_enum ;
      ...
    }
  }
}

value
Valid values are \texttt{N} (north), \texttt{E} (east), \texttt{S} (south), \texttt{W} (west), \texttt{FN} (flip north), \texttt{FE} (flip east), \texttt{FS} (flip south), and \texttt{FW} (flip west), as shown in Figure 7-1.
Figure 7-1  Orientation Examples

Example
orientation : N ;

origin Complex Attribute
Use this attribute to specify the equivalent coordinates of a placed macro origin in the GDSII coordinate system.

Syntax
phys_library(library_nameid) {
macro(cell_nameid) {
    foreign(foreign_object_nameid) {
        origin(xfloat, yfloat) ;
        ...
    }
    ...
}
}

x, y
Floating-point numbers that specify the GDSII coordinates where the macro origin is placed.

Example
The example shows that the macro origin (the lower-left corner) is located at (-2.0, -3.0) in the GDSII coordinate system.
origin(-2.0, -3.0) ;
**obs Group**

Use this group to specify an obstruction on a cell.

**Note:**

The *obs* group does not take a name.

**Syntax**

```plaintext
phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            ...
        }
    }
}
```

**Example**

```plaintext
obs() {
    ...
}
```

**Complex Attributes**

**via**

**via_iterate**

**Group**

**geometry**

**via Complex Attribute**

Use this attribute to specify a via instance at the given coordinates.

**Syntax**

```plaintext
phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            via(via_nameid, xfloat, yfloat );
            ...
        }
    }
}
```

**via_name**

The name of a via already defined in the *resource* group.

**x, y**

Floating-point numbers that represent the x- and y-coordinates for placement.
Example
via(via12, 0, 100) ;

via_iterate Complex Attribute
Use this attribute to specify an array of via instances in a particular pattern.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            via_iterate(num_xint, num_yint, space_xfloat, space_yfloat, via_nameid, xfloat, yfloat) ;
        }
    }
}

num_x, num_y
Integer numbers that represent the number of columns and rows in the array, respectively.

space_x, space_y
Floating-point numbers that specify the value for spacing between each via origin.

via_name
Specifies the name of a previously defined via to be instantiated.

x, y
Floating-point numbers that specify the endpoints.

Example
via_iterate(2, 2, 2.000, 3.000.0, via12, 176.0, 1417.0) ;

geometry Group
Use this group to specify the geometries of an obstruction on the specified macro.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            geometry(layer_nameid) {
                ...
            }
        }
    }
}
layer_name

Specifies the name of the layer where the obstruction is located.

Example

geometry(metal) {
    ...
}

Simple Attributes

core_blockage_margin
feedthru_area_layer
generate_core_blockage
preserve_current_layer_blockage
treat_current_layer_as_thin_wire

Complex Attributes

max_dist_to_combine_current_layer_blockage
path
path_iterate
polygon
polygon_iterate
rectangle
rectangle_iterate

core_blockage_margin Simple Attribute

Use this attribute to specify a value for computing the margin of a block core.

Syntax

phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            geometry(layer_nameid) {
                core_blockage_margin : valuefloat ;
            }
        } ...
    }
}

value

A positive floating-point number representing the margin.

Example

core_blockage_margin : 0.0 ;
**feedthru_area_layer Simple Attribute**

Use this attribute to prevent an area from being covered with a blockage and to prevent any merging from occurring within the specified area on the corresponding layer.

**Syntax**

```plaintext
phys_library(library_name_id) {
  macro(cell_name_id) {
    obs() {
      geometry(layer_name_id) {
        feedthru_area_layer : value_id ;
        ...
      }
    }
  }
}
```

**value**

A string representing the layer name.

**Example**

```plaintext
core_blockage_margin : 0.0 ;
```

**generate_core_blockage Simple Attribute**

Use this attribute to specify whether to generate the core blockage information.

**Syntax**

```plaintext
phys_library(library_name_id) {
  macro(cell_name_id) {
    obs() {
      geometry(layer_name_id) {
        generate_core_blockage : valueBoolean ;
        ...
      }
    }
  }
}
```

**value**

Valid values are TRUE and FALSE (default).

**Example**

```plaintext
generate_core_blockage : TRUE ;
```

**preserve_current_layer_blockage Simple Attribute**

Use this attribute to specify whether to preserve the current layer blockage information.
Syntax

phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            geometry(layer_nameid) {
                preserve_current_layer_blockage : valueBoolean ;
                ...
            }
        }
    }
}

value

Valid values are TRUE and FALSE (default).

Example

preserve_current_layer_blockage : TRUE ;

treat_current_layer_as_thin_wires Simple Attribute

Use this attribute to specify whether to treat the current layer as thin wires.

Syntax

phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            geometry(layer_nameid) {
                treat_current_layer_as_thin_wires : valueBoolean ;
                ...
            }
        }
    }
}

value

Valid values are TRUE and FALSE (default).

Example

treat_current_layer_as_thin_wires : TRUE ;

max_dist_to_combine_current_layer_blockage Complex Attribute

Use this attribute to specify a maximum distance value, beyond which blockages on the current layer are not combined.

Syntax

phys_library(library_nameid) {
    macro(cell_nameid) {
    ...
obs() {
  geometry(layer_nameid) {
    max_dist_to_combine_current_layer_blockage
    ( valuefloat, valuefloat ) ;
    ...
  }
}
}

value

Floating-point numbers that represent the maximum distance value.

Example

max_dist_to_combine_current_layer_blockage ( ) ;

path Complex Attribute

Use this attribute to specify a shape by connecting specified points. The drawn geometry is extended on both endpoints by half the wire width.

Syntax

phys_library(library_nameid) {
  macro(cell_nameid) {
    obs() {
      geometry(layer_nameid) {
        path(widthfloat, x1float, y1float, ..., ...,
        xnfloat, ynfloat) ;
        ...
      }
    }
  }
}

width

Floating-point number that represents the width of the path shape.

x1,y1, ..., xn,yn

Floating-point numbers that represent the x- and y-coordinates for each point that defines a trace. The path shape is extended from the trace outward by one half the width on both sides. If only one point is specified, a square centered on that point is generated. The width of the generated square equals the width value.

Example

path(2.0,1,1,1,4,10,4,10,8) ;
path_iterate Complex Attribute

Represents an array of paths in a particular pattern.

Syntax

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            geometry(layer_nameid) {
                path_iterate(num_xint, num_yint,
                space_xfloat, space_yfloat,
                widthfloat, x1float, y1float,...,
                xnfloat, ynfloat)
            }
        }
    }
}
```

**num_x, num_y**

Integer numbers that represent the number of columns and rows in the array, respectively.

**space_x, space_y**

Specify the value for spacing around the path.

**width**

Floating-point number that represents the width of the path shape.

**x1, y1**

Floating-point numbers that represent the first path point.

**xn, yn**

Floating-point numbers that represent the final path point.

**Example**

```
path_iterate(2,1,5.000,5.000,2.0,1,1,1,4,10,4,10,8) ;
```

polygon Complex Attribute

Represents a rectilinear polygon by connecting all the specified points.

Syntax

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
```

geometry(layer_nameid) {
    polygon(x1, y1, ..., xn, yn) ;
    ...
}
}

x1,y1,...,xn,yn

Floating-point numbers that represent the x- and y-coordinates for each point that defines the shape. Specify a minimum of four points.

You are responsible for ensuring that the resulting polygon is orthogonal.

Example
polygon(175.500, 1414.360, 176.500, 1414.360, 176.500, 1417.360, 175.500, 1417.360) ;

class polygon_iterate Complex Attribute

Represents an array of rectilinear polygons in a particular pattern.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        obs() {
            geometry(layer_nameid) {
                polygon_iterate (num_xint, num_yint,
                                space_xfloat, space_yfloat,
                                x1float, y1float, x2float,
                                y2float, x3float, y3float, ..., 
                                xnfloat, yfloat) ;
                ...
            }
        }
    }
}

num_x, num_y

Integer numbers that represent the number of columns and rows in the array, respectively.

space_x, space_y

Floating-point numbers that specify the value for spacing around the polygon.

x1, y1; x2, y2; x3, y3; ..., ...; xn, yn

Floating-point numbers that represent successive points of the polygon.
Note:
You must specify at least four points.

Example

polygon_iterate(2, 2, 2.000, 4.000, 175.500, 1414.360, 176.500, 1414.360, 176.500, 1417.360, 175.500, 1417.360);
**num_x, num_y**

Integer numbers that represent the number of columns and rows in the array, respectively.

**space_x, space_y**

Floating-point numbers that specify the value for spacing around the rectangles.

**x1, y1; x2, y2**

Floating-point numbers that specify the coordinates for the diagonally opposite corners of the rectangles.

**Example**

```
rectangle_iterate(2, 2, 2.000, 4.000, 175.500, 1417.360, 176.500, 1419.140) ;
```  

---

**site_array Group**

Use this group to specify the site array associated with a cell. The site class and site symmetry must match the cell class and cell symmetry.

**Note:** You can use this attribute only with gate array libraries.

**Syntax**

```
phys_library(library_nameId) {
  macro(cell_nameId) {
    site_array(site_nameId) {
      ...
    }
  }
}
```

**site_name**

The name of a site already defined in the `resource` group.

**Example**

```
site_array(core) {
  ...
}
```

**Simple Attribute**

**orientation**

**Complex Attributes**

**iterate**
origin

**orientation Simple Attribute**

Use this attribute to specify how you place the cells in an array.

**Syntax**

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        site_array(site_nameid) {
            orientation : valueenum;
            ...
        }
    }
}
```

**value**

Valid values are **N** (north), **E** (east), **S** (south), **W** (west), **FN** (flip north), **FE** (flip east), **FS** (flip south), and **FW** (flip west), as shown in Figure 7-2.

*Figure 7-2 Orientation Examples*

```
N (north)  E (east)  S (south)  W (west)
FN (flip north)  FE (flip east)  FS (flip south)  FW (flip west)
```

**Example**

```
orientation : N;
```

**iterate Complex Attribute**

Use this attribute to specify the dimensions and arrangement of an array of sites.
Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        site_array(site_nameid) {
            iterate(num_x_int, num_y_int, space_x_int, space_y_int) ;
            ...
        }
    }
}

num_x, num_y
Integer numbers that represent the number of rows and columns in an array, respectively.

space_x, space_y
Floating-point numbers that represent the row and column spacing, respectively.

Example
iterate(17, 1, 0.98, 11.76) ;

origin Complex Attribute
Use this attribute to specify the origin of a site array.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        site_array(site_nameid) {
            origin(x_float, y_float) ;
            ...
        }
    }
}

x, y
Floating-point numbers that specify the origin coordinates of the site array.

Example
origin(0.0, 0.0) ;
Specifying Attributes and Groups in the pin Group

For each cell, you use the macro group to specify the macro-level information and pin information. Macro-level information includes such properties as symmetry, size and obstruction. Pin information includes such properties as geometry and position.

This chapter describes the attributes and groups that you define in the pin group within the macro group.
pin Group

Use this group to specify one pin in a cell group.

Syntax

\[
\text{phys_library}(\text{library}_id) \{ \\
\quad \text{macro}(\text{cell}_id) \{ \\
\quad \quad \text{pin}(\text{pin}_id) \{ \\
\quad \quad \quad \ldots \\
\quad \quad \} \\
\quad \} \\
\} \\
\]

\text{pin}_id

Specifies the name of the pin. This name must be identical to the name of the logical \text{pin}_id that you define in the library.

Example

\[
\text{pin}(A) \{ \\
\quad \ldots \\
\quad \text{pin description} \\
\quad \ldots \\
\} \\
\]

Simple Attributes

capacitance
direction
eq_pin
must_join
pin_shape
pin_type

Complex Attributes

antenna_contact_accum_area
antenna_contact_accum_side_area
antenna_contact_area
antenna_contact_area_partial_ratio
antenna_contact_side_area
antenna_contact_side_area_partial_ratio
antenna_diffusion_area
antenna_gate_area
antenna_metal_accum_area
antenna_metal_accum_side_area
antenna_metal_accum_side_area_partial_ratio
antenna_metal_area
antenna_metal_area_partial_ratio
Groups
foreign
port

capacitance Simple Attribute
Use this attribute to specify the capacitance value for a pin.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            capacitance : value_float ;
            ...
        }
    }
}

value
A floating-point number representing the capacitance value.

Example
capacitance : 1.0 ;

direction Simple Attribute
Use this attribute to specify the direction of a pin.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            ...
            direction : value_enum ;
            ...
        }
    }
}

value
Valid values are inout, input, feedthru, output, and tristate.

Example
direction : inout ;
eq_pin Simple Attribute

Use this attribute to specify an electrically equivalent pin.

Syntax

```liberty
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            eq_pin : pin_name_id ;
            ...
        }
    }
}
```

`pin_name`

The name of an electrically equivalent pin.

Example

```liberty
eq_pin : A ;
```

must_join Simple Attribute

Use this attribute to specify the name of a pin that must be connected to the pin_group pin.

Syntax

```liberty
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            must_join : pin_name_id ;
            ...
        }
    }
}
```

`pin_name`

The name of the pin that must be connected to the pin_group pin.

Example

```liberty
must_join : A ;
```
### pin_shape Simple Attribute

Use this attribute to specify the pin shape.

**Syntax**

```plaintext
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...  
            pin_shape : value_enum ;
            ...
        }
    }
}

value

Valid values are ring, abutment, and feedthru.

**Example**

```plaintext
pin_shape : ring ;
```
antenna_contact_accum_area Complex Attribute

Use this attribute to specify the cumulative contact area.

Syntax

physical_library(library_nameid) {
  macro(cell_nameid) {
    pin(pin_nameid) {
      ...
      antenna_contact_accum_area (valuefloat);
      ...
    }
  }
}

definition

A floating-point number that represents the antenna.

Example

antenna_contact_accum_area ( 0.0 ) ;

antenna_contact_accum_side_area Complex Attribute

Use this attribute to specify the cumulative side area.

Syntax

physical_library(library_nameid) {
  macro(cell_nameid) {
    pin(pin_nameid) {
      ...
      antenna_contact_accum_side_area (valuefloat);
      ...
    }
  }
}

definition

A floating-point number that represents the antenna.

Example

antenna_contact_accum_side_area ( 0.0 ) ;
antenna_contact_area Complex Attribute

Use this pin-specific attribute and the following attributes to specify contributions coming from intracell geometries: antenna_contact_area, antenna_contact_length, total_antenna_contact_length. These attributes together account for all the geometries, including the ports of pins that appear in the cell’s physical model.

For black box cells, use this pin-specific attribute along with antenna_contact_length and antenna_contact_perimeter to specify the amount of metal connected to a block pin on a given layer.

Syntax

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            ...
            antenna_contact_area (valuefloat);
            ...
        }
    }
}
```

value

A floating-point number that represents the contributions coming from intracell geometries.

Example

```
antenna_contact_area (0.3648, 0,0,0,0,0);
```

antenna_contact_area_partial_ratio Complex Attribute

Use this attribute to specify the antenna ratio of a contact.

Syntax

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            ...
            antenna_contact_area_partial_ratio (valuefloat);
            ...
        }
    }
}
```

value

A floating-point number that represents the ratio.
**Example**

```
antenna_contact_area_partial_ratio ( 0.0 ) ;
```

---

**antenna_contact_side_area Complex Attribute**

Use this attribute to specify the side wall area of a contact.

**Syntax**

```
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            antenna_contact_side_area (value_float );
            ...
        }
    }
}
```

**value**

A floating-point number that represents the ratio.

**Example**

```
antenna_contact_side_area ( 0.0 ) ;
```

---

**antenna_contact_side_area_partial_ratio Complex Attribute**

Use this attribute to specify the antenna ratio using the side wall area of a contact.

**Syntax**

```
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            antenna_contact_side_area_partial_ratio (value_float);
            ...
        }
    }
}
```

**value**

A floating-point number that represents the ratio.
Example
antenna_contact_side_area_partial_ratio ( 0.0 ) ;

antenna_diffusion_area Complex Attribute
For black box cells, use this attribute to specify the total diffusion area connected to a block’s pin using layers less than or equal to the pin’s layer.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            ...
            antenna_diffusion_area (value_float, value_float
            value_float ...) ;
            ...
        }
    }
}

value
Floating-point numbers representing the total diffusion area.

Example
antenna_diffusion_area (0.0, 0.0, 0.0, ...);

antenna_gate_area Complex Attribute
For black box cells, use this attribute to specify the total gate area connected to a block’s pin using layers less than or equal to the pin’s layer.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            ...;
            antenna_gate_area (value_float, value_float
            value_float ...) ;
            ...
        }
    }
}

value, value, value, ...
Floating-point numbers that represent the total gate area.
Example
antenna_gate_area (0.0, 0.0, 0.0, ...) ;

antenna_metal_accum_area Complex Attribute
Use this attribute to specify the cumulative metal area.

Syntax
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            antenna_metal_accum_area (value_float);
            ...
        }
    }
}

value
A floating-point number that represents the antenna.

Example
antenna_metal_accum_area () ;

antenna_metal_accum_side_area Complex Attribute
Use this attribute to specify the cumulative side area.

Syntax
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            antenna_metal_accum_side_area (value_float);
            ...
        }
    }
}

value
A floating-point number that represents the antenna.

Example
antenna_metal_accum_side_area () ;
antenna_metal_area Complex Attribute

Use this pin-specific attribute and antenna_metal_area to specify contributions coming from intracell geometries. These attributes together account for all the geometries, including the ports of pins that appear in the cell’s physical model.

Syntax

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            ...
            antenna_metal_area (valuefloat );
            ...
        }
    }
}
```

value

A floating-point number that represents the contributions coming from intracell geometries.

Example

```
antenna_metal_area (0.3648, 0,0,0,0,0) ;
```

antenna_metal_area_partial_ratio Complex Attribute

Use this attribute to specify the antenna ratio of a metal wire.

Syntax

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            ...
            antenna_metal_area_partial_ratio (valuefloat );
            ...
        }
    }
}
```

value

A floating-point number that represents the ratio.

Example

```
antenna_metal_area_partial_ratio () ;
```
antenna_metal_side_area Complex Attribute

Use this attribute to specify the side wall area of a metal wire.

Syntax

```
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            antenna_metal_side_area (value_float);
            ...
        }
    }
}
```

value

A floating-point number that represents the ratio.

Example

```
antenna_metal_side_area () ;
```

antenna_metal_side_area_partial_ratio Complex Attribute

Use this attribute to specify the antenna ratio using the side wall area of a metal wire.

Syntax

```
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            antenna_metal_side_area_partial_ratio (value_float);
            ...
        }
    }
}
```

value

A floating-point number that represents the ratio.

Example

```
antenna_metal_side_area_partial_ratio () ;
```
**foreign Group**

Use this group to specify which GDSII structure (model) to use when an instance of a pin is placed. Only one `foreign` group is allowed in a library.

**Syntax**

```liberty
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            foreign(foreign_object_name_id) {
                ...
            }
        }
    }
}
```

*foreign_object_name*

The name of the GDSII structure (model).

**Example**

```liberty
foreign(via34) {
    ...
}
```

**Simple Attribute**

*orientation*

**Complex Attribute**

*origin*

**orientation Simple Attribute**

Use this attribute to specify how you place the cells in an array in relation to the VDD and VSS buses.

**Syntax**

```liberty
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            ...
            foreign(foreign_object_name_id) {
                orientation : value_enum ;
                ...
            }
        }
    }
}
```
value

Valid values are N (north), E (east), S (south), W (west), FN (flip north), FE (flip east), FS (flip south), and FW (flip west), as shown in Figure 8-1.

Figure 8-1 Orientation Examples

Example
orientation : N ;

origin Complex Attribute

Use this attribute to specify the equivalent coordinates of a placed foreign origin.

Syntax
phys_library(library_nameid) {
macro(cell_nameid) {
  pin(pin_nameid) {
    ...
    foreign(foreign_object_nameid) {
      ...
      origin(xfloat, yfloat) ;
    }
  }
}
}
x,y

Floating-point numbers that specify the coordinates of the foreign object's origin.
Example
origin(-1, -1) ;

---

**port Group**

Use this group to specify the port geometries for a pin.

**Syntax**

```
phys_library(library_nameid) {
  macro(cell_nameid) {
    pin(pin_nameid) {
      port() {
        ...
      }
    }
  }
}
```

*Note:* The *port* group does not take a name.

**Example**

```
port() {
  ...
}
```

**Complex Attributes**

- **via**
- **via_iterate**

**Group**

**geometry**

**via Complex Attribute**

Use this attribute to instantiate a via relative to the origin implied by the coordinates (typically the center).

```
phys_library(library_nameid) {
  macro(cell_nameid) {
    pin(pin_nameid) {
      port() {
        via(via_nameid, x, y) ;
        ...
      }
    }
  }
}
```
**via_name**

A previously defined via.

**x**

The horizontal coordinate.

**y**

The vertical coordinate.

**Example**

```
via(via23, 25.00, -30.00);
```

### via_iterate Complex Attribute

Use this attribute to instantiate an array of vias in a particular pattern.

**Syntax**

```
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            port() {
                ...
                via_iterate(num_xint, num_yint,
                   space_xfloat, space_yfloat,
                   via_nameid, xfloat, yfloat) ;
                ...
            }
        }
    }
}
```

**num_x, num_y**

Integer numbers that represent the number of columns and rows in the array, respectively.

**space_x, space_y**

Floating-point numbers that specify the value for spacing around each via.

**via_name**

Specifies the name of a previously defined via.

**x, y**

Floating-point numbers that specify the location of the first via.
Example
via_iterate(2, 2, 100, 100, via12, 0, 0);

geometry Group
Use this group to specify the geometry of an obstruction or a port.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            port() {
                ... geometry(layer_nameid) {
                }
            }
        }
    }
}

layer_name
The layer where the shape is defined.

Example
gometry(cut01) {
    ...
}

Complex Attributes
path
path_iterate	polygon
polygon_iterate
trectangle
rectangle_iterate

path Complex Attribute
Use this attribute to specify a shape by connecting specified points. The drawn geometry is extended by half the default wire width of the layer on both endpoints.

Syntax
phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            port() {
                geometry(layer_nameid) {
                    path(widthfloat, x1float, y1float, ..., ..., xnfloat, ynfloat)
                }
            }
        }
    }
}
width

Floating-point number that represents the width of the path shape.

\( x_1, y_1; \ldots; x_n, y_n \)

Floating-point numbers that represent the x- and y-coordinates for each point that defines a trace. The path shape is extended from the trace by one half of the width on both sides. If only one point is specified, a square centered on that point is generated. The width of the generated square equals the width value.

**Example**

```
path(1,1,4,4,10,10,5,10) ;
```

**path_iterate Complex Attribute**

Use this attribute to specify an array of paths in a particular pattern.

**Syntax**

```
phys_library{library_nameid} {
    macro{cell_nameid} {
        pin{pin_nameid} {
            port() {
                geometry{layer_nameid} {
                    ...
                    path_iterate(num_xint, num_yint, space_xfloat, space_yfloat, widthfloat, x1float, y1float, \ldots, xnfloat, ynfloat)
                    ...
                }
            }
        }
    }
}
```

**num_x, num_y**

Integer numbers that, respectively, represent the number of columns and rows in the array.

**space_x, space_y**

Floating-point numbers that specify the value for spacing around the path.
width

Floating-point number that represents the width of the path shape.

x1, y1

Floating-point numbers that represent the first path point.

xn, yn

Floating-point numbers that represent the final path point.

Example

```
path_iterate(2, 1, 5.000, 5.000, 1.000, 174.500, 1419.140, 177.500, 1422.140) ;
```

**polygon Complex Attribute**

Use this attribute to specify a rectilinear polygon by connecting all the specified points.

**Syntax**

```
phys_library(library_name_id) {
    macro(cell_name_id) {
        pin(pin_name_id) {
            port() {
                geometry(layer_name_id) {
                    ...
                    polygon(x1float, y1float; ..., ..., 
                        xnfloat, ynfloat)
                    ...
                }
            }
        }
    }
}
```

x1, y1; ..., ....; xn, yn

Floating-point numbers that represent the x- and y-coordinates for each point that defines the shape. You should specify a minimum of four points.

**Note:**

You are responsible for ensuring that the resulting polygon is rectilinear.

**Example**

```
polygon(175.500, 1414.360, 176.500, 1414.360, 176.500, 1417.360, 175.500, 1417.360) ;
```

**polygon_iterate Complex Attribute**

Use this attribute to specify an array of polygons in a particular pattern.
Syntax

phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            port() {
                geometry(layer_nameid) {
                    ...
                    polygon_iterate(num_xint, num_yint,
                        space_xfloat, space_yfloat,
                        x1float, y1float, x2float,
                        y2float, x3float, y3float, ...,
                        xnfloat, ynfloa
                    )
                    ...
                }
            }
        }
    }
}

num_x, num_y

Integer numbers that represent the number of columns and rows in the array, respectively.

space_x, space_y

Floating-point numbers that specify the value for spacing around the polygon.

x1, y1; x2, y2; x3, y3; ..., ...; xn, yn

Floating-point numbers that represent successive points of the polygon.

Note:
You must specify at least four points.

Example

polygon_iterate(2, 2, 2.000, 4.000, 175.500, 1414.360,
                176.500, 1414.360, 176.500, 1417.360,
                175.500, 1417.360) ;

rectangle Complex Attribute

Use this attribute to specify a rectangular shape.

Syntax

phys_library(library_nameid) {
    macro(cell_nameid) {
        pin(pin_nameid) {
            port() {
                geometry(layer_nameid) {
                    ...
                    rectangle(x1float, y1float, x2float, y2float)
                }
            }
        }
    }
}
Floating-point number that specify the coordinates for the diagonally opposing corners of the rectangle.

**Example**

```plaintext
rectangle(2, 0, 4, 0);
```

**rectangle_iterate Complex Attribute**

Use this attribute to specify an array of rectangles in a particular pattern.

**Syntax**

```plaintext
phys_library{library_nameid} {
    macro{cell_nameid} {
        pin{pin_nameid} {
            port() {
                geometry{layer_nameid} {
                    ...
                    rectangle_iterate(num_xint, num_yint,
                        space_xfloat, space_yfloat,
                        x1float, y1float, x2float, y2float)
                    ...
                }
            }
        }
    }
}
```

**num_x, num_y**

Integer numbers that represent the number of columns and rows in the array, respectively.

**space_x, space_y**

Floating-point numbers that specify the value for spacing around the rectangles.

**x1, y1; x2, y2**

Floating-point numbers that specify the coordinates for the diagonally opposite corners of the rectangles.

**Example**

```plaintext
rectangle_iterate(2, 2, 2.000, 4.000, 175.5, 1417.360,
```
Developing a Physical Library

The physical library specifies the information required for floor planning, RC estimation and extraction, placement, and routing.

You use the physical library syntax (.plib) to model your physical library.

This chapter includes the following sections:

- Creating the Physical Library
- Naming the Source File
- Naming the Physical Library
- Defining the Units of Measure
Creating the Physical Library

This section describes how to name your source file and library, and how to define the units of measure for properties in your library.

Naming the Source File

The recommended file name suffix for physical library source files is .plib.

Example

myLibplib

Naming the Physical Library

You specify the name for your physical library in the phys_library group, which is always the first executable line in a library source file.

Syntax

\[
\text{phys_library} (\text{library\_name_id}) \{
\ldots
\}
\]

Use the comment, date, and revision attributes to document your library source file.

Example

\[
\text{phys_library} (\text{sample}) \{
\quad \text{comment} : \text{"my library"} ;
\quad \text{date} : \text{"1st Jan 2002"} ;
\quad \text{revision} : \text{"Revision 2.0.5"} ;
\}
\]

Defining the Units of Measure

Use the phys_library group attributes described in Table 9-1 to specify the units of measure for properties such as capacitance and resistance. The unit statements must precede other definitions, such as the technology data, design rules, and macros.

Syntax

\[
\text{phys_library} (\text{library\_name_id}) \{
\quad \ldots
\quad \text{attribute\_name} : \text{value\_enum} ;
\quad \ldots
\}
\]
Example

```plaintext
phys_library(sample) {
    capacitance_unit : 1pf ;
    distance_unit : 1um ;
    resistance_unit : 1ohm ;
    ...
}
```

Table 9-1 lists the attribute names and values that you can use to define the units of measure.

<table>
<thead>
<tr>
<th>Property</th>
<th>Attribute name</th>
<th>Legal values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacitance</td>
<td>capacitance_unit</td>
<td>1pf, 1ff, 10ff, 100ff</td>
</tr>
<tr>
<td>Distance</td>
<td>distance_unit</td>
<td>1um, 1mm</td>
</tr>
<tr>
<td>Resistance</td>
<td>resistance_unit</td>
<td>1ohm, 100ohm, 10ohm, 1kohm</td>
</tr>
<tr>
<td>Time</td>
<td>time_unit</td>
<td>1ns, 100ps, 10ps, 1ps</td>
</tr>
<tr>
<td>Voltage</td>
<td>voltage_unit</td>
<td>1mV, 10mV, 100mV, 1V</td>
</tr>
<tr>
<td>Current</td>
<td>current_unit</td>
<td>100uA, 100mA, 1A, 1uA, 10uA,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1mA, 10mA</td>
</tr>
<tr>
<td>Power</td>
<td>power_unit</td>
<td>1mw</td>
</tr>
<tr>
<td>Database</td>
<td>dist_conversion_factor</td>
<td>Any multiple of 100</td>
</tr>
<tr>
<td>distance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resolution</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The physical library specifies the information required for floor planning, RC estimation and extraction, placement, and routing.

You use the physical library syntax (.plib) to model your physical library.

This chapter includes the following sections:

- Defining the Technology Data
- Defining the Architecture
- Defining the Layers
- Defining Vias
- Defining the Placement Sites
Defining the Technology Data

Technology data includes the process and electrical design parameters. Site-array and cell data refer to the technology data; therefore, you must define the layer data before you define site-array and cell data.

Defining the Architecture

You specify the architecture and the layer information in the resource group inside the phys_library group.

Syntax

```
phys_library(library_nameid) {
  resource(architecture enum) {
    ...
  }
}
```

architecture

The valid values are std_cell and array.

Example

```
phys_library(mylib) {
  ...
  resource(std_cell) {
    ...
  }
}
```

Defining the Layers

The layer definition is order dependent. You define the layers starting with the layer closest to the substrate and ending with the layer furthest from the substrate.

Depending on their purpose, the layers can include

- Contact layer
- Overlap layer
- Routing layer
- Device layer
Contact Layer

Contact layers define the contact cuts that enable current to flow between the device and the first routing layer or between any two routing layers; for example, cut01 between poly and metal1, or cut12 between metal1 and metal2. You define the contact layer by using the contact_layer attribute inside the resource group.

Syntax
```resource(architecture { 
    contact_layer(layer_name)
...
})
```

Example
```contact_layer(cut01) ;
```

Overlap Layer

An overlap layer provides accurate overlap checking of rectilinear blocks. You define the overlap layer by using the overlap_layer attribute inside the resource group.

Syntax
```resource(architecture { 
    overlap_layer(layer_name)
...
})
```

Example
```resource(std_cell) { 
    overlap_layer(mod) ;
...
}
```

Routing Layer

You define the routing layer and its properties by using the routing_layer group inside the resource group.

Syntax
```resource(architecture { 
    routing_layer(layer_name) { 
        attribute : value ;
    }...
})
```
Example

```liberty
resource(std_cell) ; {
    routing_layer(m1) { /* metall layer definition */
        cap_per_sq : 3.200e-04 ;
        default_routing_width : 3.200e-01 ;
        res_per_sq : 7.000e-02 ;
        routing_direction : horizontal ;
        pitch : 9.000e-01;
        spacing : 3.200e-01 ;
        cap_multiplier : ;
        shrinkage : ;
        thickness : ;
    }
}
```

Table 10-1 lists the attributes you can use to specify routing layer properties.

Note:

All numerical values are floating-point numbers.

**Table 10-1  Routing Layer Simple Attributes**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Valid values</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_routing_width</td>
<td>&gt; 0.0</td>
<td>Minimum metal width allowed on the layer; the default width for regular wiring</td>
</tr>
<tr>
<td>cap_per_sq</td>
<td>&gt; 0.0</td>
<td>Capacitance per square unit between a layer and a substrate, used to model wire-to-ground capacitance</td>
</tr>
<tr>
<td>res_per_sq</td>
<td>&gt; 0.0</td>
<td>Resistance per square unit</td>
</tr>
<tr>
<td>coupling_cap</td>
<td>&gt; 0.0</td>
<td>Coupling capacitance between parallel wires on the same layer</td>
</tr>
<tr>
<td>fringe_cap</td>
<td>&gt; 0.0</td>
<td>Fringe (sidewall) capacitance per unit length of a routing layer</td>
</tr>
<tr>
<td>routing_direction</td>
<td>horizontal, vertical</td>
<td>Preferred routing direction</td>
</tr>
<tr>
<td>pitch</td>
<td>&gt; 0.0</td>
<td>Routing pitch</td>
</tr>
<tr>
<td>spacing</td>
<td>&gt; 0.0</td>
<td>Default different net spacing (edge-edge) for regular wiring on a layer</td>
</tr>
<tr>
<td>cap_multiplier</td>
<td>&gt; 0.0</td>
<td>Cap multiplier; accounts for changes in capacitance due to nearby wires</td>
</tr>
</tbody>
</table>
Specifying Net Spacing

Use the ranged_spacing complex attribute to specify the different net spacing for special wiring on the layer. You can also use this attribute to specify the minimum spacing for a particular routing width range of the metal. You can use more than one ranged_spacing attribute to specify spacing rules for different ranges.

Each ranged_spacing attribute requires floating-point values for the minimum width for the wiring range, the maximum width for the wiring range, and the minimum spacing for the net.

Syntax

```
resource(architecture enum) {
    routing_layer(layer_name id) {
        ...
        ranged_spacing(value float, value float, value float) ;
        ...
    }
}
```

Example

```
routing_layer(m1) {
    ...
    ranged_spacing(1.60, 2.40, 1.20) ;
    ...
}
```
Device Layer

Device layers make up the transistors below the routing layers—for example, the poly layer and the active layer. To define the device layer, use the `device_layer` attribute inside the `resource` group.

Wires are not allowed on device layers. If pins appear in the device layer, you must define vias to permit the router to connect the pins to the first routing layer.

Syntax

```
resource(architecture enum) {
    device_layer(layer_name_id) ;
    ...
}
```

Example

```
resource(std_cell) {
    device_layer(poly) ;
    ...
}
```

Defining Vias

A via is the routing connection for wires in each pair of connected layers. Vias typically comprise three layers: the two connected layers and the cut layer between the connected layers.

Naming the Via

You define the via name in the `via` group inside the `resource` group.

Syntax

```
resource(architecture enum) {
    via(via_name_id) {
        ...
    }
}
```

Example

```
resource(std_cell) {
    ...
    via(via23) {
        ...
    }
    ...
}
```
Defining the Via Properties

You define the via properties by using the following attributes inside the via group.

- `is_default`
- `top_of_stack_only`
- `resistance`

Syntax

```plaintext
via(via_name_id) {
    is_default : Boolean ;
    top_of_the_stack : Boolean ;
    resistance : float ;
    ...
}
```

Example

```plaintext
via(via23) {
    is_default : TRUE;
    top_of_stack_only : FALSE;
    resistance : 1.0;
    ...
}
```

Table 10-2 lists the properties you can define with the via attributes.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Valid values</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>is_default</code></td>
<td>TRUE, FALSE</td>
<td>Default via for a given layer pair</td>
</tr>
<tr>
<td><code>top_of_stack_only</code></td>
<td>TRUE, FALSE</td>
<td>Use only on top of a via stack</td>
</tr>
<tr>
<td><code>resistance</code></td>
<td>floating-point number</td>
<td>Resistance per contact-cut rectangle</td>
</tr>
</tbody>
</table>

Defining the Geometry for Simple Vias

Define the via geometry (or geometries) by using `via_layer` groups inside a `via` group. Each `via_layer` group defines the via geometry for one layer. Use the name of the layer as the `via_layer` group name.

The `layer1` and `layer2` layers are the adjacent routing layers, where `layer1` is closer to the substrate. The contact layer is the cut layer between `layer1` and `layer2`.

For rectilinear vias, you define the geometry by using more than one rectangle function for the corresponding layer.
Syntax

\begin{verbatim}
via_layer(layer1_nameid) {
    rectangle(x11, y11, x21, y21) ; /* 1 or more rectangles */
}
via_layer(contact_nameid) {
    rectangle(x1c, y1c, x2c, y2c) ; /* 1 or more rectangles */
}
via_layer(layer2_nameid) {
    rectangle(x12, y12, x22, y22) ; /* 1 or more rectangles */
}

where (x11, y11), (x21, y21), (x1c, y1c), (x2c, y2c), (x21, y12), and (x22, y22) are the coordinates of the opposite corners of the rectangle.
\end{verbatim}

Example

\begin{verbatim}
via(via 45) {
    is_default : TRUE ;
    resistance : 1.5 ;
    via_layer(metal4) {
        rectangle(-0.3, -0.3, 0.3, 0.3) ;
    }
    via_layer(cut45) {
        rectangle(-0.18, -0.18, 0.18, 0.18) ;
    }
    via_layer(meta15) {
        rectangle(-0.27, -0.27, 0.27, 0.27) ;
    }
}
\end{verbatim}

Defining the Geometry for Special Vias

Special vias are vias that have

- Fewer than three layers, with one layer being a contact layer
- More than three layers

Vias With Fewer Than Three Layers

To define vias that have fewer than three layers, use the `via_layer` group, as shown below.

Syntax

\begin{verbatim}
via_layer(via_nameid) {
    rectangle(x1, y1, x2, y2) ;
}
\end{verbatim}
where \((x1, y1)\) and \((x2, y2)\) are the coordinates (floating-point numbers) for the opposite corners of the rectangle, as shown in Figure 10-1.

**Figure 10-1  Coordinates of a Rectangle**

![Diagram of a rectangle with coordinates \((x1, y1)\) and \((x2, y2)\)](image)

**Example**

```plaintext
via_layer(cut23) {
    rectangle(-0.18, -0.18, 0.18, 0.18) ;
}
```

**Vias With More Than Three Layers**

For vias with more than three layers, use multiple `via_layer` groups. You can have more than one `via_layer` group in your physical library.

**Syntax**

```plaintext
via_layer (via_name_id) {
    rectangle(x1_float, y1_float, x2_float, y2_float) ;
}
```

where \((x1, y1)\) and \((x2, y2)\) are the coordinates (floating-point numbers) for the opposite corners of the rectangle.

**Example**

```plaintext
via(via123) {
    resistance : 1.5 ;
    via_layer(met1) {
        rectangle(-0.3, -0.3, 0.3, 0.3):
    }
    via_layer(cut12) {
        rectangle(-0.2, -0.2, 0.2, 0.2):
    }
    via_layer(met2) {
        rectangle(-0.3, -0.3, 0.3, 0.3):
    }
    via_layer(met23) {
        rectangle(-0.2, -0.2, 0.2, 0.2):
    }
}
```
Referencing a Foreign Structure

Use the `foreign` group to specify which GDSII structure (model) to use when you place an instance of the via. You also use this group to specify the orientation and the offset with respect to the GDSII structure origin.

Note:
Only one foreign reference is allowed for each via.

Syntax

```plaintext
foreign(foreign_structure_name) {
  orientation : N | E | W | S | FN | FE | FW | FS ;
  origin(X_{float}, Y_{float}) ;
}
```

where \( x \) and \( y \) represent the offset distance.

Example

```plaintext
via(via34) {
  is_default : TRUE ;
  resistance : 2.0e-02 ;
  foreign(via34) {
    orientation : FN ;
    origin(-1, -1) ;
  }
  ...
}
```

Defining the Placement Sites

For each class of cells (such as cores and pads), you must define the available sites for placement. The methodology you use for defining placement sites depends on whether you are working with standard cell technology or gate array technology.

Standard Cell Technology

For standard cell technologies you define the placement sites by defining the site name in the `site` group inside the `resource` group, and by defining the site properties using the following attributes inside the `site` group:

- The `site_class` attribute specifies the site class. Two types of placement sites are supported:
- Core (core cell placement)
- Pad (I/O placement)

- The `symmetry` attribute specifies the site symmetry with respect to the x- and y-axes.
  
  **Note:**
  
  If you do not specify the `symmetry` attribute, the site is considered asymmetric.

- The `size` attribute specifies the site size.

**Syntax**

```
resource(architecture_type) {
  site(site_name) {
    site_class : core | pad ;
    symmetry : x | y | r | xy | rxy ;
    size(x_size, y_size) ;
  }
}
```

**site_name**

The name of the library site. Common practice is to describe the function of the site (core or pad) with the site name.

You can assign one of the following values to the `symmetry` attribute:

- `x`
  Specifies symmetry about the x-axis

- `y`
  Specifies symmetry about the y-axis

- `r`
  Specifies symmetry in 90 degree counterclockwise rotation

- `xy`
  Specifies symmetry about the x-axis and the y-axis

- `rxy`
  Specifies symmetry about the x-axis and the y-axis and in 90 degree counterclockwise rotation increments

*Figure 10-2* shows the relationship of the symmetry values to the axis.
Gate Array Technology

Follow these guidelines when working with gate array technologies:

- Define the basic sites for the core and pad in the same way you would for standard cell technologies.

- Use the array group to define arrays for the site, the floorplan, and the detail routing grid descriptions. You define the array group inside the resource group.

Defining the Floorplan Set

A floorplan is an array of sites that allow or disallow the placement of cells. You define a floorplan group or multiple floorplan groups inside an array group.

A floorplan without a name becomes the default floorplan. Subsequently, when no floorplan is specified, the default floorplan is used. Figure 10-3 shows the elements of a floorplan on a die.
Instantiating the Site Array

You instantiate arrays by using the site_array group inside the floorplan group. The orientation, availability for placement, origin, and the array pattern (that is, the number of rows and columns, as well as the row spacing and column spacing) are all defined in the site_array group.

Syntax

```plaintext
site(site_name_id) {
    stateless : pad | core;
    symmetry : x | y | r | xy | rxy;
    size(x_size_float, y_size_float);
}
array(array_name_id) {
    ...
floorplan(floorplan_name_id) {
    site_array(site_name_id) {
        orientation : N | E | W | S | FN | FE | FW | FS;
        placement_rule : regular | can_place | cannot_place;
        origin(X_float, Y_float);
        iterate(num_X_int, num_Y_int,
            space_X_float, space_Y_float);
    }
}
}
```

Figure 10-3  Elements of a Floorplan
Table 10-3 shows the values and description for each of the attributes you use to define placement sites.

**Table 10-3 Placement Site Definitions**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Valid values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>site_class</td>
<td>pad</td>
<td>I/O cell placement site</td>
</tr>
<tr>
<td></td>
<td>core</td>
<td>Core cell placement site</td>
</tr>
<tr>
<td>symmetry</td>
<td>x, y, r, xy, rxy</td>
<td>Symmetry</td>
</tr>
<tr>
<td>width, height</td>
<td></td>
<td>Site dimensions</td>
</tr>
<tr>
<td>orientation</td>
<td>N, E, W, S, FN, FE, FW, FS</td>
<td>Orientation</td>
</tr>
<tr>
<td>placement_rule</td>
<td>can_place</td>
<td>Site array available for floorplan</td>
</tr>
<tr>
<td></td>
<td>cannot_place</td>
<td>Site array not available for floorplan</td>
</tr>
<tr>
<td></td>
<td>regular</td>
<td>Placement grid</td>
</tr>
<tr>
<td>origin</td>
<td>x, y</td>
<td>Coordinate of the origin of site array</td>
</tr>
<tr>
<td>iterate</td>
<td>num_x</td>
<td>Number of columns in the site array</td>
</tr>
<tr>
<td></td>
<td>num_y</td>
<td>Number of rows in the site array</td>
</tr>
<tr>
<td>space_x</td>
<td></td>
<td>Column spacing (float)</td>
</tr>
<tr>
<td>space_y</td>
<td></td>
<td>Row spacing (float)</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
site(core) {
    site_class : core ;
    symmetry : x ;
    size (1, 10) ;
}
array(samplearray) {
    floorplan() { /* default floorplan */
        site_array(core) { /* Core cells placement */
            orientation : N ;
            placement_rule : can_place; /* available for placement */
            origin(0, 0) ;
            iterate(2, 4, 1.5, 0) ; /* site_array has 2 sites in x */
            /*direction spaced 1.5 um apart, 4 */
```
Defining the Global Cell Grid

You define the global cell grid overlaying the array by using the `routing_grid` attribute inside the `array` group. The router uses this grid during global routing.

Syntax

```liberty
array(array_name_id) {
    routing_grid() {
        routing_direction : horizontal | vertical ;
        grid_pattern (startfloat, gridsinteger, spacingfloat) ;
    }
}
```

where

- **start**
  A floating-point number representing the starting-point coordinate

- **grids**
  An integer number representing the number of grids in the x and y directions

- **spacing**
  A floating-point number representing the spacing between the grids in the x and y directions

Example

```liberty
array(samplearray) {
    routing_grid(0, 3, 1, 0, 3, 1) ;
    routing_direction(horizontal) ;
    grid_pattern(, ,) ;
    ...
}
```

Defining the Detail Routing Grid

You specify the routing track grid for the gate array by using the `tracks` group inside the `array` group. In the `tracks` group, you specify the track pattern, the track direction, and the layers available for the associated tracks.

Note:

- Define one `tracks` group for horizontal routing and one for vertical routing.
Syntax

array(array_nameid) {
    ... 
    tracks() {
        layers : "layer_1", "layer_2", ..."layer_n" ;
        routing_direction : vertical  |  horizontal ;
        track_pattern(start_point_float, num_of_tracks_float, 
                       space_between_tracks_float) ;
    }
}

where

start_point
    A floating-point number representing the starting-point coordinate

num_of_tracks
    A floating-point number representing the number of parallel tracks

space_between
    A floating-point number representing the spacing between the tracks

Example

phyis_library(example) {
    ... 
    resource(array) { /* gate array technology */ 
    ... 
    array(samplearray) {
        ... 
        tracks() {
            layers : "m1", "m3" ;
            routing_direction : horizontal ;
            track_pattern(1, 50, 10) ;
            /* 50 horizontal tracks 10 microns apart */
        } /* end tracks */
    } /* end resource */
    ... 
} /* end phys_library */
Defining the Design Rules

Specify design rules for the technology, such as minimum spacing and width, by using the `topological_design_rules` group.

This chapter includes the following sections:

- Defining Minimum Via Spacing Rules in the Same Net
- Defining Same-Net Minimum Wire Spacing
- Defining Same-Net Stacking Rules
- Defining Nondefault Rules for Wiring
- Defining Rules for Selecting Vias for Special Wiring
- Defining Rules for Generating Vias for Special Wiring
- Defining the Generated Via Size
Defining the Design Rules

The following sections describe how you define the design rules for physical libraries.

Defining Minimum Via Spacing Rules in the Same Net

The design rule checker requires the value for the edge-to-edge minimum spacing between vias.

Use the `contact_min_spacing` attribute for defining the minimum spacing between contacts in different nets. This attribute requires the name of the two contact layers and the spacing distance. To specify the minimum spacing between the same contact, use the same contact layer name twice.

**Syntax**

```plaintext
topological_design_rules() {
    contact_min_spacing(contact_layer1_id, contact_layer2_id, spacing_float);
    ...
}
```

**Example**

```plaintext
phys_library(sample) {
    ...
    topological_design_rules() {
        ...
        contact_min_spacing(cut01, cut12, 1);
        ...
    }
    ...
}
```

Defining Same-Net Minimum Wire Spacing

You can specify the minimum wire spacing between contacts in the same net by using the `same_net_min_spacing` attribute. To specify the minimum spacing between the same contact, use the same contact layer name twice.

**Syntax**

```plaintext
topological_design_rules() {
    same_net_min_spacing(layer1_name_id, layer2_name_id, spacing_float, ...);
    ...
}
```
Example

topological_design_rules() {
    same_net_min_spacing(m1, m1, 0.4, ...) ;
    same_net_min_spacing(m3, m3, 0.4, ...) ;
    ...
}

---

Defining Same-Net Stacking Rules

You can specify stacking for vias that share the same routing layer by setting the `is_stack` parameter in the `same_net_min_spacing` attribute to `TRUE`.

Syntax

topological_design_rules() {
    same_net_min_spacing(layer1_nameid, layer2_nameid, spacingfloat, is_stack[Boolean]) ;
    ...
}

Example

topological_design_rules() {
    same_net_min_spacing(m1, m1, 0.4, TRUE) ;
    same_net_min_spacing(m3, m3, 0.4, FALSE) ;
    ...
}

---

Defining Nondefault Rules for Wiring

For all regular wiring, you define the default rules by using either the `layer` group or the `via` group in the `resource` group. You define the nondefault rules for wiring by using the `wire_rule` group in the `topological_design_rules` group as shown here:

phys_library(sample) {
    ...
    topological_design_rules() {
        ...
        wire_rule(rule1) {
            via(non_default_via12) {
                ...
            }
            ...
        }
    }
}

You define the width, different net minimum spacing (edge-to-edge), and the wire extension by using the `layer_rule` group. The width and spacing specifications override the default values defined in the `routing_layer` group.
phys_library(sample) {
    ...
    topological_design_rules() {
        ...
        layer_rule(metal1) {
            /* non default regular wiring rules for metal1 */
            wire_width : 0.4 ; /* default is 0.32 */
            min_spacing : 0.4 ; /* default is 0.32 */
            wire_extension : 0.25 ; /* default is 0.4/2 */
        } /*end layer rule */
    }
}

Use the via group in the wire_rule group to define nondefault vias associated with the routing layers. This via group is similar to the via group in the resource group except that the is_default attribute is absent. For regular wiring, the via defined in the wire_rule group is considered instead of the default via defined in the resource group whenever the wire width matches the width specified in the via or layer group.

phys_library(sample) {
    ...
    topological_design_rules() {
        ...
        wire_rule(rule1) {
            via(non_default_via12) {
                ...
            }
        }
    }
}

For nondefault regular wiring, you define the via and routing layer spacing and the stacking rules by using the same_net_min_spacing attribute inside the wire_rule group. This attribute overrides the default values in the same_net_min_spacing attribute inside the topological_design_rules group.

phys_library(sample) {
    ...
    topological_design_rules() {
        ...
        wire_rule(rule1) {
            same_net_min_spacing(m1, m1, 0.32, FALSE) ;
            same_net_min_spacing(m2, m2, 0.4, FALSE) ;
            same_net_min_spacing(cut01, cut01, 0.36, FALSE) ;
            same_net_min_spacing(cut12, cut12, 0.36, FALSE) ;
        } /* end wire rule */
    } /* end design rules */
} /* end phys_library */

Use the vias attribute in the via_rule group to specify a list of vias. The router selects the first via that satisfies the design rules.
Defining Rules for Selecting Vias for Special Wiring

The `via_rule` group inside a `topological_design_rules` group defines vias used at the intersection of special wires in the same net.

You can specify multiple `via_rule` groups for a given layer pair. The rule that governs the selection of a `via_rule` group is the routing wire width range. When the width of a special wire is within the range specified, then the via rule is selected. When no via rule applies, then the default via rule is applied. The default via rule is created when you omit the routing wire width specification.

You also specify contact overhang and metal overhang, in both the horizontal and vertical directions, in the `via_rule` group. Contact overhang is the minimum amount of metal (wire) between the contact and the via edge. Metal overhang is at the edges of wire intersection. Figure 11-1 shows these relationships.

Figure 11-1  Contact Overhang and Metal Overhang

Syntax

topological_design_rules() {
  ...
  via_rule(via_rule_nameid) {
    vias : list_of_viasid;
    routing_layer_rule(routing_layer_nameid) {
      /* one for each layer associated with the via; */
      /* normally 2. */
      routing_direction : valueenum;
      /* direction of the overhang */
      contact_overhang : valuefloat;
      metal_overhang : valuefloat;
      min_wire_width : valuefloat;
      max_wire_width : valuefloat;
  }
Example
topological_design_rules() {
  ...
  via_rule(default_rule_for_m1_m2) {
    /* default via rule for the metal1, metal2 pair; */
    /* no wire width range is specified */
    via : "via12, via23" ;
    /* select via12 or via23 - whichever satisfies */
    /* the design rules*/
    routing_layer_rule(metal1) {
      routing_direction : horizontal ;
      contact_overhang : 0.1 ;
      metal_overhang : 0 ;
    }
    routing_layer_rule(metal2) {
      routing_direction : vertical ;
      contact_overhang : 0.1 ;
      metal_overhang : 0 ;
    }
  }
  ...
}

Defining Rules for Generating Vias for Special Wiring

Use the via_rule_generate group to specify the rules for generating vias used at the intersection of special wires in the same net. You define this group inside the topological_design_rules group. You can specify multiple via_rule_generate groups for a given layer pair.

The rule that governs the selection of a via_rule group is the routing wire width range. When the width of the special wire is within the range specified, then the via rule is selected. When no via rule applies, then the default via rule is applied. The default via rule is created when you omit the routing wire width specification.

Use the vias attribute in the via_rule_generate group to specify a list of vias. The router selects the first via that satisfies DRC. You also specify contact overhang and metal overhang, in both the horizontal and vertical directions, in the via_rule_generate group. Contact overhang is the minimum amount of metal (wire) between the contact and the via edge. Metal overhang is at the edges of wire intersection.

You specify the contact layer geometry generation formula in the contact_formula group inside the via_rule_generate group. The number of contact cuts in the generated array is determined by the contact spacing, contact-cut geometry, and the overhang (both contact and metal).
Syntax

topological_design_rules() {
    via_rule_generate(via_rule_nameid) {
        routing_layer_formula(routing_layer_nameid) {
            /* one for each layer associated with the via */
            /* normally 2 */
            routing_direction : valueenum;
            /* direction of the overhang */
            contact_overhang : valuefloat;
            metal_overhang : valuefloat;
            min_wire_width : valuefloat;
            max_wire_width : valuefloat;
        }
        contact_formula(contact_layer_name) {
            rectangle(x1float, y1float, x2float, y2float);
            /* specify more than 1 rectangle for */
            /* rectilinear vias */
            contact_spacing(x_spacingfloat, y_spacingfloat);
            resistance : valuefloat
        }
    }
}

Example

phys_library(sample) {
    resource(std_cell) { /* standard cell technology */
        ...
    } /* end resource */
    topological_design_rules() { /* design rules */
        same_net_min_spacing(m1, m1, 0.32, FALSE);
        /* minimum spacing required between 2 metal1 layers in the same net */
        same_net_min_spacing(m2, m2, 0.4, FALSE);
        /* minimum spacing required between 2 metal2 layers in the same net */
        same_net_min_spacing(m3, m3, 0.4, FALSE);
        /* minimum spacing required between 2 metal3 layers in the same net */
        same_net_min_spacing(cut01, cut01, 0.36, FALSE);
        /* minimum spacing required between 2 contact cut01 layers in the same net */
        same_net_min_spacing(cut12, cut12, 0.36, FALSE);
        /* minimum spacing required between 2 contact cut12 layers in the same net */
        same_net_min_spacing(cut23, cut23, 0.36, FALSE);
        /* minimum spacing required between 2 contact cut23 layers in the same net */
        /* via generation rules */
        via_rule_generate(default_rule_for_m1_m2) {
            routing_layer_formula(metal1) {
                routing_direction : horizontal;
                contact_overhang : 0.1;
                metal_overhang : 0.0;
            }
            routing_layer_rule(metal2) {
                routing_direction : vertical;
                contact_overhang : 0.1;
                metal_overhang : 0;
            }
            contact_formula(cut12) { /* rule for generating contact cut array */
            }
Defining the Generated Via Size

Generated vias are a multiple of the minimum feature size. The lithographic grid determines
the minimum feature size for the technology.

Syntax

```plaintext
min_generated_via_size(x_sizefloat, y_sizefloat);
```
Parasitic RC Estimation in the Physical Library

This chapter includes the following sections:

- Modeling Parasitic RC Estimation
- Variables Used in Parasitic RC Estimation
- Equations for Parasitic RC Estimation
- .plib Format
Modeling Parasitic RC Estimation

Figure A-1 provides an overview of the measures used in the parasitic RC estimation model.

Figure A-1  Parasitic RC Estimation Model

The following sections provide information about the variables and equations you use to model parasitic RC estimation.

Variables Used in Parasitic RC Estimation

The following sections list and describe the routing layer and routing wire variables you need to define in the RC estimation model.

Variables for Routing Layers

Define the following set of variables for each routing_layer group in your physical library.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_per_sq</td>
<td>Resistance per square of a res_per_sq routing layer.</td>
</tr>
</tbody>
</table>
### Variables for Estimated Routing Wire Model

Define the following set of variables for each `routing_wire_model` group in your physical library. Each `routing_wire_model` group represents a statistics-based design-specific estimation of interconnect topology.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cap_per_sq</code></td>
<td>Substrate capacitance per <code>cap_per_sq</code> square of a poly or metal layer (CP layer).</td>
</tr>
<tr>
<td><code>coupling_cap</code></td>
<td>Coupling capacitance per unit length between parallel wires on the same layer (CC layer).</td>
</tr>
<tr>
<td><code>fringe_cap</code></td>
<td>Fringe (sidewall) capacitance per unit length of a routing layer (CF layer).</td>
</tr>
<tr>
<td><code>edgecapacitance</code></td>
<td>Total fringe capacitance per unit length of routing layer. Specifies capacitance due to fringe, overlapping, and coupling effect.</td>
</tr>
<tr>
<td><code>inductance_per_dist</code></td>
<td>Inductance per unit length of a routing layer.</td>
</tr>
<tr>
<td><code>shrinkage</code></td>
<td>Distance that wires on the layer shrinks or expands on each side from the design to the fabricated chip. Note that negative numbers indicate expansion and positive number indicate shrinkage.</td>
</tr>
<tr>
<td><code>default_routing_width</code></td>
<td>Default routing width for wires on the layer.</td>
</tr>
<tr>
<td><code>height</code></td>
<td>Distance from the top of the substrate to the bottom of the routing layer.</td>
</tr>
<tr>
<td><code>thickness</code></td>
<td>Thickness of the routing layer.</td>
</tr>
<tr>
<td><code>plate_cap</code></td>
<td>Capacitance per unit area when the first layer overlaps the second layer. This function specifies an array of values indexed by routing layer order (CP layer, layer).</td>
</tr>
</tbody>
</table>

**Variables for Estimated Routing Wire Model**

Define the following set of variables for each `routing_wire_model` group in your physical library. Each `routing_wire_model` group represents a statistics-based design-specific estimation of interconnect topology.

**overlap_wire_ratio**

Percentage of the wiring on the first layer that overlaps the second layer. This function specifies all `overlap_wire_ratio` values in an n*(n-1) sized array, where n is the number of routing layers. For example, the `overlap_wire_ratio` values for the first routing layer (routing layer 1) are specified in `overlap_wire_ratio[0]` to `overlap_wire_ratio[n-2]`. The values for routing layer 2 are specified in `overlap_wire_ratio[n-1]` to `overlap_wire_ratio[2(n-1)]`. 
adjacent_wire_ratio

Percentage of wiring on the layer that runs adjacent to and has minimum spacing from wiring on the same layer. This function specifies percentage values of adjacent wiring for all routing layers. For example, two parallel adjacent wires with the same length would have an adjacent_wire_ratio of 50 percent.

wire_ratio_x

Percentage of total wiring in the horizontal direction that you estimate to be on each layer. The function carries an array of floating-point numbers, following the order of routing layers. That is, there are three floating-point numbers in the array if there are three routing layers. These numbers should add up to 1.00.

wire_ratio_y

Percentage of total wiring in the vertical direction that you estimate to be on each layer. The function carries an array of floating-point numbers, following the order of routing layers. That is, there are three floating point numbers in the array if there are three routing layers. And these numbers should add up to 1.00.

wire_length_x, wire_length_y

Estimated wire lengths in horizontal and vertical direction for a net.

Equations for Parasitic RC Estimation

Parasitic calculation is based on your estimates of routing topology prior to detail routing. The following sections describe how to determine those estimates.

Capacitance per Unit Length for a Layer

Use the following equations to estimate capacitance per unit length for a given layer.

\[
\text{cap}\_\text{per}\_\text{dist}_\text{layer} = \text{W} * \text{cap}\_\text{per}\_\text{area}_\text{layer} + \text{fringe}\_\text{cap}_\text{layer} + \\
\text{coupling}\_\text{cap}\_\text{per}\_\text{dist}_\text{layer}
\]

where

\[
\text{W} = (\text{default}\_\text{wire}\_\text{width} | \text{actual}\_\text{wire}\_\text{width}) - \text{shrinkage}
\]

\[
\text{cap}\_\text{per}\_\text{area}_\text{layer} = 1 - \sum \text{overlap}\_\text{wire}\_\text{ratio}\_\text{under}_\text{layer} * \\
\text{cap}\_\text{per}\_\text{sq}_\text{layer} + \\
\sum_{i=\text{other}\_\text{layer}} \text{overlap}\_\text{wire}\_\text{ratio}_j,\text{layer} * \text{plate}\_\text{cap}_\text{layer},i
\]

where
SUM_overlap_wire_ratio_underlayer =
  SUM_{j=layer_underneath}[overlap_wire_ratio_{j,layer}]

Note:
This equation represents the sum of all the overlap_wire_ratio values between the
current layer and each layer underneath the current layer.

coupling_cap_per_dist_{layer} =
2 \times adjacent_wire_ratio_{layer} \times coupling_cap_{layer}

Resistances and Capacitances for Each Routing Direction

Use the following equations to estimate capacitance and resistance values based on
orientational routing wire ratios.

capacitance_x = \text{cap_per_dist}_x \times \text{wire_length}_x
\text{capacitance}_y = \text{cap_per_dist}_y \times \text{wire_length}_y

resistance_x = \text{res_per_sq}_x \times \text{wire_length}_x / \text{width}_x
\text{resistance}_y = \text{res_per_sq}_y \times \text{wire_length}_y / \text{width}_y

where
\text{cap_per_dist}_x = \text{SUM}[\text{wire_ratio}_x_{layer} \times \text{cap_per_dist}_layer]
\text{cap_per_dist}_y = \text{SUM}[\text{wire_ratio}_y_{layer} \times \text{cap_per_dist}_layer]

\text{res_per_sq}_x = \text{SUM}[\text{wire_ratio}_x_{layer} \times \text{res_per_sq}_layer]
\text{res_per_sq}_y = \text{SUM}[\text{wire_ratio}_y_{layer} \times \text{res_per_sq}_layer]
\text{width}_x = \text{SUM}[\text{wire_ratio}_x_{layer} \times \text{W}_layer]
\text{width}_y = \text{SUM}[\text{wire_ratio}_y_{layer} \times \text{W}_layer]

.plib Format

To provide layer parasitics for RC estimation based on the equations shown in this section,
define them in the following .plib format.

```plaintext
physical_library(name) {
    ...
    resistance_lut_template (template_name_id) {
        variable_1: routing_width | routing_spacing ;
        variable_2: routing_width | routing_spacing ;
        index_1 ("float, float, float, ...") ;
        index_2 ("float, float, float, ...") ;
    }
    resource(technology) {
        field_oxide_thickness : float ;
        field_oxide_permitivity : float ;
        ...
        routing_layer(layer_name_id) {
            ...
        }
    }
}```
cap_multiplier : float ;
cap_per_sq : float ;
coupling_cap : float ;
default_routing_width : float ;
edgecapacitance : float ;
fringe_cap : float ;
height : float ;
inductance_per_dist : float ;
min_area : float ;
offset : float ;
oxide_permittivity : float ;
oxide_thickness : float ;
pitch : float ;
ranged_spacing(float, ..., float) ;
res_per_sq : float ;
routing_direction : vertical | horizontal ;
shrinkage : float ;
spacing : float ;
thickness : float ;
wire_extension : float ;
lateral_oxide (float, float) ;
resistance_table (template_nameid) {
    index_1 ("float, float, float, ...");
    index_2 ("float, float, float, ...");
    values ("float, float, float, ...");
}
}
/* end routing_layer */

plate_cap(value, value, value, value, value, ...) ;
/* capacitance between wires on lower and upper layer */
/* MUST BE DEFINED BEFORE ANY routing_wire_model GROUP DEFINITION */
/* AND AFTER ALL *_layer() DEFINITIONS */
routing_wire_model(name) {
    /* predefined routing wire ratio model for RC estimation */
    overlap_wire_ratio(value, value, value, value, value, ...) ;
    /* overlapping wiring percentage between wires on different layers. */
    /* Value between 0 and 100.0 */
    adjacent_wire_ratio(value, value, value, ...) ;
    /* Adjacent wire percentage between wires on same layers. */
    /* Value between 0.0 and 100.0 */
    wire_ratio_x(value, value, value, ...) ;
    /* x wiring percentage on each routing layer. */
    /* Value between 0.0 and 100.0 */
    wire_ratio_y(value, value, value, ...) ;
    /* y wiring percentage on each routing layer. */
    /* Value between 0.0 and 100.0 */
    wire_length_x : float ;
    /* estimated length for horizontal wire segment */
    wire_length_y : float ;
    /* estimated length for vertical wire segment */
}

topological_design_rules() {
    ...
    default_via_generate( ) { 
        via_routing_layer ( ) {
            end_of_line_overhang : ;
        }
    }
}
process_resource () {
    process_routing_layer () {
        res_per_sq : float;
        cap_per_sq : float;
        coupling_cap : float;
        /* coupling effect between parallel wires on same layer */
        fringe_cap : float; /* sidewall capacitance per unit length */
        edgecapacitance : float; /* lumped fringe capacitance */
        inductance_per_dist : float;
        shrinkage : float; /* delta width */
        default_routing_width : float; /* width */
        height : float; /* height from substrate */
        thickness : float; /* interconnect thickness */
        lateral_oxide_thickness : float;
        oxide_thickness : float;
    }
    process_via () {
        resistance : float;
    }
    process_array () {
        default_capacitance : float;
    }
    process_wire_rule () {
        process_via () {
            resistance : float;
        }
    }
}

macro() {
    ...
}

The .plib file that contains the wire_ratio model is as follows:

resource (technology) {
    routing_wire_model(name) {
        overlap_wire_ratio(value, value, value, ...);
        adjacent_wire_ratio(value, value, value, ...);
        wire_ratio_x(value, value, value, ...);
        wire_ratio_y(value, value, value, ...);
        wire_length_x : float;
        wire_length_y : float;
    }
}
Contents

1. Technology Library Group Description and Syntax
   Library-Level Attributes and Values ........................................... 1-2
   General Syntax ........................................................................... 1-4
   library Group Name ................................................................. 1-5
      Syntax ................................................................................... 1-5
      Example ................................................................................ 1-5
   library Group Example ............................................................. 1-6
   Simple Attributes ...................................................................... 1-6
      bus_naming_style Simple Attribute ........................................... 1-6
      comment Simple Attribute ....................................................... 1-7
      current_unit Simple Attribute ............................................... 1-7
      date Simple Attribute ......................................................... 1-7
      default_fpga_isd Simple Attribute .......................................... 1-8
      default_threshold_voltage_group Simple Attribute ................. 1-8
      delay_model Simple Attribute ............................................... 1-8
      distance_unit and dist_conversion_factor Attributes ................ 1-9
      critical_area_lut_template Group .......................................... 1-9
      device_layer, poly_layer, routing_layer, and cont_layer Groups .... 1-10
      em_temp_degradation_factor Simple Attribute ......................... 1-10
      input_threshold_pct_fall Simple Attribute ............................. 1-11
      input_threshold_pct_rise Simple Attribute ............................. 1-11
      is_soil Simple Attribute ...................................................... 1-12
      leakage_power_unit Simple Attribute .................................... 1-12
nom_process Simple Attribute ........................................... 1-12
nom_temperature Simple Attribute .................................... 1-13
nom_voltage Simple Attribute ......................................... 1-13
output_threshold_pct_fall Simple Attribute .............................. 1-13
output_threshold_pct_rise Simple Attribute ............................ 1-14
power_model Simple Attribute ......................................... 1-14
pulling_resistance_unit Simple Attribute ............................... 1-15
revision Simple Attribute ............................................. 1-15
slew_derate_from_library Simple Attribute ............................... 1-15
slew_lower_threshold_pct_fall Simple Attribute ......................... 1-16
slew_lower_threshold_pct_rise Simple Attribute ......................... 1-16
slew_upper_threshold_pct_fall Simple Attribute ......................... 1-16
slew_upper_threshold_pct_rise Simple Attribute ......................... 1-17
time_unit Simple Attribute ............................................ 1-17
voltage_unit Simple Attribute ......................................... 1-17

Defining Default Attribute Values in a CMOS Technology Library .......... 1-18

Complex Attributes ..................................................... 1-19

capacitive_load_unit Complex Attribute .................................. 1-19
default_part Complex Attribute ......................................... 1-20
define Complex Attribute ................................................ 1-20
define_cell_area Complex Attribute ................................... 1-21
define_group Complex Attribute ......................................... 1-22
receiver_capacitance_fall_threshold_pct Complex Attribute ............... 1-22
receiver_capacitance_rise_threshold_pct Complex Attribute ............... 1-23
technology Complex Attribute ......................................... 1-23
voltage_map Complex Attribute ...................................... 1-24

Group Statements ....................................................... 1-24

base_curves Group .................................................... 1-24
base_curve_type Complex Attribute .................................... 1-25
curve_x Complex Attribute ............................................ 1-25
curve_y Complex Attribute ............................................ 1-25
compact_lut_template Group .............................................. 1-26
base_curves_group Simple Attribute .................................. 1-26
variable_1 and variable_2 Simple Attributes ............................ 1-27
variable_3 Simple Attribute ........................................... 1-27
index_1 and index_2 Complex Attributes ........................................ 1-27
index_3 Complex Attribute .......................................................... 1-28
char_config Group ..................................................................... 1-28
  internal_power_calculation Simple Attribute .............................. 1-33
  three_state_disable_measurement_method Simple Attribute ......... 1-34
  three_state_disable_current_threshold_abs Simple Attribute ...... 1-34
  three_state_disable_current_threshold_rel Simple Attribute ...... 1-35
  three_state_disable_monitor_node Simple Attribute ................. 1-35
  three_state_cap_add_to_load_index Simple Attribute ............... 1-35
  ccs_timing_segment_voltage_tolerance_rel Simple Attribute ...... 1-36
  ccs_timing_delay_tolerance_rel Simple Attribute ..................... 1-36
  ccs_timing_voltage_margin_tolerance_rel Simple Attribute ....... 1-36
  CCS Receiver Capacitance Simple Attributes ......................... 1-37
  Input-Capacitance Measurement Simple Attributes ................. 1-37
  driver_waveform Complex Attribute ........................................ 1-38
  driver_waveform_rise Complex Attribute ................................. 1-38
  driver_waveform_fall Complex Attribute ................................ 1-38
  input_stimulus_transition Complex Attribute .......................... 1-39
  input_stimulus_interval Complex Attribute ............................. 1-39
  unrelated_output_net_capacitance Complex Attribute .............. 1-40
  default_value_selection_method Complex Attribute ................. 1-40
  default_value_selection_method_rise Complex Attribute ............ 1-40
  default_value_selection_method_fall Complex Attribute ............. 1-41
  merge_tolerance_abs Complex Attribute .................................. 1-41
  merge_tolerance_rel Complex Attribute .................................. 1-42
  merge_selection Complex Attribute ........................................ 1-42
  dc_current_template Group .................................................... 1-42
  em_lut_template Group .......................................................... 1-43
  fall_net_delay Group ............................................................ 1-45
  fall_transition_degradation Group .......................................... 1-46
  input_voltage Group ............................................................. 1-47
  fpga_isd Group .................................................................... 1-48
  lu_table_template Group .......................................................... 1-50
    normalized_voltage Variable .............................................. 1-51
    variable_1, variable_2, variable_3, and variable_4
    Simple Attributes ............................................................ 1-51
  maxcap_lut_template Group ..................................................... 1-54
  maxtrans_lut_template Group ................................................... 1-55
  normalized_driver_waveform Group ......................................... 1-56
    driver_waveform_name Simple Attribute ............................... 1-56
operating_conditions Group .............................................. 1-57
output_current_template Group ........................................ 1-59
output_voltage Group .................................................. 1-60
part Group ............................................................... 1-62
pg_current_template Group ............................................. 1-67
power_lut_template Group ............................................. 1-69
rise_net_delay Group .................................................... 1-71
rise_transition_degradation Group ................................. 1-72
sensitization Group ..................................................... 1-73
pin_names Complex Attribute ........................................ 1-73
vector Complex Attribute ............................................. 1-74
timing Group ........................................................... 1-75
type Group .................................................................. 1-75
user_parameters Group ................................................... 1-77
voltage_state_range_list Group .................................... 1-77
  voltage_state_range Attribute ..................................... 1-78
wire_load Group .......................................................... 1-78
wire_load_selection Group .............................................. 1-81
wire_load_table Group ................................................... 1-81

2. cell and model Group Description and Syntax

cell Group ................................................................. 2-2

Attributes and Values .................................................... 2-2

Simple Attributes ............................................................ 2-4
  always_on Simple Attribute ......................................... 2-4
  antenna_diode_type Simple Attribute ............................... 2-4
  area Simple Attribute .................................................. 2-4
  auxiliary_pad_cell Simple Attribute ................................ 2-5
  base_name Simple Attribute ......................................... 2-5
  bus_naming_style Simple Attribute ................................. 2-5
  cell_footprint Simple Attribute ...................................... 2-5
  cell_leakage_power Simple Attribute ................................ 2-6
  clock_gating_integrated_cell Simple Attribute ...................... 2-6
  contention_condition Simple Attribute ............................. 2-9
  dont_fault Simple Attribute .......................................... 2-9
  dont_touch Simple Attribute ......................................... 2-10
dont_use Simple Attribute ........................................... 2-10
driver_type Simple Attribute ...................................... 2-10
driver_waveform Simple Attribute ................................. 2-11
driver_waveform_rise and driver_waveform_fall
Simple Attributes .................................................. 2-12
eqctemp_degradation_factor Simple Attribute .................... 2-13
fpga_cell_type Simple Attribute ................................... 2-13
fpga_isd Simple Attribute ......................................... 2-13
interface_timing Simple Attribute .................................. 2-14
io_type Simple Attribute .......................................... 2-14
is_pad Simple Attribute ........................................... 2-14
is_pll_cell Simple Attribute ..................................... 2-15
is_clock_gating_cell Simple Attribute ........................... 2-16
is_clock_isolation_cell Simple Attribute ......................... 2-17
is_isolation_cell Simple Attribute ................................ 2-17
is_level_shifter Simple Attribute .................................. 2-17
is_macro_cell Simple Attribute .................................... 2-18
is_soi Simple Attribute ............................................ 2-18
level_shifter_type Simple Attribute ............................... 2-18
map_only Simple Attribute ........................................ 2-19
pad_cell Simple Attribute ........................................... 2-19
pad_type Simple Attribute ......................................... 2-20
power_cell_type Simple Attribute .................................. 2-20
power_gating_cell Simple Attribute ............................... 2-20
preferred Simple Attribute ........................................ 2-21
retention_cell Simple Attribute .................................... 2-21
sensitization_master Simple Attribute ............................ 2-22
single_bit_degenerate Simple Attribute ......................... 2-22
slew_type Simple Attribute ........................................ 2-23
switch_cell_type Simple Attribute ................................ 2-23
threshold_voltage_group Simple Attribute ....................... 2-23
timing_model_type Simple Attribute ............................... 2-24
use_for_size_only Simple Attribute ............................... 2-24

Complex Attributes .................................................. 2-25
input_voltage_range Attribute ..................................... 2-25
output_voltage_range Attribute .................................... 2-26
pin_equal Complex Attribute .......................................................... 2-26
pin_name_map Complex Attribute .................................................... 2-27
pin_opposite Complex Attribute ...................................................... 2-27
resource_usage Complex Attribute .................................................. 2-27

Group Statements ................................................................. 2-28
cell Group Example ........................................................................ 2-28
critical_area_table Group ............................................................... 2-30
bundle Group ................................................................................. 2-32
bus Group ....................................................................................... 2-39
char_config Group ......................................................................... 2-45
clear_condition Group .................................................................... 2-46
clock_condition Group ..................................................................... 2-47
dynamic_current Group ................................................................... 2-49
ff, latch, ff_bank, and latch_bank Groups ......................................... 2-50
  reference_pin_names Variable ......................................................... 2-50
  variable1 and variable2 Variables ................................................... 2-50
  bits Variable ................................................................................. 2-51
related_inputs Simple Attribute ..................................................... 2-51
related_outputs Simple Attribute .................................................. 2-51
typical_capacitances Simple Attribute ............................................ 2-52
when Simple Attribute ..................................................................... 2-53
switching_group Group ................................................................... 2-53
input_switching_condition Simple Attribute ..................................... 2-53
output_switching_condition Simple Attribute .................................. 2-54
min_input_switching_count Simple Attribute .................................. 2-54
max_input_switching_count Attribute ............................................. 2-55
pg_current Group ............................................................................ 2-55
compact_ccs_power Group ............................................................... 2-56
values Attribute .............................................................................. 2-57
vector Group .................................................................................. 2-57
index_1, index_, index_3, and index_4 Simple Attributes ..................... 2-58
index_output Simple Attribute ....................................................... 2-59
reference_time Simple Attribute .................................................... 2-59
values Simple Attribute ................................................................... 2-60
ff Group ......................................................................................... 2-60
ff_bank Group .................................................................................. 2-66
<table>
<thead>
<tr>
<th>Group Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fpga_condition Group</td>
<td>2-73</td>
</tr>
<tr>
<td>generated_clock Group</td>
<td>2-75</td>
</tr>
<tr>
<td>intrinsic_parasitic Group</td>
<td>2-78</td>
</tr>
<tr>
<td>total_capacitance Group</td>
<td>2-84</td>
</tr>
<tr>
<td>latch Group</td>
<td>2-85</td>
</tr>
<tr>
<td>latch_bank Group</td>
<td>2-89</td>
</tr>
<tr>
<td>leakage_current Group</td>
<td>2-97</td>
</tr>
<tr>
<td>gate_leakage Group</td>
<td>2-99</td>
</tr>
<tr>
<td>input_low_value Simple Attribute</td>
<td>2-100</td>
</tr>
<tr>
<td>input_high_value Simple Attribute</td>
<td>2-100</td>
</tr>
<tr>
<td>leakage_power Group</td>
<td>2-101</td>
</tr>
<tr>
<td>lut Group</td>
<td>2-104</td>
</tr>
<tr>
<td>mode_definition Group</td>
<td>2-104</td>
</tr>
<tr>
<td>mode_value Group</td>
<td>2-105</td>
</tr>
<tr>
<td>pg_setting_definition Group</td>
<td>2-106</td>
</tr>
<tr>
<td>default_pg_setting Simple Attribute</td>
<td>2-107</td>
</tr>
<tr>
<td>illegal_transition_if_undefined Simple Attribute</td>
<td>2-107</td>
</tr>
<tr>
<td>pg_setting_value Group</td>
<td>2-108</td>
</tr>
<tr>
<td>pg_setting_transition Group</td>
<td>2-109</td>
</tr>
<tr>
<td>pg_pin Group</td>
<td>2-110</td>
</tr>
<tr>
<td>is_insulated Simple Attribute</td>
<td>2-115</td>
</tr>
<tr>
<td>tied_to Simple Attribute</td>
<td>2-116</td>
</tr>
<tr>
<td>preset_condition Group</td>
<td>2-116</td>
</tr>
<tr>
<td>retention_condition Group</td>
<td>2-117</td>
</tr>
<tr>
<td>statetable Group</td>
<td>2-118</td>
</tr>
<tr>
<td>test_cell Group</td>
<td>2-118</td>
</tr>
<tr>
<td>type Group</td>
<td>2-123</td>
</tr>
<tr>
<td>model Group</td>
<td>2-125</td>
</tr>
</tbody>
</table>

### 3. pin Group Description and Syntax

Syntax of a pin Group in a cell or bus Group ........................................... 3-2
Simple Attributes ....................................................................................... 3-2
alive_during_power_up Simple Attribute .................................................... 3-4
always_on Simple Attribute ....................................................................... 3-4
antenna_diode_related_ground_pins Simple Attribute 3-5
antenna_diode_related_power_pins Simple Attribute 3-5
antenna_diode_type Simple Attribute 3-5
bit_width Simple Attribute 3-6
capacitance Simple Attribute 3-6
clamp_0_function Simple Attribute 3-6
clamp_1_function Simple Attribute 3-7
clamp_z_function Simple Attribute 3-7
clamp_latch_function Simple Attribute 3-7
clock Simple Attribute 3-8
clock_gate_clock_pin Simple Attribute 3-8
clock_gate_enable_pin Simple Attribute 3-8
clock_gate_test_pin Simple Attribute 3-9
clock_gate_obs_pin Simple Attribute 3-9
clock_gate_out_pin Simple Attribute 3-9
clock_isolation_cell_clock_pin Simple Attribute 3-10
complementary_pin Simple Attribute 3-10
connection_class Simple Attribute 3-11
data_in_type Simple Attribute 3-11
direction Simple Attribute 3-12
dont_fault Simple Attribute 3-12
drive_current Simple Attribute 3-13
driver_type Simple Attribute 3-13
driver_waveform Simple Attribute 3-19
driver_waveform_rise and driver_waveform_fall Simple Attributes 3-19
fall_capacitance Simple Attribute 3-19
fall_current_slope_after_threshold Simple Attribute 3-20
fall_current_slope_before_threshold Simple Attribute 3-21
fall_time_after_threshold Simple Attribute 3-21
fall_time_before_threshold Simple Attribute 3-21
fanout_load Simple Attribute 3-22
fault_model Simple Attribute 3-22
function Simple Attribute 3-23
has_buittin_pad Simple Attribute 3-25
has_pass_gate Simple Attribute 3-25
hysteresis Simple Attribute 3-25
illegal_clamp_condition Simple Attribute 3-26
input_map Simple Attribute 3-26
input_signal_level Simple Attribute 3-26
input_threshold_pct_fall Simple Attribute 3-27
input_threshold_pct_rise Simple Attribute 3-27
input_voltage Simple Attribute ................................. 3-28
internal_node Simple Attribute ................................. 3-28
inverted_output Simple Attribute .............................. 3-28
is_analog Attribute ............................................. 3-29
is_pad Simple Attribute ........................................ 3-29
is_pll_reference_pin Attribute ................................. 3-30
is_pll_feedback_pin Attribute ................................. 3-30
is_pll_output_pin Attribute .................................... 3-31
is_unbuffered Simple Attribute ............................... 3-32
isolation_cell_enable_pin Simple Attribute .................. 3-32
isolation_cell_data_pin Simple Attribute ..................... 3-33
permit_power_down Simple Attribute .......................... 3-33
alive_during_partial_power_down Simple Attribute ........... 3-33
is_isolated Simple Attribute .................................. 3-34
isolation_enable_condition Simple Attribute .................. 3-34
level_shifter_data_pin Simple Attribute ....................... 3-35
level_shifter_enable_pin Simple Attribute .................... 3-35
map_to_logic Simple Attribute ............................... 3-35
max_capacitance Simple Attribute ............................ 3-36
max_fanout Simple Attribute ................................... 3-36
max_transition Simple Attribute .............................. 3-37
min_capacitance Simple Attribute ............................ 3-37
min_fanout Simple Attribute .................................... 3-37
min_period Simple Attribute ................................... 3-38
min_pulse_width_high Simple Attribute ...................... 3-38
min_pulse_width_low Simple Attribute ....................... 3-39
multicell_pad_pin Simple Attribute ............................ 3-39
nextstate_type Simple Attribute ............................... 3-39
output_signal_level Simple Attribute ......................... 3-40
output_signal_level_high Simple Attribute ................... 3-40
output_signal_level_low Simple Attribute .................... 3-40
output_voltage Simple Attribute ............................. 3-40
pg_function Simple Attribute .................................. 3-41
pin_func_type Simple Attribute ............................... 3-41
power_down_function Simple Attribute ........................ 3-41
prefer_tied Simple Attribute ................................. 3-42
primary_output Simple Attribute ............................. 3-42
pulling_current Simple Attribute ............................. 3-42
pulling_resistance Simple Attribute .......................... 3-43
pulse_clock Simple Attribute .................................. 3-43
related_ground_pin Simple Attribute .......................... 3-43
related_power_pin Simple Attribute .......................................................... 3-44
restore_action Simple Attribute .............................................................. 3-44
restore_condition Simple Attribute ......................................................... 3-45
restore_edge_type Simple Attribute ......................................................... 3-45
rise_capacitance Simple Attribute .......................................................... 3-45
rise_current_slope_after_threshold Simple Attribute ............................. 3-46
rise_current_slope_before_threshold Simple Attribute ........................... 3-46
rise_time_after_threshold Simple Attribute ............................................. 3-47
rise_time_before_threshold Simple Attribute .......................................... 3-47
save_action Simple Attribute .................................................................. 3-47
save_condition Simple Attribute ............................................................. 3-48
signal_type Simple Attribute .................................................................. 3-48
slew_control Simple Attribute ................................................................ 3-50
slew_lower_threshold_pct_fall Simple Attribute ...................................... 3-50
slew_lower_threshold_pct_rise Simple Attribute ...................................... 3-51
slew_upper_threshold_pct_fall Simple Attribute ...................................... 3-51
slew_upper_threshold_pct_rise Simple Attribute ...................................... 3-52
state_function Simple Attribute ............................................................. 3-52
std_cell_main_rail Simple Attribute ......................................................... 3-52
switch_function Simple Attribute ........................................................... 3-53
switch_pin Simple Attribute ................................................................... 3-53
test_output_only Simple Attribute .......................................................... 3-53
three_state Simple Attribute .................................................................. 3-54
x_function Simple Attribute .................................................................. 3-54
Complex Attributes .................................................................................. 3-54
fall_capacitance_range Complex Attribute ........................................... 3-54
power_gating_pin Complex Attribute ....................................................... 3-55
retention_pin Complex Attribute ............................................................. 3-55
rise_capacitance_range Complex Attribute .............................................. 3-56
Group Statements .................................................................................... 3-56
ccsn_first_stage Group ........................................................................... 3-57
Syntax ........................................................................................................ 3-57
Simple Attributes ..................................................................................... 3-59
Complex Attribute ................................................................................... 3-59
Group Statements ..................................................................................... 3-59
is_inverting Simple Attribute .................................................................. 3-59
is_needed Simple Attribute ..................................................................... 3-59
is_pass_gate Simple Attribute .................................................................. 3-60
miller_cap_fall Simple Attribute ............................................................... 3-60
miller_cap_rise Simple Attribute ............................................................... 3-60
mode Attribute ......................................................................................... 3-61
stage_type Simple Attribute ........................................... 3-61
when Simple Attribute ............................................. 3-61
mode Complex Attribute ........................................... 3-62
dc_current Group ................................................... 3-62
output_voltage_fall Group ........................................... 3-62
output_voltage_rise Group ...................................... 3-63
propagated_noise_high Group .................................. 3-63
propagated_noise_low Group .................................... 3-64
ccsn_last_stage Group .............................................. 3-64
char_config Group ................................................... 3-64
electromigration Group ............................................ 3-65
  Simple Attributes .................................................. 3-65
  Complex Attributes ............................................. 3-66
  Group Statement .................................................. 3-66
related_pin Simple Attribute .................................. 3-66
related_bus_pins Simple Attribute .............................. 3-66
when Simple Attribute ............................................. 3-67
index_1 and index_2 Complex Attributes ....................... 3-67
values Complex Attribute ....................................... 3-68
em_max_toggle_rate Group ....................................... 3-68
input_ccb Group ................................................... 3-69
  Syntax ............................................................... 3-69
  Example ............................................................ 3-69
  Simple Attributes ............................................... 3-69
  related_ccb_node Simple Attribute ............................ 3-69
output_ccb Group ................................................... 3-70
internal_power Group ............................................... 3-70
  Simple Attributes .................................................. 3-70
  Complex Attribute ............................................. 3-70
  Group Statements .................................................. 3-70
Syntax for One-Dimensional, Two-Dimensional, and
Three-Dimensional Tables ........................................ 3-71
equal_or_opposite_output Simple Attribute .................... 3-72
falling_together_group Simple Attribute ......................... 3-73
power_level Simple Attribute ..................................... 3-73
related_pin Simple Attribute .................................. 3-74
related_pg_pin Simple Attribute ................................ 3-74
rising_together_group Simple Attribute ......................... 3-75
switching_interval Simple Attribute ............................. 3-75
switching_together_group Simple Attribute ....................... 3-76
when Simple Attribute ............................................. 3-76
mode Complex Attribute ................................. 3-78
fall_power Group ........................................ 3-78
power Group ............................................. 3-79
rise_power Group ........................................ 3-80
max_cap Group ............................................ 3-82
max_trans Group ........................................ 3-83
min_pulse_width Group ................................ 3-84

Syntax ................................................. 3-84
Simple Attributes ..................................... 3-85
Complex Attributes ................................. 3-85
constraint_high Simple Attribute ................. 3-85
when Simple Attribute ............................... 3-85
constraint_low Simple Attribute .................. 3-85
dsf_cond Simple Attribute ........................... 3-86
mode Complex Attribute ............................... 3-86

minimum_period Group ................................ 3-87

Syntax ................................................ 3-87
Simple Attributes ..................................... 3-87
Complex Attributes ................................. 3-87
constraint Simple Attribute ....................... 3-87
when Simple Attribute ............................... 3-88
dsf_cond Simple Attribute ........................... 3-88
mode Complex Attribute ............................... 3-88

receiver_capacitance Group ......................... 3-89
Groups ................................................. 3-89
Complex Attribute .................................... 3-91
Simple Attributes ..................................... 3-91

timing Group in a pin Group ......................... 3-92
Simple Attributes ..................................... 3-93
Complex Attributes .................................. 3-94
Group Statements .................................... 3-94
clock_gating_flag Simple Attribute .............. 3-95
default_timing Simple Attribute .................... 3-96
fpga_arc_condition Simple Attribute ............. 3-96
interdependence_id Simple Attribute ............... 3-96
output_signal_level_high Simple Attribute ....... 3-98
output_signal_level_low Simple Attribute ......... 3-98
related_output_pin Simple Attribute ............. 3-99
related_pin Simple Attribute ....................... 3-99
dsf_cond Simple Attribute ........................... 3-100
dsf_cond_end Simple Attribute ...................... 3-100
sdf_cond_start Simple Attribute .................................................. 3-101
sdf_edges Simple Attribute ......................................................... 3-101
sensitization_master Simple Attribute ........................................ 3-101
 timing_sense Simple Attribute .................................................. 3-102
timing_type Simple Attribute ...................................................... 3-103
wave_rise_sampling_index and wave_fall_sampling_index Attributes ...... 3-109
when Simple Attribute .............................................................. 3-110
when_end Simple Attribute ......................................................... 3-111
when_start Simple Attribute ....................................................... 3-111
active_input_ccb Complex Attribute ............................................. 3-111
active_output_ccb Complex Attribute ........................................... 3-112
function Complex Attribute ....................................................... 3-112
propagating_ccb Complex Attribute ............................................ 3-112
reference_input Complex Attribute ............................................. 3-113
mode Complex Attribute ........................................................... 3-113
pin_name_map Complex Attribute ................................................. 3-117
wave_rise and wave_fall Complex Attributes .................................. 3-118
wave_rise_time_interval and wave_fall_time_interval Complex Attributes .......................... 3-119
ccs_retain_rise and ccs_retain_fall Groups .................................... 3-120
cell_degradation Group .............................................................. 3-120
cell_fall Group .......................................................... 3-121
cell_rise Group ........................................................... 3-122
char_config Group ............................................................... 3-124
compact_ccs_retain_rise and compact_ccs_retain_fall Groups .......... 3-125
compact_ccs_rise and compact_ccs_fall Groups ................................ 3-125
base_curves_group Simple Attribute ............................................ 3-126
values Complex Attribute ......................................................... 3-126
fall_constraint Group .............................................................. 3-126
fall_propagation Group ............................................................. 3-127
fall_transition Group ............................................................... 3-128
ocv_sigma_cell_fall Group .......................................................... 3-129
ocv_sigma_cell_rise Group .......................................................... 3-130
ocv_sigma_fall_constraint Group ................................................... 3-130
ocv_sigma_fall_transition Group .................................................... 3-131
ocv_sigma_rise_constraint Group ................................................... 3-132
ocv_sigma_rise_transition Group .................................................... 3-133
ocv_sigma_retaining_fall Group .................................................... 3-133
ocv_sigma_retaining_rise Group .................................................... 3-134
ocv_sigma_retain_fall_slew Group .................................................. 3-134
ocv_sigma_retain_rise_slew Group ................................................ 3-135
ocv_mean_shift_“ Groups ............................................................ 3-135
ocv_skewness_* Groups ............................................. 3-136
ocv_std_dev_* Groups ............................................... 3-136
output_current_fall Group ........................................... 3-136
output_current_rise Group ......................................... 3-138
receiver_capacitance_fall Group .................................. 3-138
receiver_capacitance_rise Group ................................... 3-138
receiver_capacitance1_fall Group ................................ 3-138
receiver_capacitance1_rise Group ................................ 3-139
receiver_capacitance2_fall Group ................................ 3-139
receiver_capacitance2_rise Group ................................ 3-139
retaining_fall Group .................................................. 3-139
retaining_rise Group .................................................. 3-140
retain_fall_slew Group ................................................. 3-141
retain_rise_slew Group ................................................. 3-141
rise_constraint Group .................................................. 3-142
rise_propagation Group ................................................. 3-144
rise_transition Group .................................................. 3-144
tlatch Group ............................................................. 3-145
edge_type Simple Attribute......................................... 3-146
tdisable Simple Attribute............................................. 3-146
This chapter describes the role of the library group in defining a CMOS technology library. This chapter also includes descriptions and syntax examples for all the attributes and groups that you can define within the library group, with the following exceptions:

- The cell and model groups, which are described in Chapter 2, “cell and model Group Description and Syntax.”
- The pin group, which is described in “pin Group Description and Syntax” in Chapter 3.

This chapter provides information about the library group in the following sections:

- Library-Level Attributes and Values
- General Syntax
- library Group Name
- library Group Example
- Simple Attributes
- Defining Default Attribute Values in a CMOS Technology Library
- Complex Attributes
- Group Statements
Library-Level Attributes and Values

The library group is the superior group in a technology library. The library group contains all the groups and attributes that define the technology library.

Example 1-1 lists alphabetically a sampling of the attributes, groups, and values that you can define within a technology library. Example 1-2 on page 1-4 shows the general syntax and the functional order in which the attributes usually appear within a library group.

Example 1-1 Attributes, Groups, and Values in a Technology Library

```plaintext
library {name_string} {
    ... library description ...
}

/* Library Description: Simple Attributes */

bus_naming_style : "string";
comment : "string";
current_unit : value_enum;
date : "date";
delay_model : value_enum;
em_temp_degradation_factor : float;
inout_threshold_pct_fall : trip_point value;
inout_threshold_pct_rise : trip_point value;
is_soI : true | false;
leakage_power_unit : value_enum;
nom_process : float;
nom_temperature : float;
nom_voltage : float;
output_threshold_pct_fall : trip_point value;
output_threshold_pct_rise : trip_point value;
power_model : table_lookup;
pulling_resistance_unit : 1ohm | 10ohm | 100ohm | 1kohm;
revision : float | string;
slew_derate_from_library : derate value;
slew_lower_threshold_pct_fall : trip_point value;
slew_lower_threshold_pct_rise : trip_point value;
slew_upper_threshold_pct_fall : trip_point value;
slew_upper_threshold_pct_rise : trip_point value;
time_unit : 1ps | 10ps | 100ps | 1ns;
voltage_unit : 1mV | 10mV | 100mV | 1V;

/* Library Description: Default Attributes */

default_cell_leakage_power : float;
default_connection_class : name | name_list_string;
default_fanout_load : float;
default_inout_pin_cap : float;
default_input_pin_cap : float;
default_max_capacitance : float;
default_max_fanout : float;
default_max_transition : float;
default_operating_conditions : name_string;
```
default_output_pin_cap : float;
default_wire_load : namestring;
default_wire_load_area : float;
default_wire_load_capacitance : float;
default_wire_load_mode : top | segmented | enclosed;
default_wire_load_resistance : float;
default_wire_load_selection : namestring;

/* Library Description: Scaling Attributes */
k_process_cell_fall : float ; /* nonlinear model only */
k_process_cell_rise : float ; /* nonlinear model only */
k_process_fall_propagation : float ; /* nonlinear model only */
k_process_fall_transition : float ; /* nonlinear model only */
k_process_pin_cap : float;
k_process_rise_propagation : float ; /* nonlinear model only */
k_process_rise_transition : float ; /* nonlinear model only */
k_process_wire_cap : float;
k_temp_cell_rise : float ; /* nonlinear model only */
k_temp_cell_fall : float ; /* nonlinear model only */
k_temp_fall_propagation : float ; /* nonlinear model only */
k_temp_fall_transition : float ; /* nonlinear model only */
k_temp_pin_cap : float;
k_temp_rise_propagation : float /* nonlinear model only */
k_temp_rise_transition : float ; /* nonlinear model only */
k_temp_rise_wire_resistance : float;
k_temp_rise_wire_resistance : float;
k_temp_wire_cap : float;
k_volt_cell_fall : float ; /* nonlinear model only */
k_volt_cell_rise : float ; /* nonlinear model only */
k_volt_fall_propagation : float ; /* nonlinear model only */
k_volt_fall_transition : float ; /* nonlinear model only */
k_volt_pin_cap : float;
k_volt_rise_propagation : float ; /* nonlinear model only */
k_volt_rise_transition : float ; /* nonlinear model only */
k_volt_wire_cap : float;

/* Library Description: Complex Attributes */
capacitive_load_unit (value, unit);
default_part (default_part_nameid, speed_gradeid);
define (name, object, type) ; /*user-defined attributes only */
define_cell_area (area_name, resource_type);
define_group (attribute_namestring, group_namestring,attribute_typestring);
library_features (value_1, value_2, ..., value_n);
receiver_capacitance_rise_threshold_pct ("float, float, ...") ;
receiver_capacitance_fall_threshold_pct ("float, float, ...") ;
technology ("name") ;

/* Library Description: Group Statements*/
cell (name) { }
dc_current_template (template_nameid) { }
em_lut_template (name) { }
fall_net_delay : name;
fall_transition_degradation (name) { }
input_voltage (name) { }
lu_table_template (name) { }
ocv_derate (name) { }
ocv_table_template (template_name) { }
operating_conditions (name) { }
output_voltage (name) { }
part (name) { }
power_lut_template (template_nameid) { }
rise_net_delay : name;
rise_transition_degradation () { }
timing (name | name_list) { }
type (name) { }
voltage_state_range_list
wire_load (name) { }
wire_load_selection (name) { }
wire_load_table (name) { }

---

**General Syntax**

Example 1-2 shows the general syntax of the library group. The first line names the library. Subsequent lines show simple and complex attributes that apply to the library as a whole, such as technology, date, and revision.

The example indicates where you place the default and scaling factors in library syntax. Group statements complete the library syntax.

Every cell in the library has a separate cell description.

Note: Example 1-2 does not contain every attribute or group listed in Example 1-1.

**Example 1-2  General Syntax of a Technology Library**

library (namestring) {

/* Library-Level Simple and Complex Attributes */

technology (nameenum);
delay_model : "model";
bus_naming_style : "string";
date : "date";
comment : "string";
time_unit : "unit";
voltage_unit : "unit";
leakage_power_unit : "unit";
current_unit : "unit";
pulling_resistance_unit : "unit";
capacitive_load_unit (value, unit);
define_cell_area (area_name, resource_type);
revision : float | string;

/* Default Attributes and Values (not shown here)*/

/* Scaling Factors Attributes and Values (not shown here)*/
library Group Name

The first line of the library group statement names the library. It is the first executable line in your library.

Syntax

```plaintext
library(name_string) {
  ... library description ...
}
```

Example

A library called example1 looks like this:

```plaintext
library (example1) {
  ... library description...
}
```
library Group Example

Example 1-3 shows a portion of a library group for a CMOS library. It contains buses and uses a nonlinear timing delay model.

Example 1-3 CMOS library Group

```plaintext
library (example1) {
  technology (cmos) ;
  delay_model : table_lookup ;
  date : "December 12, 2013" ;
  revision : 2013.12 ;
  bus_naming_style : "Bus%sPin%d" ;
  ...
}
```

Simple Attributes

Following are descriptions of the library group simple attributes. Similar sections describing the default, complex, and group statement attributes complete this chapter.

bus_naming_style Simple Attribute

The `bus_naming_style` attribute defines the naming convention for buses in the library.

Syntax

```plaintext
bus_naming_style : "string";
```

string

Contains alphanumeric characters, braces, underscores, dashes, or parentheses. Must contain one `%s` symbol and one `%d` symbol. The `%s` and `%d` symbols can appear in any order with at least one nonnumeric character in between.

The colon character is not allowed in a `bus_naming_style` attribute value because the colon is used to denote a range of bus members. You construct a complete bused-pin name by using the name of the owning bus and the member number. The owning bus name is substituted for the `%s`, and the member number replaces the `%d`.

If you do not define the `bus_naming_style` attribute, the default naming convention is applied, as shown.

Example

```plaintext
bus_naming_style : "%s[%d]" ;
```
When the default naming convention is applied, member 1 of bus A becomes $A[1]$.

The next example shows how you can use the `bus_naming_style` attribute to apply a different naming convention.

**Example**
```
bus_naming_style : "Bus%sPin%d" ;
```

When this naming convention is applied, bus member 1 of bus A becomes `BusAPin1`, bus member 2 becomes `BusAPin2`, and so on.

---

**comment Simple Attribute**

You use the `comment` attribute to include copyright or other product information in the library report. You can include only one comment line in a library.

**Syntax**
```
comment : "string" ;
```

**string**

You can use an unlimited number of characters in the string, but all the characters must be enclosed within quotation marks.

---

**current_unit Simple Attribute**

The `current_unit` attribute specifies the unit for the drive current generated by output pads. The `pulling_current` attribute for a pull-up or pull-down transistor also represents its values in the specified unit.

**Syntax**
```
current_unit : value_enum ;
```

**value**

The valid values are 1uA, 10uA, 100uA, 1mA, 10mA, 100mA, and 1A. No default exists for the `current_unit` attribute if the attribute is omitted.

**Example**
```
current_unit : 100uA ;
```

---

**date Simple Attribute**

The optional `date` attribute identifies the date your library was created.
Syntax

```
date : "date" ;
```

`date`

You can use any format within the quotation marks to report the date.

Example

```
date : "12 December 2003" ;
```

---

default_fpga_isd Simple Attribute

If you define more than one `fpga_isd` group, you must use the `default_fpga_isd` attribute to specify which of those `fpga_isd` groups is the default.

Syntax

```
default_fpga_isd : fpga_isd_name_id ;
```

`fpga_isd_name`

The name of the default `fpga_isd` group.

Example

```
default_fpga_isd : lib_isd ;
```

---

default_threshold_voltage_group Simple Attribute

The optional `default_threshold_voltage_group` attribute specifies a cell's category based on its threshold voltage characteristics.

Syntax

```
default_threshold_voltage_group : group_name_id ;
```

`group_name`

A string value representing the name of the category.

Example

```
default_threshold_voltage_group : "high_vt_cell" ;
```

---

delay_model Simple Attribute

Use the `delay_model` attribute to specify which delay model to use in the delay calculations.
The delay_model attribute must be the first attribute in the library if a technology attribute is not present. Otherwise, it should follow the technology attribute.

**Syntax**

```
delay_model : valueenum;
```

**value**

Valid value is table_lookup.

**Example**

```
delay_model : table_lookup;
```

distance_unit and dist_conversion_factor Attributes

The distance_unit attribute specifies the distance unit and the resolution, or accuracy, of the values in the critical_area_table table in the critical_area_lut_template group. The distance and area values are represented as floating-point numbers that are rounded in the critical_area_table. The distance values are rounded by the dist_conversion_factor and the area values are rounded by the dist_conversion_factor squared.

**Syntax**

```
distance_unit : enum (um, mm);
dist_conversion_factor : integer;
```

**Example**

```
library(my_library) {

distance_unit : um;
dist_conversion_factor : 1000;
critical_area_lut_template (caa_template) {
    variable_1 : defect_size_diameter;
    index_1 ("0.05, 0.10, 0.15, 0.20, 0.25, 0.30");
}
```

critical_area_lut_template Group

The critical_area_lut_template group is a critical area lookup table used only for critical area analysis modeling. The defect_size_diameter is the only valid value.

**Syntax**

```
critical_area_lut_template {template_name} {
```
Example

```plaintext
library(my_library) {

distance_unit : um;
dist_conversion_factor : 1000;
critical_area_lut_template (caa_template) {
    variable_1 : defect_size_diameter;
    index_1 ("0.05, 0.10, 0.15, 0.20, 0.25, 0.30");
}
}
```

**device_layer, poly_layer, routing_layer, and cont_layer Groups**

Because yield calculation varies among different types of layers, Liberty syntax supports the following types of layers: device, poly, routing, and contact (via) layers. The device_layer, poly_layer, routing_layer, and cont_layer groups define layers that have critical area data modeled on them for cells in the library. The layer definition is specified at the library level. It is recommended that you declare the layers in order, from the bottom up. The layer names specified here must match the actual layer names in the corresponding physical libraries.

**Syntax**

```plaintext
device_layer(string) {} /* such as diffusion layer OD */
```

Example

```plaintext
library(my_library) {

distance_unit : um;
dist_conversion_factor : 1000;
critical_area_lut_template (caa_template) {
    variable_1 : defect_size_diameter;
    index_1 ("0.05, 0.10, 0.15, 0.20, 0.25, 0.30");
}
}
```

```plaintext
device_layer(string) {} /* such as diffusion layer OD */
poly_layer(string) {} /* such as poly layer */
routing_layer(string) {} /* such as M1, M2, ... */
cont_layer(string) {} /* via layer, such as VIA */
```

**em_temp_degradation_factor Simple Attribute**

The em_temp_degradation_factor attribute specifies the electromigration exponential degradation factor.

**Syntax**

```plaintext
em_temp_degradation_factor : valuefloat ;
```
value
A floating-point number in centigrade units consistent with other temperature specifications throughout the library.

Example
em_temp_degradation_factor : 40.0 ;

input_threshold_pct_fall Simple Attribute
Use the `input_threshold_pct_fall` attribute to set the default threshold point on an input pin signal falling from 1 to 0. You can specify this attribute at the pin-level to override the default.

Syntax
input_threshold_pct_fall : trip_pointfloat ;

trip_point
A floating-point number between 0.0 and 100.0 that specifies the threshold point of an input pin signal falling from 1 to 0. The default is 50.0.

Example
input_threshold_pct_fall : 60.0 ;

input_threshold_pct_rise Simple Attribute
Use the `input_threshold_pct_rise` attribute to set the default threshold point on an input pin signal rising from 0 to 1. You can specify this attribute at the pin-level to override the default.

Syntax
input_threshold_pct_rise : trip_pointfloat ;

trip_point
A floating-point number between 0.0 and 100.0 that specifies the threshold point of an input pin signal rising from 0 to 1. The default is 50.0.

Example
input_threshold_pct_rise : 40.0 ;
is_soi Simple Attribute

The is_soi attribute specifies that all the cells in a library are silicon-on-insulator (SOI) cells. The default is false, which means that the library cells are bulk-CMOS cells.

If the is_soi attribute is specified at both the library and cell levels, the cell-level value overrides the library-level value.

Syntax

\[
\text{is_soi : true | false ;}
\]

Example

\[
\text{is_soi : true ;}
\]

For more information about the is_soi attribute and SOI cells, see the "Advanced Low-Power Modeling" chapter of the Liberty User Guide, Vol. 1.

leakage_power_unit Simple Attribute

The leakage_power_unit attribute is defined at the library level. It indicates the units of the power values in the library. If this attribute is missing, the leakage-power values are expressed without units.

Syntax

\[
\text{leakage_power_unit : valueenum ;}
\]

value

Valid values are 1mW, 100mW, 10mW, 1mW, 100nW, 10nW, 1nW, 100pW, 10pW, and 1pW.

Example

\[
\text{leakage_power_unit : "100mW" ;}
\]

nom_process Simple Attribute

The nom_process attribute defines process scaling, one of the nominal operating conditions for a library.

Syntax

\[
\text{nom_process : valuefloat ;}
\]
value

A floating-point number that represents the degree of process scaling in the cells of the library.

Example
nom_process : 1.0 ;

---

**nom_temperature Simple Attribute**

The `nom_temperature` attribute defines the temperature (in centigrade), one of the nominal operating conditions for a library.

**Syntax**
nom_temperature : value_float ;

**value**

A floating-point number that represents the temperature of the cells in the library.

Example
nom_temperature : 25.0 ;

---

**nom_voltage Simple Attribute**

The `nom_voltage` attribute defines voltage, one of the nominal operating conditions for a library.

**Syntax**
nom_voltage : value_float ;

**value**

A floating-point number that represents the voltage of the cells in the library.

Example
nom_voltage : 5.0 ;

---

**output_threshold_pct_fall Simple Attribute**

Use the `output_threshold_pct_fall` attribute to set the value of the threshold point on an output pin signal falling from 1 to 0.
Syntax
output_threshold_pct_fall : trip_pointfloat ;

trip_point
A floating-point number between 0.0 and 100.0 that specifies the threshold point of an output pin signal falling from 1 to 0. The default is 50.0.

Example
output_threshold_pct_fall : 40.0 ;

output_threshold_pct_rise Simple Attribute
Use the output_threshold_pct_rise attribute to set the value of the threshold point on an output pin signal rising from 0 to 1.

Syntax
output_threshold_pct_rise : trip_pointfloat ;

trip_point
A floating-point number between 0.0 and 100.0 that specifies the threshold point of an output pin signal rising from 0 to 1. The default is 50.0.

Example
output_threshold_pct_rise : 40.0 ;

definition of the power_model attribute
Use the power_model attribute to specify the power model in your library.

Syntax
power_model : value ;

value
The valid value is table_lookup.

Example
power_model : table_lookup ;
pulling_resistance_unit Simple Attribute

Use the `pulling_resistance_unit` attribute to define pulling resistance unit values for pull-up and pull-down devices.

**Syntax**

```plaintext
pulling_resistance_unit : "unit" ;
```

**unit**

Valid unit values are 1ohm, 10ohm, 100ohm, and 1kohm. No default exists for `pulling_resistance_unit` if the attribute is omitted.

**Example**

```plaintext
pulling_resistance_unit : "10ohm" ;
```

revision Simple Attribute

The optional `revision` attribute defines a revision number for your library.

**Syntax**

```plaintext
revision : value ;
```

**value**

The value can be either a floating-point number or a string.

**Example**

```plaintext
revision : V3.1a ;
```

slew_derate_from_library Simple Attribute

Use the `slew_derate_from_library` attribute to specify how the transition times need to be derated to match the transition times between the characterization trip points.

**Syntax**

```plaintext
slew_derate_from_library : deratefloat ;
```

**derate**

A floating-point number between 0.0 and 1.0. The default is 1.0.

**Example**

```plaintext
slew_derate_from_library : 0.5;
```
**slew_lower_threshold_pct_fall Simple Attribute**

Use the `slew_lower_threshold_pct_fall` attribute to set the default lower threshold point that is used to model the delay of a pin falling from 1 to 0. You can specify this attribute at the pin-level to override the default.

**Syntax**

```
slew_lower_threshold_pct_fall : trip_pointvalue ;
```

**trip_point**

A floating-point number between 0.0 and 100.0 that specifies the lower threshold point used to model the delay of a pin falling from 1 to 0. The default is 20.0.

**Example**

```
slew_lower_threshold_pct_fall : 30.0 ;
```

**slew_lower_threshold_pct_rise Simple Attribute**

Use the `slew_lower_threshold_pct_rise` attribute to set the default lower threshold point that is used to model the delay of a pin rising from 0 to 1. You can specify this attribute at the pin-level to override the default.

**Syntax**

```
slew_lower_threshold_pct_rise : trip_pointvalue ;
```

**trip_point**

A floating-point number between 0.0 and 100.0 that specifies the lower threshold point used to model the delay of a pin rising from 0 to 1. The default is 20.0.

**Example**

```
slew_lower_threshold_pct_rise : 30.0 ;
```

**slew_upper_threshold_pct_fall Simple Attribute**

Use the `slew_upper_threshold_pct_fall` attribute to set the default upper threshold point that is used to model the delay of a pin falling from 1 to 0. You can specify this attribute at the pin-level to override the default.

**Syntax**

```
slew_upper_threshold_pct_fall : trip_pointvalue ;
```
**trip_point**

A floating-point number between 0.0 and 100.0 that specifies the upper threshold point to model the delay of a pin falling from 1 to 0. The default is 80.0.

**Example**

```
slew_upper_threshold_pct_fall : 70.0 ;
```

**slew_upper_threshold_pct_rise** Simple Attribute

Use the `slew_upper_threshold_pct_rise` attribute to set the value of the upper threshold point that is used to model the delay of a pin rising from 0 to 1. You can specify this attribute at the pin-level to override the default.

**Syntax**

```
slew_upper_threshold_pct_rise : trip_pointvalue ;
```

**trip_point**

A floating-point number between 0.0 and 100.0 that specifies the upper threshold point used to model the delay of a pin rising from 0 to 1. The default is 80.0.

**Example**

```
slew_upper_threshold_pct_rise : 70.0 ;
```

**time_unit** Simple Attribute

Use the optional `time_unit` attribute to specify the time units.

**Syntax**

```
time_unit : unit ;
```

**unit**

Valid values are 1ps, 10ps, 100ps, and 1ns. The default is 1ns.

**Example**

```
time_unit : 100ps ;
```

**voltage_unit** Simple Attribute

Use the `voltage_unit` attribute to specify the voltage units.
Additionally, the `voltage` attribute in the `operating_conditions` group represents values in the voltage units.

**Syntax**

```plaintext
voltage_unit : unit ;

unit

Valid values are 1mV, 10mV, 100mV, and 1V. The default is 1V.
```

**Example**

```plaintext
voltage_unit : 100mV;
```

---

**Defining Default Attribute Values in a CMOS Technology Library**

Within the `library` group of a CMOS technology library, you can define default values for the `pin` and `timing` group attributes. Then, as needed, you can override the default settings by defining corresponding attributes in the individual `pin` or `timing` groups.

**Table 1-1** lists the default attributes that you can define within the `library` group and the attributes that override them.

**Table 1-1  CMOS Default Attributes for All Models**

<table>
<thead>
<tr>
<th>Default attribute</th>
<th>Description</th>
<th>Override with</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>default_cell_leakage_power</code></td>
<td>Default leakage power</td>
<td><code>cell_leakage_power</code></td>
</tr>
<tr>
<td><code>default_connection_class</code></td>
<td>Default connection class</td>
<td><code>connection_class</code></td>
</tr>
<tr>
<td><code>default_fanout_load</code></td>
<td>Fanout load of input pins</td>
<td><code>fanout_load</code></td>
</tr>
<tr>
<td><code>default_inout_pin_cap</code></td>
<td>Capacitance of inout pins</td>
<td><code>capacitance</code></td>
</tr>
<tr>
<td><code>default_input_pin_cap</code></td>
<td>Capacitance of input pins</td>
<td><code>capacitance</code></td>
</tr>
<tr>
<td><code>default_max_capacitance</code></td>
<td>Maximum capacitance of output pins</td>
<td><code>max_capacitance</code></td>
</tr>
</tbody>
</table>
Complex Attributes

Following are the descriptions of the technology library complex attributes.

**capacitive_load_unit Complex Attribute**

The `capacitive_load_unit` attribute specifies the unit for all capacitance values within the technology library, including default, maximum fanout, pin, and wire capacitances.
Syntax

```plaintext
capacitive_load_unit (value\textit{float}, \textit{unit\textit{enum}}) ;
```

**value**
- A floating-point number.

**unit**
- Valid values are ff and pf.

**Example**

The first line in the following example sets the capacitive load unit to 1 pf. The second line represents capacitance in terms of the standard unit load of an inverter.

```plaintext
capacitive_load_unit (1, pf) ;
capacitive_load_unit (0.059, pf) ;
```

The `capacitive_load_unit` attribute has no default when the attribute is omitted.

---

**default_part Complex Attribute**

The `default_part` attribute specifies the default part and the speed used for an FPGA design.

**Syntax**

```plaintext
default_part (default_part\_name\textit{string}, \textit{speed\_grade\_string}) ;
```

**default_part\_name**
- The name of the default part.

**speed\_grade**
- The speed grade the design uses.

**Example**

```plaintext
default_part ("AUTO", "-5") ;
```

---

**define Complex Attribute**

Use this attribute to define new, temporary, or user-defined attributes for use in symbol and technology libraries.

**Syntax**

```plaintext
define ("attribute\_name", "group\_name", "attribute\_type") ;
```
attribute_name
The name of the attribute you are creating.

group_name
The name of the group statement in which the attribute is to be used.

attribute_type
The type of the attribute that you are creating; valid values are Boolean, string, integer, or float.

You can use either a space or a comma to separate the arguments. The following example shows how to define a new string attribute called bork, which is valid in a pin group:

**Example**
```
define ("bork", "pin", "string") ;
```

You give the new library attribute a value by using the simple attribute syntax:
```
bork : "nimo" ;
```

---

**define_cell_area Complex Attribute**

The `define_cell_area` attribute defines the area resources a cell uses, such as the number of pad slots.

**Syntax**
```
define_cell_area (area_name, resource_type) ;
```

**area_name**
A name of a resource type. You can associate more than one `area_name` attribute with each of the predefined resource types.

**resource_type**
The resource type can be
- pad_slots
- pad_input_driver_sites
- pad_output_driver_sites
- pad_driver_sites

Use the pad_driver_sites type when you do not need to discriminate between input and output pad driver sites.
You can define as many cell area types as you need, as shown here.

**Example**

define_cell_area (bond_pads, pad_slots) ;
define_cell_area (pad_drivers, pad_driver_sites) ;

After you define the cell area types, specify the resource type in a cell group to identify how many of each resource type the cell requires, as shown here.

**Example**
cell(IV_PAD) {
  bond_pads : 1 ;
  ...
}

---

**define_group Complex Attribute**

Use this special attribute to define new, temporary, or user-defined groups for use in technology libraries.

**Syntax**

define_group (group_id, parent_name_id) ;

group
  The name of the user-defined group.

parent_name
  The name of the group statement in which the attribute is to be used.

**Example**
The following example shows how you define a new group called myGroup:

define_group (myGroup, timing ) ;

---

**receiver_capacitance_fall_threshold_pct Complex Attribute**

The receiver_capacitance_fall_threshold_pct attribute specifies the points that separate the voltage fall segments in the multi-segment receiver capacitance model. Specify each point as a percentage of the rail voltage between 0.0 and 100.0.

Specify monotonically decreasing values with the receiver_capacitance_fall_threshold_pct attribute.
**Syntax**

receiver_capacitance_fall_threshold_pct ("float, float,...");

**Example**

receiver_capacitance_fall_threshold_pct ("100, 80, 70, 60, 50, 30, 0");

In this example, six segments are defined and the first segment is from 100 percent to 80 percent of the rail voltage.

---

**receiver_capacitance_rise_threshold_pct Complex Attribute**

The receiver_capacitance_rise_threshold_pct attribute specifies the points that separate the voltage rise segments in the multi-segment receiver capacitance model. Specify the points as percentage of the rail voltage between 0.0 and 100.0.

Specify monotonically increasing values with the receiver_capacitance_rise_threshold_pct attribute.

**Syntax**

receiver_capacitance_rise_threshold_pct ("float, float,...");

**Example**

receiver_capacitance_rise_threshold_pct ("0, 30, 50, 60, 70, 80, 100");

In this example, six segments are defined and the first segment is from zero percent to 30 percent of the rail voltage.

---

**technology Complex Attribute**

The technology attribute statement specifies the technology family being used in the library. When you define the technology attribute, it must be the first attribute you use and it must be placed at the top of the listing (see Example 1-2 on page 1-4).

**Syntax**

technology (nameenum);

**name**

Valid value is CMOS.

**Example**

technology (cmos);
**voltage_map Complex Attribute**

Use the `voltage_map` attribute to associate a voltage name with relative voltage values referenced by the cell-level `pg_pin` groups.

**Syntax**

```plaintext
voltage_map (voltage_name_id, voltage_value_float);
```

- `voltage_name`
  - Specifies a power supply.

- `voltage_value`
  - Specifies a voltage value.

**Example**

```plaintext
voltage_map (VDD1, 3.0);
```

---

**Group Statements**

Following are the descriptions of the technology library group statements.

---

**base_curves Group**

The `base_curves` group is a library-level group that contains the detailed description of normalized base curves.

**Syntax**

```plaintext
library (my_compact_ccs_lib) {
    ...
    base_curves (base_curves_name) {
    ...
    }
}
```

**Example**

```plaintext
library(my_lib) {
    ...
    base_curves (ctbctl) {
    ...
    }
}
```
Complex Attributes

base_curve_type
curve_x
curve_y

---

**base_curve_type Complex Attribute**

The `base_curve_type` attribute specifies the type of base curve. The valid values for `base_curve_type` are `ccs_half_curve` and `ccs_timing_half_curve`. The `ccs_half_curve` value allows you to model compact CCS power and compact CCS timing data within the same `base_curves` group. You must specify `ccs_half_curve` before specifying `ccs_timing_half_curve`.

**Syntax**

```
based_curve_type: enum (ccs_half_curve, ccs_timing_half_curve);
```

**Example**

```
base_curve_type: ccs_timing_half_curve;
```

---

**curve_x Complex Attribute**

Each base curve consists of one `curve_x` and one `curve_y`. The `curve_x` attribute should be defined before `curve_y` for clarity and easy implementation. Only one `curve_x` attribute can be specified for each `base_curves` group. The data array is the x-axis value of the normalized base curve. For a `ccs_timing_half_curve` base curve, the `curve_x` value must be between 0 and 1 and increase monotonically.

**Syntax**

```
curve_x: ("float", float);
```

**Example**

```
curve_x: ("0.2, 0.5, 0.8");
```

---

**curve_y Complex Attribute**

Each base curve consists of one `curve_x` and one `curve_y`. The `curve_x` attribute should be defined before `curve_y` for clarity and easy implementation. For compact CCS power, the valid region for `curve_y` is [-30, 30].

The `curve_y` attribute includes the following:

- The `curve_id` value, which specifies the base curve identifier.
- The data array, which is the y-axis value of the normalized base curve.
Syntax
curve_y (curve_id, "float..., float") ;

Example
curve_y (1, "0.8, 0.5, 0.2");

**compact_lut_template Group**
The **compact_lut_template** group is a lookup table template used for compact CCS timing and power modeling.

**Syntax**
library (my_compact_ccs_lib) {
    ...  
    compact_lut_template (template_name) {
        ...
    }
}

**Example**
library (my_lib) {
    ...  
    compact_lut_template (LTT3) {
        ...
    }
}

**Simple Attributes**
base_curves_group
variable_1
variable_2
variable_3

**Complex Attributes**
index_1
index_2
index_3

**base_curves_group Simple Attribute**
The **base_curves_group** attribute is required in the **compressed_lut_template** group. Its value is the specified **base_curves_group** name. The type of base curve in the **base_curves group** determines the **index_3** values when the **compact_lut_template** group is used.

**Syntax**
base_curves_group : base_curves_name ;
Example
base_curves_group : ctbcl ;

---

**variable_1 and variable_2 Simple Attributes**

The only valid values for the `variable_1` and `variable_2` attributes are `input_net_transition` and `total_output_net_capacitance`.

**Syntax**

```
variable_1 : input_net_transition | total_output_net_capacitance;
variable_2 : input_net_transition | total_output_net_capacitance;
```

**Example**

```
variable_1 : input_net_transition ;
variable_2 : total_output_net_capacitance ;
```

---

**variable_3 Simple Attribute**

The only legal string value for the `variable_3` attribute is `curve_parameters`.

**Syntax**

```
variable_3 : curve_parameters ;
```

**Example**

```
variable_3 : curve_parameters ;
```

---

**index_1 and index_2 Complex Attributes**

The `index_1` and `index_2` attributes are required. The `index_1` and `index_2` attributes define the `input_net_transition` and `total_output_net_capacitance` values. The `index` value for `input_net_transition` or `total_output_net_capacitance` is a floating-point number.

**Syntax**

```
index_1 ("float..., float") ;
index_2 ("float..., float") ;
```

**Example**

```
index_1 ("0.1, 0.2") ;
index_2 ("1.0, 2.0") ;
```
index_3 Complex Attribute

The string values in index_3 are determined by the base_curve_type value in the base_curve group. When ccs_timing_half_curve is the base_curve_type value, the following six string values (parameters) should be defined: init_current, peak_current, peak_voltage, peak_time, left_id, right_id; their order is not fixed.

More than six parameters are allowed if a more robust syntax is required or for circumstances where more parameters are needed to describe the original data.

Syntax

index_3 ("string..., string") ;

Example

index_3 ("init_current, peak_current, peak_voltage, peak_time, left_id, right_id") ;

char_config Group

The char_config group is a group of attributes including simple and complex attributes. These attributes represent library characterization configuration, and specify the settings to characterize the library. Use the char_config group syntax to apply an attribute value to a specific characterization model. You can specify multiple complex attributes in the char_config group. You can also specify a single complex attribute multiple times for different characterization models.

You can also define the char_config group within the cell, pin, and timing groups. However, when you specify the same attribute in multiple char_config groups at different levels, such as at the library, cell, pin, and timing levels, the attribute specified at the lower level gets priority over the ones specified at the higher levels. For example, the pin-level char_config group attributes have higher priority over the library-level char_config group attributes.

Syntax

library (library_name) {
    char_config() { /* characterization configuration attributes */
    }
    ... cell (cell_name) {
        char_config() { /* characterization configuration attributes */
        }
        ...
        pin (pin_name) {
            char_config() {
            ...
}
/* characterization configuration attributes */
} 
timing() {
    char_config() {
        /* characterization configuration attributes */
    }
} /* end of timing */
... /* end of pin */
... /* end of cell */
...

**Simple Attributes**

- internal_power_calculation
- three_state_disable_measurement_method
- three_state_disable_current_threshold_abs
- three_state_disable_current_threshold_rel
- three_state_disable_monitor_node
- three_state_cap_add_to_load_index
- ccs_timing_segment_voltage_tolerance_rel
- ccs_timing_delay_tolerance_rel
- ccs_timing_voltage_margin_tolerance_rel
- receiver capacitance1 voltage lower threshold pct rise
- receiver capacitance1 voltage upper threshold pct rise
- receiver capacitance1 voltage lower threshold pct fall
- receiver capacitance1 voltage upper threshold pct fall
- receiver capacitance2 voltage lower threshold pct rise
- receiver capacitance2 voltage upper threshold pct rise
- receiver capacitance2 voltage lower threshold pct fall
- receiver capacitance2 voltage upper threshold pct fall
- capacitance voltage lower threshold pct rise
- capacitance voltage lower threshold pct fall
- capacitance voltage upper threshold pct rise
- capacitance voltage upper threshold pct fall

**Complex Attributes**

- driver waveform
- driver waveform rise
- driver waveform fall
- input stimulus transition
- input stimulus interval
- unrelated output net capacitance
- default value selection method
- default value selection method rise
- default value selection method fall
- merge tolerance abs
- merge tolerance rel
- merge selection
Characterization Models

Table 1-2 lists the valid characterization models for the char_config group attributes.

**Table 1-2  Valid Characterization Models for the char_config Group**

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Default model. The all model has the lowest priority among the valid models for the char_config group. Any other model overrides the all model.</td>
</tr>
<tr>
<td>nldm</td>
<td>Default nonlinear delay model (NLDM)</td>
</tr>
<tr>
<td>nldm_delay</td>
<td>Specific NLDMs that have higher priority over the default NLDM</td>
</tr>
<tr>
<td>nldm_transition</td>
<td></td>
</tr>
<tr>
<td>capacitance</td>
<td>Capacitance model</td>
</tr>
<tr>
<td>constraint</td>
<td>Default constraint model</td>
</tr>
<tr>
<td>constraint_setup</td>
<td>Specific constraint models with higher priority over the default constraint model</td>
</tr>
<tr>
<td>constraint_hold</td>
<td></td>
</tr>
<tr>
<td>constraint_recovery</td>
<td></td>
</tr>
<tr>
<td>constraint_removal</td>
<td></td>
</tr>
<tr>
<td>constraint_skew</td>
<td></td>
</tr>
<tr>
<td>constraint_min_pulse_width</td>
<td></td>
</tr>
<tr>
<td>constraint_no_change</td>
<td></td>
</tr>
<tr>
<td>constraint_non_seq_setup</td>
<td></td>
</tr>
<tr>
<td>constraint_non_seq_hold</td>
<td></td>
</tr>
<tr>
<td>constraint_minimum_period</td>
<td></td>
</tr>
<tr>
<td>nlpm</td>
<td>Default nonlinear power model (NLPM)</td>
</tr>
<tr>
<td>nlpm_leakage</td>
<td>Specific NLPM with higher priority over the default NLPM</td>
</tr>
<tr>
<td>nlpm_input</td>
<td></td>
</tr>
<tr>
<td>nlpm_output</td>
<td></td>
</tr>
</tbody>
</table>

Selection Methods

Table 1-3 lists the valid selection methods used by the char_config group attributes.

**Table 1-3  Valid Selection Methods for the char_config Group**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>Selects a random value from the state-dependent data.</td>
</tr>
</tbody>
</table>
Table 1-3  Valid Selection Methods for the char_config Group (Continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>Selects the minimum value from the state-dependent data at each index point.</td>
</tr>
<tr>
<td>max</td>
<td>Selects the maximum value from the state-dependent data at each index point.</td>
</tr>
<tr>
<td>average</td>
<td>Selects an average value from the state-dependent data at each index point.</td>
</tr>
<tr>
<td>min_mid_table</td>
<td>Selects the minimum value from the state-dependent data in a lookup table. The minimum value is selected by comparing the middle value in the lookup table, with each of the table-values. Note: The middle value corresponds to an index value. If the number of index values is odd, then the middle value is taken as the median value. However, if the number of index values is even, then the smaller of the two values is selected as the middle value.</td>
</tr>
<tr>
<td>max_mid_table</td>
<td>Selects the maximum value from the state-dependent data in the lookup table. The maximum value is selected by comparing the middle value in the lookup table, with each of the table-values. Note: The middle value corresponds to an index value. If the number of index values is odd, then the middle value is taken as the median value. However, if the number of index values is even, then the smaller of the two values is selected as the middle value.</td>
</tr>
<tr>
<td>follow_delay</td>
<td>Selects the value from the state-dependent data for delay selection. This method is valid only for the nldm_transition characterization model, that is, the follow_delay method applies specifically to default transition-table selection and not any other default-value selection.</td>
</tr>
</tbody>
</table>

Example

```plaintext
library (library_test) {
  lu_table_template(waveform_template) {
    variable_1 : input_net_transition;
    variable_2 : normalized_voltage;
    index_1  ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7");
    index_2  ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }
  normalized_driver_waveform (waveform_template) {
    driver_waveform_name : input_driver;
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", \
            "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", \
            ... \
            "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  }
}
normalized_driver_waveform (waveform_template) {
    driver_waveform_name : input_driver_cell_test;
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", \
        "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", \
        ...
        "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
normalized_driver_waveform (waveform_template) {
    driver_waveform_name : input_driver_rise;
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", \
        "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", \
        ...
        "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
normalized_driver_waveform (waveform_template) {
    driver_waveform_name : input_driver_fall;
    values ("0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09", \
        "0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19", \
        ...
        "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
char_config() {
    /* library level default attributes*/
    driver_waveform(all, input_driver);
    input_stimulus_transition(all, 0.1);
    input_stimulus_interval(all, 100.0);
    unrelated_output_net_capacitance(all, 1.0);
    default_value_selection_method(all, any);
    merge_tolerance_abs(nldm, 0.1);
    merge_tolerance_abs(constraint, 0.1);
    merge_tolerance_abs(capacitance, 0.01);
    merge_tolerance_abs(nlpm, 0.05);
    merge_tolerance_rel(all, 2.0);
    merge_selection(all, max);
    internal_power_calculation : exclude_switching_rise;
    three_state_disable_measurement_method : current;
    three_state_disable_current_threshold_abs : 0.05;
    three_state_disable_current_threshold_rel : 2.0;
    three_state_disable_monitor_node : tri_monitor;
    three_state_cap_add_to_load_index : true;
    ccs_timing_segment_voltage_tolerance_rel : 1.0;
    ccs_timing_delay_tolerance_rel : 2.0;
    ccs_timing_voltage_margin_tolerance_rel : 1.0;
    receiver_capacitance1_voltage_lower_threshold_pct_rise : 20.0;
    receiver_capacitance1_voltage_upper_threshold_pct_rise : 50.0;
    receiver_capacitance1_voltage_lower_threshold_pct_fall : 50.0;
    receiver_capacitance1_voltage_upper_threshold_pct_fall : 80.0;
    receiver_capacitance2_voltage_lower_threshold_pct_rise : 20.0;
    receiver_capacitance2_voltage_upper_threshold_pct_rise : 50.0;
    receiver_capacitance2_voltage_lower_threshold_pct_fall : 50.0;
    receiver_capacitance2_voltage_upper_threshold_pct_fall : 80.0;
    capacitance_voltage_lower_threshold_pct_rise : 20.0;
    capacitance_voltage_lower_threshold_pct_fall : 50.0;
capacitance_voltage_upper_threshold_pct_rise : 50.0;
capacitance_voltage_upper_threshold_pct_fall : 80.0;
...}
...cell (cell_test) {
  char_config() {
    /* input driver for cell_test specifically */
    driver_waveform (all, input_driver_cell_test);
    /* specific default arc selection method for constraint */
    default_value_selection_method (constraint, max);
    default_value_selection_method_rise(nldm_transition, min);
    default_value_selection_method_fall(nldm_transition, max);
    ...
  }
  ...
  pin(pin1) {
    char_config() {
      driver_waveform_rise(delay, input_driver_rise);
    }
  ...
timing() {
  char_config() {
    driver_waveform_rise(constraint, input_driver_rise);
    driver_waveform_fall(constraint, input_driver_fall);
    /* specific ccs segmentation tolerance for this timing arc */
    ccs_timing_segment_voltage_tolerance_rel: 2.0;
  }
  } /* timing */
} /* pin */
} /* cell */
} /* lib */

**internal_power_calculation Simple Attribute**

To specify the characterization method to account for the switching energy in the
*internal_power* tables, set the *internal_power_calculation* attribute. Specify this attribute only if the *internal_power* group exists in the library.

**Syntax**

```
internal_power_calculation : exclude_switching_on_rise |
                          exclude_switching_on_rise_and_fall |
                          include_switching ;
```

**exclude_switching_on_rise**

The switching energy is deducted only from the *rise_power* table values.
exclude_switching_on_rise_and_fall

The switching energy is deducted from both the rise_power and fall_power table values.

include_switching

The switching energy is not deducted from the table values in the internal_power group.

Example

internal_power_calculation : exclude_switching_on_rise ;

three_state_disable_measurement_method Simple Attribute

The three_state_disable_measurement_method attribute specifies the method to identify the three-state condition of a pin. In a pin group, this attribute is valid only for a three-state pin. You must define this attribute if the library contains one or more three-state cells.

Syntax

three_state_disable_measurement_method : voltage | current ;

voltage

This method measures the voltage waveform to the gate input of the three-state stage.

current

This method measures the leakage current flowing through the pullup and pulldown resistors of the three-state stage.

Example

three_state_disable_measurement_method : current ;

three_state_disable_current_threshold_abs Simple Attribute

The three_state_disable_current_threshold_abs attribute specifies the absolute current threshold value to distinguish between the low- and high-impedance states of a three-state output pin. The unit of the absolute current threshold value is specified in the current_unit attribute of the library group.

In the pin group, this attribute is valid only for an inout pin. If you define both the three_state_disable_current_threshold_abs and three_state_disable_current_threshold_rel attributes, the pin enters the high-impedance state upon reaching either of the two threshold values.
three_state_disable_current_threshold_abs : float ;

Example
three_state_disable_current_threshold_abs : 0.05 ;

three_state_disable_current_threshold_rel Simple Attribute
The three_state_disable_current_threshold_rel attribute specifies the relative current threshold value to distinguish between the low- and high-impedance states of a three-state output pin. The relative current threshold value is specified as a percentage of the peak current, for example, 100.0 for 100 percent of the peak current.

In the pin group, this attribute is valid only for an inout pin. If you define both the three_state_disable_current_threshold_abs and three_state_disable_current_threshold_rel attributes, the pin enters the high-impedance state upon reaching either of the two threshold values.

Syntax
three_state_disable_current_threshold_rel : float ;

Example
three_state_disable_current_threshold_rel : 2.0 ;

three_state_disable_monitor_node Simple Attribute
The three_state_disable_monitor_node attribute specifies the internal node that is probed for the three-state voltage measurement method.

In the pin group, this attribute is valid only for an inout pin. You must define this attribute for the voltage method.

Syntax
three_state_disable_monitor_node : string ;

Example
three_state_disable_monitor_node : tri_monitor ;

three_state_cap_add_to_load_index Simple Attribute
The three_state_cap_add_to_load_index attribute specifies that the pin capacitance of a three-state pin is added to each index value of the total_output_net_capacitance variable. The valid values are true and false.

You must define this attribute.
Syntax
three_state_cap_add_to_load_index : true | false ;

Example
three_state_cap_add_to_load_index : true ;

ccs_timing_segment_voltage_tolerance_rel Simple Attribute

The ccs_timing_segment_voltage_tolerance_rel attribute specifies the maximum permissible voltage difference between the simulation waveform and the CCS waveform to select the CCS model point. The floating-point value is specified in percent, where 100.0 represents a 100 percent voltage difference.

You must define this attribute when the library includes a CCS model.

Syntax
ccs_timing_segment_voltage_tolerance_rel: float ;

Example
ccs_timing_segment_voltage_tolerance_rel: 1.0 ;

ccs_timing_delay_tolerance_rel Simple Attribute

The ccs_timing_delay_tolerance_rel attribute specifies the acceptable difference between the CCS waveform delay and the delay measured from simulation. The floating-point value is specified in percent, where 100.0 represents 100 percent acceptable difference.

You must define this attribute if the library includes a CCS model.

Syntax
ccs_timing_delay_tolerance_rel: float ;

Example
ccs_timing_delay_tolerance_rel: 2.0 ;

ccs_timing_voltage_margin_tolerance_rel Simple Attribute

The ccs_timing_voltage_margin_tolerance_rel attribute specifies the voltage tolerance for a signal to acquire the rail-voltage value. The floating-point value is specified as a percentage of the rail voltage, such as 96.0 for 96 percent of the rail voltage.

You must define this attribute if the library includes a CCS model.

Syntax
ccs_timing_voltage_margin_tolerance_rel: float ;
Example
ccs_timing_voltage_margin_tolerance_rel: 1.0 ;

CCS Receiver Capacitance Simple Attributes

The following CCS receiver capacitance attributes specify the current-integration limits, as a percentage of the voltage, to calculate the CCS receiver capacitances. The floating-point values of these attributes can vary from 0.0 to 100.0.

You must define all these attributes if the library includes a CCS model.

Syntax
receiver_capacitance1_voltage_lower_threshold_pct_rise : float ;
receiver_capacitance1_voltage_upper_threshold_pct_rise : float ;
receiver_capacitance1_voltage_lower_threshold_pct_fall : float ;
receiver_capacitance1_voltage_upper_threshold_pct_fall : float ;
receiver_capacitance2_voltage_lower_threshold_pct_rise : float ;
receiver_capacitance2_voltage_upper_threshold_pct_rise : float ;
receiver_capacitance2_voltage_lower_threshold_pct_fall : float ;
receiver_capacitance2_voltage_upper_threshold_pct_fall : float ;

Example
receiver_capacitance1_voltage_lower_threshold_pct_rise : 20.0 ;
receiver_capacitance1_voltage_upper_threshold_pct_rise : 50.0 ;
receiver_capacitance1_voltage_lower_threshold_pct_fall : 50.0 ;
receiver_capacitance1_voltage_upper_threshold_pct_fall : 80.0 ;
receiver_capacitance2_voltage_lower_threshold_pct_rise : 20.0 ;
receiver_capacitance2_voltage_upper_threshold_pct_rise : 50.0 ;
receiver_capacitance2_voltage_lower_threshold_pct_fall : 50.0 ;
receiver_capacitance2_voltage_upper_threshold_pct_fall : 80.0 ;

Input-Capacitance Measurement Simple Attributes

The following input-capacitance measurement attributes specify the corresponding threshold values for the rising and falling voltage waveforms, to calculate the NLDM input-pin capacitance. Each floating-point threshold value is specified as a percentage of the supply voltage, and can vary from 0.0 to 100.0.

You must define all these attributes.

Syntax
capacitance_voltage_lower_threshold_pct_rise : float ;
capacitance_voltage_lower_threshold_pct_fall : float ;
capacitance_voltage_upper_threshold_pct_rise : float ;
capacitance_voltage_upper_threshold_pct_fall : float ;

dcapacitance_voltage_lower_threshold_pct_rise : 20.0 ;
capacitance_voltage_lower_threshold_pct_fall : 20.0 ;
capacitance_voltage_upper_threshold_pct_rise : 50.0 ;
capacitance_voltage_upper_threshold_pct_fall : 50.0 ;
capacitance_voltage_lower_threshold_pct_fall : 50.0 ;
capacitance_voltage_upper_threshold_pct_rise : 50.0 ;
capacitance_voltage_upper_threshold_pct_rise : 80.0 ;

driver_waveform Complex Attribute
The driver_waveform attribute defines the driver waveform to characterize a specific characterization model.

You can define the driver_waveform attribute within the char_config group at the library, cell, pin, and timing levels. If you define the driver_waveform attribute within the char_config group at the library level, the library-level normalized_driver_waveform group is ignored when the driver_waveform_name attribute is not defined.

Syntax
driver_waveform ( char_model, waveform_name ) ;

Example
driver_waveform ( all, input_driver ) ;

driver_waveform_rise Complex Attribute
The driver_waveform_rise attribute defines a specific rising driver waveform to characterize a specific characterization model.

You can define the driver_waveform_rise attribute within the char_config group at the library, cell, pin, and timing levels. If you define the driver_waveform_rise attribute within the char_config group at the library level, the library-level normalized_driver_waveform group is ignored when the driver_waveform_name attribute is not defined.

Syntax
driver_waveform_rise ( char_model, waveform_name ) ;

Example
driver_waveform_rise ( all, input_driver ) ;

driver_waveform_fall Complex Attribute
The driver_waveform_fall attribute defines a specific falling driver waveform to characterize a specific characterization model.

You can define the driver_waveform_fall attribute within the char_config group at the library, cell, pin, and timing levels. If you define the driver_waveform_fall attribute within
the `char_config` group at the library level, the library-level `normalized_driver_waveform` group is ignored when the `driver_waveform_name` attribute is not defined.

**Syntax**

```
driver_waveform_fall (char_model, waveform_name) ;
```

**Example**

```
driver_waveform_fall ( all, input_driver ) ;
```

**input_stimulus_transition Complex Attribute**

The `input_stimulus_transition` attribute specifies the transition time for all the input-signal edges except the arc input pin’s last transition, during generation of the input stimulus for simulation.

The time units of the `input_stimulus_transition` attribute are specified by the library-level `time_unit` attribute.

You must define this attribute.

**Syntax**

```
input_stimulus_transition (char_model, float) ;
```

**Example**

```
input_stimulus_transition ( all, 0.1 ) ;
```

**input_stimulus_interval Complex Attribute**

The `input_stimulus_interval` attribute specifies the time-interval between the input-signal toggles to generate the input stimulus for a characterization cell. The time units of this attribute are specified by the library-level `time_unit` attribute.

You must define the `input_stimulus_interval` attribute.

**Syntax**

```
input_stimulus_interval (char_model, float) ;
```

**Example**

```
input_stimulus_interval ( all, 100.0 ) ;
```
unrelated_output_net_capacitance Complex Attribute

The unrelated_output_net_capacitance attribute specifies a load value for an output pin that is not a related output pin of the characterization model. The valid value is a floating-point number, and is defined by the library-level capacitive_load_unit attribute.

If you do not specify this attribute for the nldm_delay and nlpm_output characterization models, the unrelated output pins use the load value of the related output pin. However, you must specify this attribute for any other characterization model.

Syntax
unrelated_output_net_capacitance ( char_model, float ) ;

Example
unrelated_output_net_capacitance ( all, 1.0 ) ;

default_value_selection_method Complex Attribute

The default_value_selection_method attribute defines the method of selecting a default value for

- The delay arc from state-dependent delay arcs
- The constraint arc from state-dependent constraint arcs
- Pin-based minimum pulse-width constraints from simulated results with side pin combinations
- Internal power arcs from multiple state-dependent internal_power groups
- The cell_leakage_power attribute from the state-dependent values in leakage power models
- The input-pin capacitance from capacitance values for input-slew values used for timing characterization

Syntax
default_value_selection_method ( char_model, method ) ;

For valid values of the method argument, see Table 1-3 on page 1-30.

Example
default_value_selection_method ( all, any ) ;

default_value_selection_method_rise Complex Attribute

Use the default_value_selection_method_rise attribute when the selection method for rise is different from the selection method for fall.
You must define either the `default_value_selection_method` attribute, or the `default_value_selection_method_rise` and `default_value_selection_method_fall` attributes.

**Syntax**

```plaintext
default_value_selection_method_rise ( char_model, method ) ;
```

For valid values of the `method` argument, see Table 1-3 on page 1-30.

**Example**

```plaintext
default_value_selection_method_rise ( all, any ) ;
```

**default_value_selection_method_fall** Complex Attribute

Use the `default_value_selection_method_fall` attribute when the selection method for fall is different from the selection method for rise.

You must define either the `default_value_selection_method` attribute, or the `default_value_selection_method_rise` and `default_value_selection_method_fall` attributes.

**Syntax**

```plaintext
default_value_selection_method_fall ( char_model, method ) ;
```

For valid values of the `method` argument, see Table 1-3 on page 1-30.

**Example**

```plaintext
default_value_selection_method_fall ( all, any ) ;
```

**merge_tolerance_abs** Complex Attribute

The `merge_tolerance_abs` attribute specifies the absolute tolerance to merge arc simulation results. Specify the absolute tolerance value in the corresponding library unit.

If you specify both the `merge_tolerance_abs` and `merge_tolerance_rel` attributes, the results are merged if either or both the tolerance conditions are satisfied. If you do not specify any of these attributes, data including identical data is not merged.

**Syntax**

```plaintext
merge_tolerance_abs ( char_model, float ) ;
```

**Example**

```plaintext
merge_tolerance_abs ( constraint, 0.1 ) ;
```
merge_tolerance_rel Complex Attribute

The `merge_tolerance_rel` attribute specifies the relative tolerance to merge arc simulation results. Specify the relative tolerance value in percent, for example, 10.0 for 10 percent.

If you specify both the `merge_tolerance_abs` and `merge_tolerance_rel` attributes, the results are merged if either or both the tolerance conditions are satisfied. If you do not specify any of these attributes, data including identical data is not merged.

Syntax

```plaintext
merge_tolerance_rel ( char_model, float ) ;
```

Example

```plaintext
merge_tolerance_rel ( all, 2.0 ) ;
```

merge_selection Complex Attribute

The `merge_selection` attribute specifies the method to select the merged data. When multiple sets of state-dependent data are merged, the attribute selects a particular set of the state-dependent data to represent the merged data.

You must define the `merge_selection` attribute if you have defined the `merge_tolerance_abs` or `merge_tolerance_rel` attribute.

Syntax

```plaintext
merge_selection ( char_model, method ) ;
```

For valid values of the `method` argument, see Table 1-3 on page 1-30.

Example

```plaintext
merge_tolerance_rel ( all, max ) ;
```

For more information about the `char_config` group and group attributes, see the “Configuring Library Characterization Settings” chapter in the Liberty User Guide, Vol. 1.

dc_current_template Group

The `dc_current_template` group defines a template for specifying a two-dimensional `dc_current` table or a three-dimensional `vector` table.

Syntax

```plaintext
library (library_name) {
  dc_current_template (template_name) {
    ...
    template_description ...
  }
}
```
Simple Attributes

variable_1
variable_2
variable_3

Complex Attributes

index_1
index_2
index_3

variable_1, variable_2, and variable_3 Simple Attributes

For a two-dimensional dc_current table, the value you can assign to variable_1 is input_voltage, and the value you can assign to variable_2 is output_voltage.

For a three-dimensional vector table, the value you can assign to variable_1 is input_net_transition, and the value you can assign to variable_2 is output_net_transition. The value you can assign to variable_3 is time.

index_1, index_2, and index_3 Complex Attributes

Along with variable_1, variable_2, and variable_3, you must specify the index values.

index_1 ("float, ..., float")
index_2 ("float, ..., float")
index_3 ("float, ..., float")

Example

library (my_library) {

...  
dc_current_template (my_template) {

  variable_1 : input_net_transition ;
  variable_2 : output_net_transition ;
  variable_3 : time ;
  index_1 ("0.0, 0.0") ;
  index_2 ("0.0, 0.0") ;
  index_3 ("0.0, 0.0") ;

}
...  
}

em_lut_template Group

The em_lut_template group is defined at the library group level.

Syntax

library (namestring) {

  em_lut_template(namestring) {

    variable_1 : input_transition_time | total_output_net_capacitance ;
  

variable_2 : input_transition_time | total_output_net_capacitance;
index_1 : ("float, ..., float");
index_2 : ("float, ..., float");
}
}

The `em_lut_template` group creates a template of the index used by the `electromigration` group defined in the `pin` group level.

**variable_1, variable_2, and variable_3 Simple Attributes**

Following are the values that you can assign to the templates for electromigration tables. Use `variable_1` to assign values to one-dimensional tables; use `variable_2` to assign values for two-dimensional tables; and use `variable_3` to assign values for three-dimensional tables:

variable_1 : input_transition_time | total_output_net_capacitance;
variable_2 : input_transition_time | total_output_net_capacitance;

The value you assign to `variable_1` is determined by how the `index_1` complex attribute is measured, and the value you assign to `variable_2` is determined by how the `index_2` complex attribute is measured.

Assign `input_transition_time` to `variable_1` if the complex attribute `index_1` is measured with the input net transition time of the pin specified in the `related_pin` attribute or the pin associated with the `electromigration` group. Assign `total_output_net_capacitance` to `variable_1` if the complex attribute `index_1` is measured with the loading of the output net capacitance of the pin associated with the `em_max_toggle_rate` group.

Assign `input_transition_time` to `variable_2` if the complex attribute `index_2` is measured with the input net transition time of the pin specified in the `related_pin` or the `related_bus_pins` attribute or the pin associated with the `electromigration` group. Assign `total_output_net_capacitance` to `variable_2` if the complex attribute `index_2` is measured with the loading of the output net capacitance of the pin associated with the `electromigration` group.

**index_1 and index_2 Complex Attributes**

You can use these optional attributes to specify the first and second dimension breakpoints used to characterize cells for electromigration within the library.

**Syntax**

index_1 ("float, ..., float");
index_2 ("float, ..., float");
For index_1, the floating-point numbers that specify the breakpoints of the first dimension of the electromigration table used to characterize cells for electromigration within the library. For index_2, the floating-point numbers that specify the breakpoints for the second dimension of the electromigration table used to characterize cells for electromigration within the library.

You can overwrite the values entered for the em_lut_template group’s index_1 by entering values for the em_max_toggle_rate group’s index_1. You can overwrite the values entered for the em_lut_template group’s index_2 by entering values for the em_max_toggle_rate group’s index_2.

The following rules describe the relationship between variables and indexes:

- If you have variable_1, you can have only index_1.
- If you have variable_1 and variable_2, you can have index_1 and index_2.
- The value you enter for variable_1 (used for one-dimensional tables) is determined by how index_1 is measured. The value you enter for variable_2 (used for two-dimensional tables) is determined by how index_2 is measured.

Examples

```
em_lut_template (output_by_cap_and_trans) {
  variable_1 : total_output_net_capacitance;
  variable_2 : input_transition_time;
  index_1 ("0.0, 5.0, 20.0");
  index_2 ("0.0, 1.0, 2.0");
}
em_lut_template (input_by_trans) {
  variable_1 : input_transition_time;
  index_1 ("0.0, 1.0, 2.0");
}
```

fall_net_delay Group

The fall_net_delay group is defined at the library level, as shown here:

```
library (name) {
  fall_net_delay (name) {
    ... fall_net_delay description ...
  }
}
```

Complex Attributes

```text
index_1 ("float,...,float") ;
index_2 ("float,...,float") ;
values ("float,...,float","float,...,float");
```
The `rise_net_delay` and the `fall_net_delay` groups define, in the form of lookup tables, the values for rise and fall net delays. This indexing allows the library developer to model net delays as any function of `output_transition` and `rc_product`.

The net delay tables in one library have no effect on computations related to cells from other libraries. To overwrite the lookup table default index values, specify the new index values before the net delay values.

Example 1-4 shows an example of the `fall_net_delay` group.

**Example 1-4  fall_net_delay Group**

```plaintext
fall_net_delay (net_delay_table_template) {
  index_1 ("0, 1, 2");
  index_2 ("1, 0, 2");
  values ("0.00, 0.57", "0.10, 0.48");
}
```

---

**fall_transition_degradation Group**

The `fall_transition_degradation` group is defined at the library level, as shown here:

```plaintext
library (name) {
  fall_transition_degradation(name) {
    ... fall transition degradation description ...
  }
}
```

**Complex Attributes**

- `index_1 ("float, ..., float") ;`
- `index_2 ("float, ..., float") ;`
- `values ("float, ..., float", "float, ..., float") ;`

The `fall_transition_degradation` group and the `rise_transition_degradation` group describe, in the form of lookup tables, the transition degradation functions for rise and fall transitions. The lookup tables are indexed by the transition time at the net driver and the connect delay between the driver and a particular load. This indexing allows the library developer to model degraded transitions as any function of output-pin transition and connect delay between the driver and the load.

Transition degradation tables are used for indexing into any delay table in a library that has the `input_net_transition`, `constrained_pin_transition`, or `related_pin_transition` table parameters in the `lu_table_template` group.

The transition degradation tables in one library have no effect on computations related to cells from other libraries. Example 1-5 shows a `fall_transition_degradation` group.

**Example 1-5  fall_transition_degradation Group**

```plaintext
fall_transition_degradation(trans_deg) {
```
input_voltage Group

An input_voltage group is defined in the library group to designate a set of input voltage ranges for your cells.

Syntax

```
library (name_string) {
    input_voltage (name_string) {
        vil : float | expression;
        vih : float | expression;
        vimin : float | expression;
        vimax : float | expression;
    }
}
```

vil

The maximum input voltage for which the input to the core is guaranteed to be a logic 0.

vih

The minimum input voltage for which the input to the core is guaranteed to be a logic 1.

vimin

The minimum acceptable input voltage.

vimax

The maximum acceptable input voltage.

After you define an input_voltage group, you can use its name with the input_voltage simple attribute in a pin group of a cell. For example, you can define an input_voltage group with a set of high and low thresholds and minimum and maximum voltage levels and use the pin group to assign those ranges to the cell pin, as shown here.

Example

```
pin() {
    ...
    input_voltage : my_input_voltages ;
    ...
}
```
The value of each attribute is expressed as a floating-point number, an expression, or both. 

Table 1-4 lists the predefined variables that can be used in an expression.

Table 1-4 Voltage-Level Variables for the input_voltage Group

<table>
<thead>
<tr>
<th>CMOS or BiCMOS variable</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD</td>
<td>5V</td>
</tr>
<tr>
<td>VSS</td>
<td>0V</td>
</tr>
<tr>
<td>VCC</td>
<td>5V</td>
</tr>
</tbody>
</table>

The default values represent nominal operating conditions. These values fluctuate with the voltage range defined in the operating_conditions group.

All voltage values are in the units you define with the library group voltage_unit attribute.

Example 1-6 shows a collection of input_voltage groups.

Example 1-6 input_voltage Groups

```plaintext
input_voltage(CMOS) {  
    vil : 0.3 * VDD ;  
    vih : 0.7 * VDD ;  
    vimin : -0.5 ;  
    vimax : VDD + 0.5 ;  
}

input_voltage(TTL_5V) {  
    vil : 0.8 ;  
    vih : 2.0 ;  
    vimin : -0.5 ;  
    vimax : VDD + 0.5 ;  
}
```

fpga_isd Group

You can define one or more fpga_isd groups at the library level to specify the drive current, I/O voltages, and slew rates for FPGA parts and cells.

Note:

When you specify more than one fpga_isd group, you must also define the library-level default_fpga_isd attribute to specify which fpga_isd group is the default.
Syntax
library (namestring) {
  fpga_isd (fpga_isd_namestring ) {
    ... description ...
  }
}

Example
fpga_isd (part_cell_isd) {
  ... description ...
}

Simple Attributes
drive
io_type
slew

drive Simple Attribute
The drive attribute is optional and specifies the output current of the FPGA part or the FPGA cell.

Syntax
drive : value_id

value
  A string

Example
drive : 24 ;

io_type Simple Attribute
The io_type attribute is required and specifies the input or output voltage of the FPGA part or the FPGA cell.

Syntax
io_type : value_id

value
  A string

Example
io_type : LVTTL ;
slew Simple Attribute

The `slew` attribute is optional and specifies whether the slew of the FPGA part or the FPGA cell is FAST or SLOW.

Syntax

```
slew : value_id
```

value

Valid values are FAST and SLOW.

Example

```
slew : FAST ;
```

---

lu_table_template Group

Use the `lu_table_template` group to define templates of common information to use in lookup tables. Define the `lu_table_template` group at the library level, as shown:

Syntax

```
library (namestring) {
...
lu_table_template(namestring) {
    variable_1 : valueenum;
    variable_2 : valueenum;
    variable_3 : valueenum;
    index_1 ("float, ... , float");
    index_2 ("float, ... , float");
    index_3 ("float, ... , float");
}
}
```

Simple Attributes

```
variable_1
variable_2
variable_3
```

Complex Attributes

```
index_1
index_2
index_3
```
The `normalized_voltage` variable is specified under the `lu_table_template` table to describe a collection of waveforms under various input slew values. For a given input slew in `index_1` (for example, `index_1[0] = 1.0 ns`), the `index_2` values are a set of points that represent how the voltage rises from 0 to VDD in a rise arc, or from VDD to 0 in a fall arc.

**Note:**

The `normalized_voltage` variable can be used only with driver waveform syntax. For more information, see the “Driver Waveform Support” section in the “Timing Arcs” chapter in the *Liberty User Guide, Vol. 1*.

**Syntax**

```plaintext
lu_table_template (waveform_template) {
  variable_1 : input_net_transition;
  variable_2 : normalized_voltage;
  index_1 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7");
  index_2 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
```

**Rise Arc Example**

```plaintext
normalized_driver_waveform (waveform_template) {
  index_1 ("1.0"); /* Specifies the input net transition*/
  index_2 ("0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0"); /* Specifies the voltage normalized to VDD */
  values ("0, 0.2, 0.4, 0.6, 0.8, 0.9, 1.1"); /* Specifies the time when the voltage reaches the index_2 values*/
}
```

The `lu_table_template` table represents an input slew of 1.0 ns, when the voltage is 0%, 10%, 30%, 50%, 70%, 90% or 100% of VDD, and the time values are 0, 0.2, 0.4, 0.6, 0.8, 0.9, 1.1 (ns). Note that the time value can go beyond the corresponding input slew because a long tail might exist in the waveform before it reaches the final status.

**variable_1, variable_2, variable_3, and variable_4 Simple Attributes**

In Composite Current Source (CCS) Noise Tables:

Use lookup tables to create the lookup-table templates for the following groups within the `ccsn_first_stage` and `ccsn_last_stage` groups: the `dc_current` group and vectors of the `output_voltage_rise` group, `output_voltage_fall` group, `propagated_noise_low` group, and `propagated_noise_high` group.

You can assign the following values to the variables to specify the template used for the `dc_current` tables:

```plaintext
lu_table_template(dc_template_name) {
```
variable_1 : input_voltage;
variable_2 : output_voltage;
}

You can assign the following values to the variables to specify the template used for the vectors of the output_current_rise group and output_current_fall group.

lu_table_template(output_voltage_template_name) {

variable_1 : input_net_transition;
variable_2 : total_output_net_capacitance;
variable_3 : time;
}

You can assign the following values to the variables to specify the template used for the vector's of the propagated_noise_low and propagated_noise_high group.

lu_table_template(propagated_noise_template_name) {

variable_1 : input_noise_height;
variable_2 : input_noise_width;
variable_3 : total_output_net_capacitance;
variable_4 : time;

In Timing Delay Tables:

Following are the values that you can assign for variable_1, variable_2, and variable_3 to the templates for timing delay tables:

input_net_transition | total_output_net_capacitance |
output_net_length | output_net_wire_cap |
output_net_pin_cap |
related_out_total_output_net_capacitance |
related_out_output_net_length |
related_out_output_net_wire_cap |
related_out_output_net_pin_cap ;

The values that you can assign to the variables of a table specifying timing delay depend on whether the table is one-, two-, or three-dimensional.

In Constraint Tables:

You can assign the following values to the variable_1, variable_2, and variable_3 variables in the templates for constraint tables:

constrained_pin_transition | related_pin_transition | related_out_total_output_net_capacitance |
related_out_output_net_length |
related_out_output_net_wire_cap |
related_out_output_net_pin_cap ;

In Wire Delay Tables:
The following is the value set that you can assign for variable_1, variable_2, and variable_3 to the templates for wire delay tables:

fanout_number | fanout_pin_capacitance | driver_slew ;

The values that you can assign to the variables of a table specifying wire delay depends on whether the table is one-, two-, or three-dimensional.

In Net Delay Tables:

The following is the value set that you can assign for variable_1 and variable_2 to the templates for net delay tables:

output_transition | rc_product ;

The values that you can assign to the variables of a table specifying net delay depend on whether the table is one- or two-dimensional.

In Degradation Tables:

The following values apply only to templates for transition time degradation tables:

variable_1 : output_pin_transition | connect_delay ;
variable_2 : output_pin_transition | connect_delay ;

The cell degradation table template allows only one-dimensional tables:

variable_1 : input_net_transition

The following rules show the relationship between the variables and indexes:

- If you have variable_1, you must have index_1.
- If you have variable_1 and variable_2, you must have index_1 and index_2.
- If you have variable_1, variable_2, and variable_3, you must have index_1, index_2, and index_3.

CMOS Nonlinear Timing Model Examples

lu_table_template (constraint) {
  variable_1 : related_pin_transition ;
  variable_2 : related_out_total_output_net_capacitance ;
  variable_3 : constrained_pin_transition ;
  index_1 ("1.0, 1.5, 2.0") ;
  index_2 ("1.5, 1.0, 2.0") ;
  index_3 ("1.0, 2.0, 1.5") ;
}

lu_table_template (basic_template) {
  variable_1 : input_net_transition ;
  variable_2 : total_output_net_capacitance ;
  index_1 ("0.0, 0.5, 1.5, 2.0") ;
  index_2 ("0.0, 2.0, 4.0, 6.0") ;
}
maxcap_lut_template Group

The maxcap_lut_template group defines a template for specifying the maximum acceptable capacitance of an input or an output pin.

Syntax

```
library (namestring) {
  maxcap_lut_template (template_nameid) {
    ... template description ...
  }
}
```

Simple Attributes

- `variable_1`
- `variable_2`

Complex Attributes

- `index_1`
- `index_2`

variable_1 and variable_2 Simple Attributes

The value you can assign to `variable_1` is `frequency`. The value you can assign to `variable_2` is `input_transition_time`.

index_1 and index_2 Complex Attributes

Along with `variable_1` and `variable_2`, you must specify the index values.

```
index_1 ("float, ..., float");
index_2 ("float, ..., float");
```

Example

```
library (my_library) {
  ...
  maxcap_lut_template (my_template) {
    variable_1 : frequency;
    variable_2 : input_transition_time;
    index_1 ("100.0000, 200.0000");
    index_2 ("0.0, 0.0");
  }
  ...
}
```
maxtrans_lut_template Group

The `maxtrans_lut_template` group defines a template for specifying the maximum acceptable transition time of an input or an output pin.

Syntax

```plaintext
library (namestring) { 
  maxtrans_lut_template (template_nameid) { 
    ... template description ...
  }
}
```

Simple Attributes

`variable_1`

`variable_2`

Complex Attributes

`index_1`

`index_2`

`variable_1` and `variable_2` Simple Attributes

The value you can assign to `variable_1` is `frequency`. The value you can assign to `variable_2` is `input_transition_time`.

`index_1` and `index_2` Complex Attributes

Along with `variable_1` and `variable_2`, you must specify the index values.

```plaintext
index_1 ("float, ..., float") ;
index_2 ("float, ..., float") ;
```

Example

```plaintext
library (my_library) {
  ...
  maxtrans_lut_template (my_template) {
    variable_1 : frequency ;
    variable_2 : input_transition_time ;
    index_1 ("100.0000, 200.0000") ;
    index_2 ("0.0, 0.0") ;
    values ("0, 0.2, 0.4, 0.6, 0.8, 0.9, 1.1") ;
  }
  ...
}
```
normalized_driver_waveform Group

The library-level normalized_driver_waveform group represents a collection of driver waveforms under various input slew values. The index_1 specifies the input slew and index_2 specifies the normalized voltage.

Note that the slew index in the normalized_driver_waveform table is based on the slew derate and slew trip points of the library (global values). When applied on a pin or cell with different slew or slew derate, the new slew should be interpreted from the waveform.

Simple Attributes

**driver_waveform_name**

Complex Attributes

**index_1**
**index_2**
**values**

Syntax

```plaintext
normalized_driver_waveform(waveform_template_name) {
    driver_waveform_name : string; /* Specifies the name of the driver waveform table */
    index_1 ("float..., float"); /* Specifies input net transition */
    index_2 ("float..., float"); /* Specifies normalized voltage */
    values ( "float..., float", \ /* Specifies the time in library units */
            ..., \    
            "float..., float");
}
```

Example

```plaintext
normalized_driver_waveform (waveform_template) {
    index_1 ("1.0"); /* Specifies the Input net transition*/
    index_2 ("0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0"); /* Specifies the voltage normalized to VDD */
    values ("0, 0.2, 0.4, 0.6, 0.8, 0.9, 1.1"); /* Specifies the time when the voltage reaches the index_2 values*/
}
```

driver_waveform_name Simple Attribute

The driver_waveform_name string attribute differentiates the driver waveform table from other driver waveform tables when multiple tables are defined. The cell-specific, rise-specific, and fall-specific driver waveform usage modeling depend on this attribute.

The driver_waveform_name attribute is optional. You can define a driver waveform table without the attribute, but there can be only one table in a library, and that table is regarded as the default driver waveform table for all cells in the library. If more than one table is
defined without the attribute, the last table is used. The other tables are ignored and not stored in the library database file.

Syntax

driver_waveform_name : string ;

Example

normalized_driver_waveform (waveform_template) {
  driver_waveform_name : clock_driver ;
  index_1 ("0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75");
  index_2 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  values ("0.012, 0.03, 0.045, 0.06, 0.075, 0.090, 0.105, 0.13, 0.145, ...

operating_conditions Group

Use this group to define operating conditions; that is, process, voltage, and temperature. You define an operating_conditions group at the library-level, as shown here:

Syntax

library (namestring) {
  operating_conditions (namestring) {
    ... operating conditions description ...
  }
}

Simple Attributes

calc_mode : nameid ;
parameteri : float ;
process : float ;
process_label : "string" ;
temperature : float ;
tree_type : valueenum;
voltage : float ;

calc_mode Simple Attribute

An optional attribute, you can use calc_mode to specify an associated process mode.

Syntax

calc_mode : nameid ;

name

  The name of the associated process mode.
**parameter Simple Attribute**

Use this optional attribute to specify values for up to five user-defined variables.

**Syntax**
```
parameter {i} : valuefloat ; /* i = 1..5 */
```

- **value**
  A floating-point number representing the variable value.

**process Simple Attribute**

Use the `process` attribute to specify a scaling factor to account for variations in the outcome of the actual semiconductor manufacturing steps.

**Syntax**
```
process : valuefloat ;
```

- **value**
  A floating-point number from 0 through 100.

**process_label Simple Attribute**

Use the `process_label` attribute to specify the name of the current process.

**Syntax**
```
process_label : "process_name" ;
```

- **process_name**
  A string.

**temperature Simple Attribute**

Use the `temperature` attribute to specify the ambient temperature in which the design is to operate.

**Syntax**
```
temperature : valuefloat ;
```

- **value**
  A floating-point number representing the ambient temperature.

**tree_type Simple Attribute**

Use the `tree_type` attribute to specify the environment interconnect model.
Syntax

```
tree_type : value_enum;
```

```
value
```

Valid values are best_case_tree, balanced_tree, and worst_case_tree.

**voltage Simple Attribute**

Use the `voltage` attribute to specify the operating voltage of the design; typically 5 volts for a CMOS library.

Syntax

```
voltage : value_float;
```

```
value
```

A floating-point number from 0 through 1000, representing the absolute value of the actual voltage.

Example

```
operating_conditions (MPSS) {
  calc_mode : worst;
  process : 1.5;
  process_label : "ss";
  temperature : 70;
  voltage : 4.75;
  tree_type : worst_case_tree;
}
```

---

**output_current_template Group**

Use the `output_current_template` group to describe a table template for composite current source (CCS) modeling.

Syntax

```
library (name_string) {
  output_current_template(template_name_id) {
    variable_1 : value_enum;
    variable_2 : value_enum;
    variable_3 : value_enum;
    index_1 : ("float, ..., float");
    index_2 : ("float, ..., float");
    index_3 : ("float, ..., float");
  }
}
```
Simple Attributes

- variable_1
- variable_2
- variable_3

Complex Attributes

- index_1
- index_2
- index_3

**variable_1, variable_2, and variable_3 Simple Attributes**

The table template specifying composite current source (CCS) driver and receiver models can have three variables: variable_1, variable_2, and variable_3. The valid values for variable_1 and variable_2 are input_net_transition and total_output_net_capacitance. The only valid value for variable_3 is time.

**index_1, index_2, and index3 Complex Attributes**

Along with variable_1 and variable_2, you must specify the index values.

- index_1 ("float, ..., float")
- index_2 ("float, ..., float")

**Example**

```plaintext
library (my_library) {

... output_current_template (CCT) {
    variable_1 : input_transition;
    variable_2 : total_output_net_capacitance;
    variable_3 ; time;
    index_1 ("0.1, 0.2");
    index_2 ("1.0, 2.0");
    index_3 ("0.1, 0.2, 0.3, 0.4, 0.5");
} ...
}
```

**output_voltage Group**

You define an output_voltage group in the library group to designate a set of output voltage level ranges to drive output cells.

**Syntax**

```plaintext
library (name_string) {
    output_voltage(name_string) {
        vol : float | expression;
        voh : float | expression;
    }
}
```
The value for \( \text{vol}, \text{voh}, \text{vomin}, \text{and} \ \text{vomax} \) is a floating-point number or an expression. An expression allows you to define voltage levels as a percentage of VSS or VDD.

\( \text{vol} \)

The maximum output voltage generated to represent a logic 0.

\( \text{voh} \)

The minimum output voltage generated to represent a logic 1.

\( \text{vomin} \)

The minimum output voltage the pad can generate.

\( \text{vomax} \)

The maximum output voltage the pad can generate.

Table 1-5 lists the predefined variables you can use in an \text{output\_voltage} expression attribute. Separate variables are defined for CMOS and BiCMOS.

### Table 1-5 Voltage-Level Variables for the output\_voltage Group

<table>
<thead>
<tr>
<th>CMOS or BiCMOS variable</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD</td>
<td>5V</td>
</tr>
<tr>
<td>VSS</td>
<td>0V</td>
</tr>
<tr>
<td>VCC</td>
<td>5V</td>
</tr>
</tbody>
</table>

The default values represent nominal operating conditions. These values fluctuate with the voltage range defined in the \text{operating\_conditions} groups.

All voltage values are in the units you define with the \text{voltage\_unit} attribute within the \text{library} group. Example 1-7 shows an example of an \text{output\_voltage} group.

**Example 1-7 output\_voltage Group**

```plaintext
output\_voltage(GENERAL) {
  vol : 0.4 ;
}```
part Group

You use a part group to describe a specific FPGA device. Use multiple part groups to describe multiple devices.

Syntax

library (namestring) {
  part(namestring) {
    ...device description...
  }
  part(namestring) {
    ...device description...
  }
}

Simple Attributes

default_step_level
fpga_isd
num_blockrams
num_cols
num_ffs
num_luts
num_rows
pin_count

Complex Attributes

max_count
valid_speed_grade
valid_step_level

Group

speed_grade

default_step_level Simple Attribute

Use the default_step_level attribute to specify one of the valid step levels as the default for the FPGA device. You specify valid step levels with the valid_step_levels complex attribute.

Syntax

default_step_level : "name" ;
"name"

An alphanumeric string identifier, enclosed within double quotation marks, representing the default step level for the device.

Example
default_step_level ("STEP0") ;

fpga_isd Simple Attribute

Use this optional attribute to reference the drive, io_type, and slew information contained in a library-level fpga_isd group.

Syntax
fpga_isd : fpga_isd_nameid ;

default_step_level

fpga_isd_name

The name of a library-level fpga_isd group.

Example
fpga_isd : part_cell_isd ;

num_blockrams Simple Attribute

Use the num_blockrams attribute to specify the number of block select RAMs on the FPGA device.

Syntax
num_blockrams : value_int ;

value

An integer representing the number of block select RAMs on the device.

Example
num_blockrams : 10 ;

num_cols Simple Attribute

Use the num_cols attribute to specify the number of logic block columns on the FPGA device.

Syntax
num_cols : value_int ;
value

An integer representing the number of logic blocks on the FPGA device.

Example

num_cols : 30 ;

num_ffs Simple Attribute

Use the num_ffs attribute to specify the number of flip-flops on the device.

Syntax

num_ffs : value_int ;

value

An integer representing the number of flip-flops on the FPGA device.

Example

num_ffs : 2760 ;

num_luts Simple Attribute

Use the num_luts attribute to specify the total number of lookup tables available for the FPGA device. The num_luts value is used to determine the total number of slices that make up all the configurable logic blocks (CLBs) of the FPGA device, as shown in the following equation.

\[
\text{Total number of CLB slices} = \frac{\text{Total number of lookup tables (num_luts)}}{2}
\]

Syntax

num_luts : value_int ;

value

An integer representing the number of lookup tables on the FPGA device.

Example

num_luts : 100 ;

num_rows Simple Attribute

Use the num_rows attribute to specify the number of logic block rows on the FPGA device.

Syntax

num_rows : value_int ;
value

An integer representing the number of block rows on the FPGA device.

Example

num_rows : 20 ;

pin_count Simple Attribute

Use the pin_count attribute to specify the number of pins on the device.

Syntax

pin_count : value_int ;

value

An integer representing the number of pins on the FPGA device.

Example

pin_count : 94 ;

max_count Complex Attribute

Use the max_count attribute to specify the resource constraints for the FPGA device.

Syntax

max_count (resource_nameid, valueint ) ;

resource_name

The name of the resource being constrained.

value

An integer representing the maximum constraint of the resource.

Example

max_count (BUGFGTS, 4) ;

valid_speed_grade Complex Attribute

Use the valid_speed_grade attribute to specify the various speed grades for the FPGA device.

Syntax

valid_speed_grade ("name_1", "name_2", ..."name_n" ) ;
A list of alphanumeric string identifiers, each enclosed within double quotation marks, represents the various speed grades for the device. Each identifier corresponds to an operating condition under which the library is characterized and under which the device is used.

**Example**
```
valid_speed_grade ("-6", "-5", "-4") ;
```

### valid_step_levels Complex Attribute

Use the `valid_step_levels` attribute to specify the various step levels for the FPGA device.

**Syntax**
```
valid_step_levels ("name_1id", "name_2id", ..."name_nid") ;
```

A list of alphanumeric string identifiers, each enclosed within double quotation marks, representing various step levels for the device. Each identifier corresponds to an operating condition under which the library is characterized and under which the device is used.

**Example**
```
valid_step_levels ("STEP0", "STEP1", "STEP2") ;
```

### speed_grade Group

The `speed_grade` group associates a valid speed grade with a valid step level.

**Syntax**
```
part(name_string) {
  ...
  speed_grade (name_string) {
    ...step_level_description...
  }
}
```

**name**

Specifies one of the valid speed grades listed in the `valid_speed_grade` attribute.

**Simple Attribute**
```
fpga_isd
```
Complex Attribute
step_level

Example
speed_grade ( ) {
  ...
}

fpga_isd Simple Attribute
Use this optional attribute to reference the drive, io_type, and slew information contained in a library-level fpga_isd group.

Syntax
fpga_isd : fpga_isd_name_id ;

fpga_isd_name
  The name of a library-level fpga_isd group.

Example
fpga_isd : part_cell_isd ;

step_level Complex Attribute
Use the step_level attribute to specify one of the valid step levels listed in the valid_step_level attribute.

Syntax
step_level (name_string) ;

name
  The alphanumeric identifier for a valid step level.

Example
step_level ( ) ;

pg_current_template Group
In the composite current source (CCS) power library format, instantaneous power data is specified as 1- to n-dimensional tables of current waveforms in the pg_current_template group. This library-level group creates templates of common information that power and ground current vectors use.
Syntax
library (name_string) {
  pg_current_template (template_name_id) {
    ... template description ...
  }
}

Simple Attributes
variable_1
variable_2
variable_3
variable_4

Complex Attributes
index_1
index_2
index_3
index_4

variable_1, variable_2, variable_3, and variable_4 Simple Attributes

The variable values can be input_net_transition, total_output_net_capacitance, and time. The last variable must be time and is required. The group can contain none or at most one input_net_transition variable. It can contain none or up to two total_output_net_capacitance variables.

index_1, index_2, index_3, and index_4 Complex Attributes

The index values are optional.

index_1 ("float, ...") ;
index_2 ("float, ...") ;
index_3 ("float, ...") ;
index_4 ("float, ...") ;

Example
library (my_library) {
  ...
  pg_current_template (my_template) {
    variable_1 : input_net_transition ;
    variable_2 : total_output_net_capacitance ;
    variable_3 : total_output_net_capacitance ;
    variable_4 : time ;
    index_1 ("100.0000, 200.0000") ;
    index_2 ("0.0, 0.0") ;
    index_3 ("0.0, 0.0") ;
    index_4 ("0.0, 0.0") ;
  }
  ...
}
power_lut_template Group

The power_lut_template group is defined within the library group, as shown here:

Syntax

```plaintext
library (name) {
    power_lut_template (template_name) {
        variable_1 : input_transition_time |
        total_output_net_capacitance |
        equal_or_opposite_output_net_capacitance;
        variable_2 : input_transition_time |
        total_output_net_capacitance |
        equal_or_opposite_output_net_capacitance;
        variable_3 : input_transition_time |
        total_output_net_capacitance |
        equal_or_opposite_output_net_capacitance;
        index_1 ("float, ... , float");
        index_2 ("float, ... , float");
        index_3 ("float, ... , float");
    }
}
```

Simple Attributes

- variable_1
- variable_2
- variable_3

Complex Attributes

- index_1
- index_2
- index_3

Group
domain

The power_lut_template group creates a template of the index used by the internal_power group (defined in a pin group within a cell).

The name of the template (template_name) is a name you choose that can be used later for reference.

Note:

A power_lut_template with the name scalar is predefined; its size is 1. You can refer to it by entering scalar as the name of a fall_power group, power group, or rise_power group within the internal_power group (defined in the pin group).
variable_1, variable_2, and variable_3 Simple Attributes

The variable_1 attribute in the power_lut_template group specifies the first dimensional variable used by the library developer to characterize cells in the library for internal power.

The variable_2 attribute in the power_lut_template group specifies the second dimensional variable the library developer uses to characterize cells in the library for internal power.

The variable_3 attribute in the power_lut_template group specifies the third dimensional variable the library developer uses to characterize cells in the library for internal power.

If the index_1 attribute is measured with the loading of the output net capacitance of the pin specified in the pin group that contains the internal_power group (defined in a cell group), the value for variable_1 is equal_or_opposite_output_net_capacitance.

If the index_1 attribute is measured with the input transition time of the pin specified in the pin group or the related_pin attribute of the internal_power group, the value for variable_1 is input_transition_time.

If the index_2 attribute is measured with the loading of the output net capacitance of the pin specified in the pin group that contains the internal_power group (defined in a cell group), the value for variable_2 is equal_or_opposite_output_net_capacitance.

If the index_2 attribute is measured with the input transition time of the pin specified in the pin group or the related_pin attribute of the internal_power group, the value for variable_2 is input_transition_time.

If the index_3 attribute is measured with the loading of the output net capacitance of the pin specified in the pin group that contains the internal_power group (defined in a cell group), the value for variable_3 is equal_or_opposite_output_net_capacitance.

If the index_3 attribute is measured with the input transition time of the pin specified in the pin group or the related_pin attribute of the internal_power group, the value for variable_3 is input_transition_time.

index_1, index_2, and index_3 Complex Attributes

The index_1 complex attribute in the power_lut_template group specifies the breakpoints of the first dimension used to characterize cells for internal power within the library. The values specified in this attribute must be in a monotonically increasing order. You can overwrite the index_1 attribute by providing the same attribute in the fall_power group, power group, or rise_power group within the internal_power group (defined in the pin group). The index_1 attribute is required in the power_lut_template group.

The index_2 complex attribute in the power_lut_template group specifies the breakpoints of the second dimension used to characterize cells for internal power within the library. You can overwrite the index_2 attribute by providing the same attribute in the fall_power group, power group, or rise_power group within the internal_power group (defined in the
pin group). The \texttt{index} \_\texttt{2} attribute is required in the \texttt{power} \_\texttt{lut} \_\texttt{template} group if the \texttt{variable} \_\texttt{2} attribute is present.

The \texttt{index} \_\texttt{3} complex attribute in the \texttt{power} \_\texttt{lut} \_\texttt{template} group specifies the breakpoints of the third dimension used to characterize cells for internal power within the library. You can overwrite the \texttt{index} \_\texttt{3} attribute in the \texttt{internal} \_\texttt{power} group by providing the same attribute in the \texttt{fall} \_\texttt{power} group, \texttt{power} group, or \texttt{rise} \_\texttt{power} group within the \texttt{internal} \_\texttt{power} group (defined in the pin group). The \texttt{index} \_\texttt{3} attribute is required in the \texttt{power} \_\texttt{lut} \_\texttt{template} group if the \texttt{variable} \_\texttt{3} attribute is present.

Example 1-8 shows four \texttt{power} \_\texttt{lut} \_\texttt{template} groups.

\textbf{Example 1-8 \ Four \texttt{power} \_\texttt{lut} \_\texttt{template} Groups}

\begin{verbatim}
  power_lut_template (output_by_cap) { 
    variable_1 : total_output_net_capacitance ;
    index_1 ("0.0, 5.0, 20.0") ;
  }
  power_lut_template (output_by_cap_and_trans) { 
    variable_1 : total_output_net_capacitance ;
    variable_2 : input_transition_time ;
    index_1 ("0.0, 5.0, 20.0") ;
    index_2 ("0.1, 1.0, 5.0") ;
  }
  power_lut_template (input_by_trans) { 
    variable_1 : input_transition_time ;
    index_1 ("0.0, 1.0, 5.0") ;
  }
  power_lut_template (output_by_cap2_and_trans) { 
    variable_1 : total_output_net_capacitance ;
    variable_2 : input_transition_time ;
    variable_3 : equal_or_opposite_output_net_capacitance ;
    index_1 ("0.0, 5.0, 20.0") ;
    index_2 ("0.1, 1.0, 5.0") ;
    index_3 ("0.1, 0.5, 1.0") ;
  }
\end{verbatim}

\textbf{rise} \_\textbf{net} \_\textbf{delay Group}

The \texttt{rise} \_\texttt{net} \_\texttt{delay} group is defined at the library level, as shown here:

\begin{verbatim}
library (name) { 
  rise_net_delay(name) { 
    ... rise net delay description ...
  }
  fall_net_delay (name){ 
    ... fall net delay description ...
  }
}
\end{verbatim}

\textbf{Complex Attributes}

\begin{verbatim}
index_1 ("float,...,float") ;
\end{verbatim}
index_2 ("float,...,float")
values ("float,...,float","float,...,float");

The rise_net_delay and the fall_net_delay groups define, in the form of lookup tables, the values for rise and fall net delays. This indexing allows the library developer to model net delays as any function of output_transition and rc_product.

The net delay tables in one library have no effect on computations related to cells from other libraries.

To overwrite the lookup table default index values, specify the new index values before the net delay values.

Example 1-9 shows an example of the rise_net_delay group.

Example 1-9  rise_net_delay Group

```plaintext
rise_net_delay (net_delay_template_table) {
  index_1 ("0, 1, 2");
  index_2 ("1, 0, 2");
  values ("0.00, 0.21", "0.11, 0.23");
}
```

See also fall_net_delay Group.

rise_transition_degradation Group

The rise_transition_degradation group is defined at the library level, as shown here:

```plaintext
library (name) {
  rise_transition_degradation(name) {
    ... rise transition degradation description ...
  }
}
```

Complex Attributes

```plaintext
index_1 ("float, ..., float")
index_2 ("float, ..., float")
values ("float, ..., float", "float, ...,float")
```

The rise_transition_degradation group and the fall_transition_degradation group describe, in the form of lookup tables, the transition degradation functions for rise and fall transitions. The lookup tables are indexed by the transition time at the net driver and the connect delay between the driver and a particular load. This indexing allows the library developer to model degraded transitions as any function of output-pin transition and connect delay between the driver and the load.
Transition degradation tables are used for indexing into any delay table in a library that has the \texttt{input\_net\_transition}, \texttt{constrained\_pin\_transition}, or \texttt{related\_pin\_transition} table parameters in the \texttt{lu\_table\_template} group.

The transition degradation tables in one library have no effect on computations related to cells from other libraries. Example 1-10 shows an example of the \texttt{rise\_transition\_degradation} group.

\begin{verbatim}
Example 1-10  rise_transition_degradation Group

rise_transition_degradation (trans_deg) {
    index_1 ("0, 1, 2") ;
    index_2 ("1, 0, 2") ;
    values ("0.0, 0.6", "1.0, 1.6") ;
}
\end{verbatim}

See also “\texttt{fall\_transition\_degradation Group}” on page 1-46.

\section*{sensitization Group}

The \texttt{sensitization} group defined at the library level describes the complete state patterns for a specific list of pins (defined by the \texttt{pin\_names} attribute) that are referenced and instantiated as stimuli in the timing arc.

Vector attributes in the group define all possible pin states used as stimuli. Actual stimulus waveforms can be described by a combination of these vectors. Multiple \texttt{sensitization} groups are allowed in a library. Each \texttt{sensitization} group can be referenced by multiple cells, and each cell can make reference to multiple \texttt{sensitization} groups.

\begin{verbatim}
Syntax

library(library_name) {
    ...
    sensitization (sensitization_group_name) {
        ...
    }
}
\end{verbatim}

\subsection*{Complex Attributes}

\begin{verbatim}
pin_names
vector
\end{verbatim}

\begin{verbatim}
pin_names Complex Attribute

The \texttt{pin\_names} attribute specified at the library level defines a default list of pin names. All vectors in this \texttt{sensitization} group are the exhaustive list of all possible transitions of the input pins and their subsequent output response.
\end{verbatim}
The pin_names attribute is required, and it must be declared in the sensitization group before all vector declarations.

Syntax
pin_names (string..., string);

Example
pin_names (IN1, IN2, OUT);

### vector Complex Attribute

Similar to the pin_names attribute, the vector attribute describes a transition pattern for the specified pins. The stimulus is described by an ordered list of vectors.

The arguments for the vector attribute are as follows:

vector id

The vector id argument is an identifier to the vector string (a number tag that defines the list of possible sensitization combinations in a cell). The vector id value must be an integer greater than or equal to zero and unique among all vectors in the current sensitization group. It is recommended that you start numbering from 0 or 1.

vector string

The vector string argument represents a pin transition state. The string consists of the following transition status values: 0, 1, X, and Z where each character is separated by a space. The number of elements in the vector string must equal the number of arguments in pin_names.

The vector attribute can also be declared as:

vector (positive_integer, "{0\|1\|X\|Z} [0\|1\|X\|Z]\ldots");

Syntax
vector (integer, string);

Example
sensitization(sensitization_nand2) {
  pin_names ( IN1, IN2, OUT1 );
  vector ( 1, "0 0 1" );
  vector ( 2, "0 1 1" );
  vector ( 3, "1 0 1" );
  vector ( 4, "1 1 0" );
}
**timing Group**

A timing group is defined in a bundle, a bus, or a pin group within a cell. The timing group can be used to identify the name or names of multiple timing arcs. A timing group identifies multiple timing arcs, by identifying a timing arc in a pin group that has more than one related pin or when the timing arc is part of a bundle or a bus.

The following syntax shows a timing group in a pin group within a cell group.

**Syntax**

```plaintext
library (name_string) {
  cell (name) {
    pin (name) {
      timing (name | name_list) {
        ... timing description ...
      }
    }
  }
}
```

**type Group**

If your library contains bused pins, you must define type groups and define the structural constraints of each bus type in the library. The type group is defined at the library group level, as shown here:

**Syntax**

```plaintext
library (name_string) {
  type (name) {
    ... type description ...
  }
}
```

**name**

Identifies the bus type.

**Simple Attributes**

```plaintext
base_type : base;
bit_from : integer;
bit_to : integer;
bit_width : integer;
data_type : data;
downto : true | false;
```
base_type : base;

Only the array base type is supported.

bit_from : integer;

Indicates the member number assigned to the most significant bit (MSB) of successive array members. The default is 0.

bit_to : integer;

Indicates the member number assigned to the least significant bit (LSB) of successive array members. The default is 0.

bit_width : integer;

Designates the number of bus members. The default is 1.

data_type : data;

Indicates that only the bit data type is supported.

downto : true | false;

A true value indicates that member number assignment is from high to low. A false value indicates that member number assignment is from low to high.

Example 1-11 illustrates a type group statement.

Example 1-11 type Group Description

```plaintext
type (BUS4) {
  base_type : array;
  bit_from : 0;
  bit_to : 3;
  bit_width : 4;
  data_type : bit;
  downto : false;
}
```

It is not necessary to use all attributes.

Example 1-12 Alternative type Group Descriptions

```plaintext
type (BUS4) {
  base_type : array;
  data_type : bit;
  bit_width : 4;
  bit_from : 0;
  bit_to : 3;
}

type (BUS4) {
  base_type : array;
```
data_type : bit 
bit_width : 4 
bit_from : 3 
downto : true 
}

After you define a type group, you can use it in a bus group to describe bused pins.

user_parameters Group

Use the user_parameters group to specify default values for up to five user-defined process variables.

Define a user_parameters group in a library group as follows.

**Syntax**

```plaintext
type
user_parameters () {
...
parameter descriptions...
}
}
```

**Simple Attributes**

```plaintext
parameter
```

**parameter/ Simple Attributes**

Use each generic attribute to specify a default value for a user-defined process variable. You can specify up to five variables.

**Syntax**

```plaintext
parameteri : value float ;
```

**value**

A floating-point number representing a variable value.

**Example**

```plaintext
parameter1: 0.5 ;
```

voltage_state_range_list Group

The voltage_state_range_list group defines the voltage range of all the states for a specified power rail. The name of the group (voltage_name) is a power rail name, which must also be defined in the same library using the voltage_map attribute. Each voltage_name can have only one corresponding voltage_state_range_list group.
Syntax
library (library_name) {
  ...
  voltage_state_range_list (voltage_name) {
    /* list of voltage state range */
    ...
  } /* end voltage_state_range_list group */
}

voltage_state_range Attribute

The voltage_state_range attribute defines the corner-independent operating voltages for the state, pg_state, of the power rail. Specify this attribute in the voltage_state_range_list group.

Syntax
voltage_state_range_list (voltage_name) {
  voltage_state_range(pg_state, nom_voltage);
  voltage_state_range(pg_state, min_voltage, max_voltage);
  voltage_state_range(pg_state, min_voltage, nom_voltage, max_voltage);
}

• pg_state is the state name of the power rail
• min_voltage and max_voltage are floating-point numbers for the minimum and maximum operating voltage of the state.
• nom_voltage is a floating-point number and the default operating voltage that applies to all designs.

Example
voltage_state_range_list(VDD) {
  voltage_state_range(normal, 0.81, 0.9, 0.99);
  voltage_state_range(under_drive, 0.665, 0.77);
  voltage_state_range(on, 0.7);
}

wire_load Group

A wire_load group is defined in a library group, as follows.

Syntax
library (name) {
  wire_load (name) {
    ... wire load description ...
  }
}
Simple Attributes
area : float;
capacitance : float;
resistance : float;
slope : float;

Complex Attribute
fanout_length

area Simple Attribute
Use this attribute to specify area per unit length of interconnect wire.

Syntax
area : valuefloat;

value
A floating-point number representing the area.

Example
area: 0.5;

capacitance Simple Attribute
Use this attribute to specify capacitance per unit length of interconnect wire.

Syntax
capacitance : valuefloat;

value
A floating-point number representing the capacitance.

Example
capacitance : 1.2;

resistance Simple Attribute
Use this attribute to specify wire resistance per unit length of interconnect wire.

Syntax
resistance : valuefloat;

value
A floating-point number representing the resistance.
Example
resistance : 0.001 ;

slope Simple Attribute
Use this attribute to characterize linear fanout length behavior beyond the scope of the longest length specified in the fanout_length attribute.

Syntax
slope : valuefloat ;

value
A floating-point number representing the slope in units per fanout.

Example
slope : 0.186

fanout_length Complex Attribute
Use this attribute to define values for fanout and length when you create the wire load manually.

Syntax
fanout_length ( fanoutint,lengthfloat,average_capacitancefloat,\nstandard_deviationfloat,number_of_netsint ) ;

fanout
An integer representing the total number of pins, minus one, on the net driven by the given output.

length
A floating-point number representing the estimated amount of metal that is statistically found on a network with the given number of pins.

Examples
library (example)
...
wire_load (small) {
  area : 0.0 ;
  capacitance : 1.0 ;
  resistance : 0.0 ;
  slope : 0.0 ;
  fanout_length (1,1.68) ;
}
}
library (example) {
...
wire_load ("90x90") {
  lu_table_template (wire_delay_table_template) {
    variable_1 : fanout_number;
    variable_2 : fanout_pin_capacitance;
    variable_3 : driver_slew;
    index_1("0.12,3.4");
    index_2("0.12,4.24");
    index_3("0.1,2.7,3.12");
  }
}
}

wire_load_selection Group

A wire_load_selection group is defined in a library group, as follows.

Syntax
library (name) {
  wire_load_selection (name) {
    ... wire load selection criteria ...
  }
}

Complex Attribute
wire_load_from_area (float, float, string);

Example
wire_load_selection (normal) {
  wire_load_from_area (100, 200, average);
}

wire_load_table Group

A wire_load_table group is defined in a library group, as follows.

Syntax
library (name) {
  wire_load_table (name) {
    ... wire_load_table description ...
  }
}
**Complex Attributes**

- `fanout_area (integer, float)` ;
- `fanout_capacitance (integer, float)` ;
- `fanout_length (integer, float)` ;
- `fanout_resistance (integer, float)` ;

**Example**

```plaintext
library (wlut) {
    wire_load_table ("05x05") {
        fanout_area (1, 0.2) ;
        fanout_capacitance (1, 0.15);
        fanout_length (1, 0.2) ;
        fanout_resistance (1, 0.17) ;
    }
}
```
Every cell in a library has a cell description (a cell group) within the library group. A cell group can contain simple and complex attributes and other group statements. Every model in a library also has a model description (a model group) within the library group. A model group can include the same simple and complex attributes and group statements as a cell group, plus two new attributes that can be used only in the model group.

This chapter describes the attributes and groups that can be included within cell and model groups, with the exception of the pin group, which is described in Chapter 3, “pin Group Description and Syntax.”

This chapter is organized as follows:

- **cell Group**
  - Attributes and values
  - Simple attributes
  - Complex attributes
  - Group statements

- **model Group**
  - Attributes and values

Within each division, the attributes and group statements are presented alphabetically.
A cell group is defined within a library group, as shown here:

```plaintext
library (namestring) {
    cell (namestring) {
        ... cell description ...
    }
}
```

Attributes and Values

Example 2-1 lists alphabetically all the attributes and groups that you can define within a cell group.

Example 2-1 Attributes and Values for a cell Group

```plaintext
/* Simple Attributes for cell group */

always_on : true | false ;
antenna_diode_type : power | ground | power_and_ground ;
area : float ;
avuxiliary_pad_cell : true | false ;
base_name : cell_base_namestring ;
bus_naming_style : "string" ;
cell_footprint : footprint_typestring ;
cell_leakage_power : float ;
clock_gating_integrated_cell : string_value ;
contention_condition: "Boolean expression" ;
dont_fault : sa0 | sa1 | sa01 ;
dont_touch : true | false ;
dont_use : true | false ;
driver_type : nameid ;
em_temp_degradation_factor : valuefloat ;
interface_timing : true | false ;
io_type : nameid ;
is_clock_gating_cell : true | false ;
is_clock_isolation_cell : true | false ;
is_isolation_cell : true | false ;
is_level_shifter : true | false ;
is_macro_cell : true | false ;
is_soi : true | false ;
map_only : true | false ;
pad_cell : true | false ;
pad_type : clock ;
power_cell_type : ;
preference : true | false ;
retention_cell : retention_cell_style ;
```
single_bit_degenerate : string;
/* black box, bus, and bundle cells only*/
slew_type : nameid;
timing_model_type : "string";
use_for_size_only : true | false;

/* Complex Attributes for cell Group */

pin_equal ("name_list_string") ;
pin_opposite ("name_list1_string", "name_list2_string") ;
resource_usage (resource_nameid, number_of_resourcesid);

/* Group Statements for cell Group */

bundle (name_string) { }
bus (name_string) { }
clear_condition () {} 
clock_condition () {} 
dynamic_current () {} 
ff (variable1_string, variable2_string) { }
ff_bank (variable1_string, variable2_string, bits_integer) { }
generated_clock () {}
intrinsic_parasitic () {} 
latch (variable1_string, variable2_string) { }
latch_bank (variable1_string, variable2_string, bits_integer) { }
leakage_current () {}
leakage_power () { }
lut (name_string) { }
mode_definition () {} 
pin (name_string | name_list_string) { }
preset_condition () {} 
retention_condition () {} 
statetable ("input node names", "internal node names") { }
test_cell () {} 
type (name_string) { }

Descriptions of the attributes and group statements follow.
Simple Attributes

This section lists, alphabetically, the simple attributes for the cell and model groups.

always_on Simple Attribute
The always_on simple attribute models always-on cells or signal pins. Specify the attribute at the cell level to determine whether a cell is an always-on cell. Specify the attribute at the pin level to determine whether a pin is an always-on signal pin.

Syntax
always_on : Boolean expression ;

Boolean expression
Valid values are true and false.

Example
always_on : true ;

antenna_diode_type Simple Attribute
The antenna_diode_type attribute specifies the type of the antenna-diode cell. Valid values are power, ground, and power_and_ground.

Syntax
antenna_diode_type : true | false ;

Example
antenna_diode_type : power ;

area Simple Attribute
Use the area attribute to define the cell area in a cell or model group.

Syntax
area : float ;

float
A floating-point number. No units are explicitly given for the value, but you should use the same unit for the area of all cells in a library. Typical area units include gate equivalents, square microns, and transistors.
The following example shows a cell with an area of three units:

```
area : 3 ;
```

For unknown or undefined (black box) cells, the `area` attribute is optional. If no area statement is present, the default value is 0. Typically, unless a cell is a pad cell, it has an `area` attribute. Give pad cells an area of 0, because they are not used as internal gates.

---

**auxiliary_pad_cell Simple Attribute**

See “pad_cell Simple Attribute” on page 2-19.

---

**base_name Simple Attribute**

Use the `base_name` attribute to define a name for the output cell generated by VHDL or Verilog. If you omit this attribute, the cell is given the name "io_cell_name".

**Syntax**

```
bases_name : "cell_base_nameid" ;
```

**cell_base_name**

An alphanumeric string, enclosed in quotation marks, representing a name for the output cell.

**Example**

```
base_name : "IBUF" ;
```

---

**bus_naming_style Simple Attribute**

Use the `bus_naming_style` attribute to define the naming convention for buses in the library.

**Syntax**

```
bus_naming_style : "string" ;
```

**Example**

```
bus_naming_style : "Bus%sPin%d" ;
```

---

**cell_footprint Simple Attribute**

Use the `cell_footprint` attribute to assign a footprint class to a cell.
Syntax

cell_footprint : class_name id;

class_name

A character string that represents a footprint class. The string is case-sensitive: And4 is different from and4.

Example

cell_footprint : 5MIL;

Use this attribute to assign the same footprint class to all cells that have the same geometric layout characteristics.

---

cell Leakage_power Simple Attribute

Use the cell Leakage_power attribute to define the leakage power of a cell. You must define this attribute for cells with state-dependent leakage power. If cell Leakage_power is missing or negative, the value of the default_cell Leakage_power attribute defined in the library is assumed.

Note:

Cells with state-dependent leakage power also need the leakage_power Group. See “leakage_power Group” on page 2-101.

Syntax

cell Leakage_power : value float;

value

A floating-point number indicating the leakage power of the cell.

Example

cell Leakage_power : 0.2;

---

clock gating integrated_cell Simple Attribute

You can use the clock gating integrated_cell attribute to enter specific values that determine which integrated cell functionality the clock-gating tool uses.

Syntax

clock gating integrated_cell : generic | value id;
generic

by accessing the state tables and state functions of the library cell pins.

value

A concatenation of up to four strings that describe the functionality of the cell to the clock-gating software:

The first string specifies the type of sequential element you want. The options are latch-gating logic and none.

The second string specifies whether the logic is appropriate for rising- or falling-edge-triggered registers. The options are posedge and negedge.

The third (optional) string specifies whether you want test control logic located before or after the latch or flip-flop, or not at all. The options for cells set to latch or flip-flop are precontrol (before), postcontrol (after), or no entry. The options for cells set to no gating logic are control and no entry.

The fourth (optional) string, which exists only if the third string does, specifies whether you want observability logic or not. The options are obs and no entry. Table 2-1 lists some example values for the clock_gating_integrated_cell attribute.

Table 2-1  Some Values for the clock_gating_integrated_cell Attribute

<table>
<thead>
<tr>
<th>Value of clock_gating_integrated_cell</th>
<th>Integrated cell must contain</th>
</tr>
</thead>
<tbody>
<tr>
<td>latch_negedge</td>
<td>1. Latch-based gating logic</td>
</tr>
<tr>
<td></td>
<td>2. Logic appropriate for falling-edge-triggered registers</td>
</tr>
<tr>
<td>latch_posedge_postcontrol</td>
<td>1. Latch-based gating logic</td>
</tr>
<tr>
<td></td>
<td>2. Logic appropriate for rising-edge-triggered registers</td>
</tr>
<tr>
<td></td>
<td>3. Test control logic located after the latch</td>
</tr>
<tr>
<td>latch_negedge_precontrol</td>
<td>1. Latch-based gating logic</td>
</tr>
<tr>
<td></td>
<td>2. Logic appropriate for falling-edge-triggered registers</td>
</tr>
<tr>
<td></td>
<td>3. Test control logic located before the latch</td>
</tr>
</tbody>
</table>
For more details about the clock-gating integrated cells, see the “Modeling Power and Electromigration” chapter in the *Liberty User Guide, Vol. 1*.

**Example**

```vhs
clock_gating_integrated_cell : latch_posedge_precontrol_obs ;
```

### Setting Pin Attributes for an Integrated Cell

The clock-gating tool requires setting the pins of your integrated cells using the attributes listed in **Table 2-2**. Setting some of the pin attributes, such as those for test and observability, is optional.

**Table 2-2 Pin Attributes for Integrated Clock Gating Cells**

<table>
<thead>
<tr>
<th>Integrated cell pin name</th>
<th>Data direction</th>
<th>Required attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>in</td>
<td>clock_gate_clock_pin</td>
</tr>
<tr>
<td>enable</td>
<td>in</td>
<td>clock_gate_enable_pin</td>
</tr>
<tr>
<td>test_mode or scan_enable</td>
<td>in</td>
<td>clock_gate_test_pin</td>
</tr>
<tr>
<td>enable_clock</td>
<td>out</td>
<td>clock_gate_out_pin</td>
</tr>
</tbody>
</table>
Setting Timing for an Integrated Cell

You set both the setup and hold arcs on the enable pin by setting the `clock_gate_enable_pin` attribute for the integrated cell to true. The setup and hold arcs for the cell are determined by the edge values you enter for the `clock_gating_integrated_cell` attribute. Table 2-3 lists the edge values and the corresponding setup and hold arcs.

### Table 2-3 Values of the `clock_gating_integrated_cell` Attribute

<table>
<thead>
<tr>
<th>Value of <code>clock_gating_integrated_cell</code> attribute</th>
<th>Setup arc</th>
<th>Hold arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>latch_posedge</td>
<td>rising</td>
<td>rising</td>
</tr>
<tr>
<td>latch_negedge</td>
<td>falling</td>
<td>falling</td>
</tr>
<tr>
<td>none_posedge</td>
<td>falling</td>
<td>rising</td>
</tr>
<tr>
<td>none_negedge</td>
<td>rising</td>
<td>falling</td>
</tr>
</tbody>
</table>

contention_condition Simple Attribute

The `contention_condition` attribute specifies the contention conditions for a cell. Contention is a clash of “0” and “1” signals. In certain cells, it can be a forbidden condition and cause circuits to short.

**Syntax**

```
contention_condition : "Boolean expression" ;
```

**Example**

```
contention_condition : "ap * an" ;
```

dont_fault Simple Attribute

It is a string attribute that you can set on a library cell or pin for test.

**Syntax**

```
dont_fault : sa0 | sa1 | sa01 ;
```

`sa0, sa1, sa01`

The value you enter determines whether the `dont_fault` attribute are placed on stuck at 0 (sa0), stuck at 1 (sa1), or stuck on both faults (sa01).
Example
dont_fault : sa0 ;

The dont_fault attribute can also be defined in the pin group.

dont_touch Simple Attribute

The dont_touch attribute with a true value indicates that all instances of the cell must remain in the network.

Syntax
dont_touch : valueBoolean ;

value

Valid values are true and false.

Example
dont_touch : true ;

dont_use Simple Attribute

The dont_use attribute with a true value indicates that a cell should not be added to a design during optimization.

Syntax
dont_use : valueBoolean ;

value

Valid values are true and false.

Example
dont_use : true ;

driver_type Simple Attribute

Use the driver_type attribute to specify the drive power of the output or the I/O cell.
Syntax

driver_type : name_id;

name

An alphanumeric string identifier, enclosed in quotation marks, representing the drive power.

Example

driver_type : "4";

driver_waveform Simple Attribute

The driver_waveform attribute is an optional string attribute that allows you to define a cell-specific driver waveform. The value must be the driver_waveform_name predefined in the normalized_driver_waveform table.

When the attribute is defined, the cell uses the specified driver waveform during characterization. When it is not specified, the common driver waveform (the normalized_driver_waveform table without the driver_waveform_name attribute) is used for the cell.

Syntax

cell (cell_name) {
    ...
    driver_waveform : string;
    driver_waveform_rise : string;
    driver_waveform_fall : string;
}

Example

cell (my_cell1) {
    driver_waveform : clock_driver;
    ...
}
cell (my_cell2) {
    driver_waveform : bus_driver;
    ...
}
cell (my_cell3) {
    driver_waveform_rise : rise_driver;
    driver_waveform_fall : fall_driver;
    ...
}
**driver_waveform_rise and driver_waveform_fall**

**Simple Attributes**

The `driver_waveform_rise` and `driver_waveform_fall` string attributes are similar to the `driver_waveform` attribute. These two attributes allow you to define rise-specific and fall-specific driver waveforms. The `driver_waveform` attribute can coexist with the `driver_waveform_rise` and `driver_waveform_fall` attributes, though the `driver_waveform` attribute becomes redundant.

You should specify a driver waveform for a cell by using the following priority:

1. Use the `driver_waveform_rise` for a rise arc and the `driver_waveform_fall` for a fall arc during characterization. If they are not defined, specify the second and third priority driver waveforms.
2. Use the cell-specific driver waveform (defined by the `driver_waveform` attribute).
3. Use the library-level default driver waveform (defined by the `normalized_driver_waveform` table without the `driver_waveform_name` attribute).

The `driver_waveform_rise` attribute can refer to a `normalized_driver_waveform` that is either rising or falling. It is possible to invert the waveform automatically during runtime if necessary.

**Syntax**

```plaintext
cell (cell_name) {
    ... 
    driver_waveform : string;
    driver_waveform_rise : string;
    driver_waveform_fall : string;
}
```

**Example**

```plaintext
cell (my_cell1) {
    driver_waveform : clock_driver;
    ... 
}
cell (my_cell2) {
    driver_waveform : bus_driver;
    ... 
}
cell (my_cell3) {
    driver_waveform_rise : rise_driver;
    driver_waveform_fall : fall_driver;
    ... 
}
```
**em_temp_degradation_factor Simple Attribute**

The `em_temp_degradation_factor` attribute specifies the electromigration exponential degradation factor.

**Syntax**

```plaintext
em_temp_degradation_factor : value_float ;
```

**value**

A floating-point number in centigrade units consistent with other temperature specifications throughout the library.

**Example**

```plaintext
em_temp_degradation_factor : 40.0 ;
```

---

**fpga_cell_type Simple Attribute**

Interprets a combination timing arc between the clock pin and the output pin as a rising edge arc or as a falling edge arc.

**Syntax**

```plaintext
fpga_cell_type : value_enum ;
```

**value**

Valid values are `rising_edge_clock_cell` and `falling_edge_clock_cell`.

**Example**

```plaintext
fpga_cell_type : rising_edge_clock_cell ;
```

---

**fpga_isd Simple Attribute**

Use the `fpga_isd` attribute to reference the drive, io_type, and slew information contained in a library-level `fpga_isd` group.

**Syntax**

```plaintext
fpga_isd : fpga_isd_name_id ;
```

**fpga_isd_name**

The name of the library-level `fpga_isd` group.
interface_timing Simple Attribute

Indicates that the timing arcs are interpreted according to interface timing specifications semantics. If this attribute is missing or its value is set to false, the timing relationships are interpreted as those of a regular cell rather than according to interface timing specification semantics.

Syntax

```
interface_timing : true | false ;
```

The following example shows a cell with `interface_timing` set to true, indicating that interface timing semantics are to be applied.

Example

```
interface_timing : true ;
```

io_type Simple Attribute

Use the `io_type` attribute to define the I/O standard used by this I/O cell.

Syntax

```
io_type : nameid ;
```

name

An alphanumeric string identifier, enclosed in quotation marks, representing the I/O standard.

Example

```
io_type : "LVTTL" ;
```

is_pad Simple Attribute

The `is_pad` attribute identifies a pad pin on any I/O cell. You can also specify the `is_pad` attribute on PG pins. The valid values are `true` and `false`. If the cell-level `pad_cell` attribute is specified on a I/O cell, the `is_pad` attribute must be set to `true` in either a `pg_pin` group or on a signal pin for that cell.

Examples

```
cell(INBUF) {
```
is_pll_cell Simple Attribute

The `is_pll_cell` Boolean attribute identifies a phase-locked loop cell. A phase-locked loop (PLL) is a feedback control system that automatically adjusts the phase of a locally-generated signal to match the phase of an input signal.

Syntax

cell (cell_name) {
    is_pll_cell : true;
    pin (ref_pin_name) {
        is_pll_reference_pin : true;
        direction : output;
    }
    ...}

Example

cell(my_pll) {
    is_pll_cell : true;
    pin( REFCLK ) {
        direction : input;
        is_pll_reference_pin : true;
    }
    pin( FBKCLK ) {
        direction : input;
        is_pll_feedback_pin : true;
    }
    pin (OUTCLK1) {
        direction : output;
is_pll_output_pin : true;
timing() { /* Timing Arc */
    related_pin: "REFCLK";
    timing_type: combinational_rise;
    timing_sense: positive_unate;
    cell_rise(scalar) { /* Can be a LUT as well to support NLDM and CCS models */
        values("0.0")
    }
}
timing() { /* Timing Arc */
    related_pin: "REFCLK";
    timing_type: combinational_fall;
    timing_sense: positive_unate;
    cell_fall(scalar) {
        values("0.0")
    }
}
}

pin (OUTCLK2) {
    direction : output;
    is_pll_output_pin : true;
    timing() { /* Timing Arc */
        related_pin: "REFCLK";
        timing_type: combinational_rise;
        timing_sense: positive_unate;
        cell_rise(scalar) { /* Can be a LUT as well to support NLDM and CCS models */
            values("0.0")
        }
    }
    timing() { /* Timing Arc */
        related_pin: "REFCLK";
        timing_type: combinational_fall;
        timing_sense: positive_unate;
        cell_fall(scalar) {
            values("0.0")
        }
    }
}

---

is_clock_gating_cell Simple Attribute

The cell-level is_clock_gating_cell attribute specifies that a cell is for clock gating.

**Syntax**

```plaintext
is_clock_gating_cell : true | false ;
```
Example

\texttt{is\_clock\_gating\_cell : true;)

Set this attribute only on 2-input AND, NAND, OR, and NOR gates; inverters; buffers; and 2-input D latches.

\underline{is\_clock\_isolation\_cell Simple Attribute}

The \texttt{is\_clock\_isolation\_cell} attribute identifies a cell as a clock-isolation cell. The default is \texttt{false}, meaning that the cell is a standard cell. For information about pin-level attributes of the clock-isolation cell, see “\texttt{clock\_isolation\_cell\_clock\_pin Simple Attribute}” on page 3-10 and “\texttt{isolation\_cell\_enable\_pin Simple Attribute}” on page 3-32.

Syntax

\texttt{is\_clock\_isolation\_cell : true | false ;}

Example

\texttt{is\_clock\_isolation\_cell : true ;}

\underline{is\_isolation\_cell Simple Attribute}

The cell-level \texttt{is\_isolation\_cell} attribute specifies that a cell is an isolation cell. The pin-level \texttt{isolation\_cell\_enable\_pin} attribute specifies the enable input pin for the isolation cell.

Syntax

\texttt{is\_isolation\_cell : true | false ;}

Example

\texttt{is\_isolation\_cell : true ;}

\underline{is\_level\_shifter Simple Attribute}

The cell-level \texttt{is\_level\_shifter} attribute specifies that a cell is a level shifter cell. The pin-level \texttt{level\_shifter\_enable\_pin} attribute specifies the enable input pin for the level shifter cell.

Syntax

\texttt{is\_level\_shifter : Boolean expression ;}

\textit{Boolean expression}

Valid values are true and false.
Example

is_level_shifter : true ;

---

### is_macro_cell Simple Attribute

The **is_macro_cell** simple Boolean attribute identifies whether a cell is a macro cell. If the attribute is set to true, the cell is a macro cell. If it is set to false, the cell is not a macro cell.

**Syntax**

is_macro_cell : Boolean expression ;

**Boolean expression**

Valid values are true and false.

Example

is_macro_cell : true ;

---

### is_soi Simple Attribute

The **is_soi** attribute specifies that the cell is a silicon-on-insulator (SOI) cell. The default is false, which means that the cell is a bulk-CMOS cell.

If the **is_soi** attribute is specified at both the library and cell levels, the cell-level value overrides the library-level value.

**Syntax**

is_soi : true | false ;

Example

is_soi : true ;

For more information about the **is_soi** attribute and SOI cells, see the “Advanced Low-Power Modeling” chapter of the *Liberty User Guide, Vol. 1*.

---

### level_shifter_type Simple Attribute

The **level_shifter_type** attribute specifies the voltage conversion type that is supported. Valid values are:

LH

  Low to High
HL
  High to Low

HL_LH
  High to Low and Low to High

The `level_shifter_type` attribute is optional.

**Syntax**

```plaintext
level_shifter_type : level_shifter_type_value ;
```

**Example**

```plaintext
level_shifter_type : HL_LH ;
```

---

**map_only Simple Attribute**

The `map_only` attribute with a true value indicates that a cell is excluded from logic-level optimization during compilation.

**Syntax**

```plaintext
map_only : true | false ;
```

---

**pad_cell Simple Attribute**

In a cell group or a model group, the `pad_cell` attribute identifies a cell as a pad cell.

**Syntax**

```plaintext
pad_cell : true | false ;
```

If the `pad_cell` attribute is included in a cell definition (true), at least one pin in the cell must have an `is_pad` attribute.

**Example**

```plaintext
pad_cell : true ;
```

If more than one pad cell can be used to build a logical pad, use the `auxiliary_pad_cell` attribute in the cell definitions of all the component pad cells.

**Syntax**

```plaintext
auxiliary_pad_cell : true | false ;
```
Example
auxiliary_pad_cell : true ;

If the pad_cell or auxiliary_pad_cell attribute is omitted, the cell is treated as an internal core cell rather than as a pad cell.

Note:
A cell with an auxiliary_pad_cell attribute can also be used as a core cell; a pull-up or pull-down cell is an example of such a cell.

---

**pad_type Simple Attribute**

Use the pad_type attribute to identify a type of pad_cell or auxiliary_pad_cell that requires special treatment.

**Syntax**

```
pad_type : value ;
```

**Example**

```
pad_type : clock;
```

---

**power_cell_type Simple Attribute**

Use the power_cell_type attribute to specify the cell type.

**Syntax**

```
power_cell_type : value_enum ;
```

**value**

Valid values are stdcell (standard cell) and macro (macro cell).

**Example**

```
power_cell_type : stdcell ;
```

---

**power_gating_cell Simple Attribute**

Note:
The power_gating_cell attribute has been replaced by the retention_cell attribute. See “retention_cell Simple Attribute” on page 2-21.

The cell-level power_gating_cell attribute specifies that a cell is a power-switch cell. A power-switch cell has two modes. When functioning in normal mode, the power-switch cell
functions as a regular cell. When functioning in power-saving mode, the power-switch cell’s power supply is shut off.

The pin-level map_to_logic attribute specifies which logic level the power_gating_cell is tied to when the cell is functioning in normal mode.

Syntax

```
power_gating_cell : power_gating_cell_name id ;
```

**power_gating_cell_name**

A string identifying the power-switch cell name.

**Example**

```
power_gating_cell : "my_gating_cell" ;
```

---

**preferred Simple Attribute**

The preferred attribute with a true value indicates that the cell is the preferred replacement during the gate-mapping phase of optimization.

Syntax

```
preferred : true | false ;
```

**Example**

```
preferred : true ;
```

This attribute can be applied to a cell with preferred timing or area attributes. For example, in a set of 2-input NAND gates, you might want to use gates with higher drive strengths wherever possible. This practice is useful primarily in design translation.

---

**retention_cell Simple Attribute**

The retention_cell attribute identifies a retention cell. The retention_cell_style value is a random string.

Syntax

```
retention_cell : retention_cell_style ;
```

**Example**

```
retention_cell : my_retention_cell ;
```
sensitization_master Simple Attribute

The sensitization_master attribute defines the sensitization group referenced by the cell to generate stimuli for characterization. The attribute is required if the cell contains sensitization information. Its string value should be any sensitization group name predefined in the current library.

Syntax

sensitization_master : sensitization_group_name;

sensitization_group_name

A string identifying the sensitization group name predefined in the current library.

Example

sensitization_master : sensi_2in_1out;

single_bit_degenerate Simple Attribute

The single_bit_degenerate attribute names a single-bit library cell that can be used by an optimization tool to link a multibit black-box cell with the single-bit version of the cell.

Syntax

single_bit_degenerate : "cell_nameid";

cell_name

A character string identifying a single-bit cell.

Example

cell (FDX2) {
  area : 18 ;
  single_bit_degenerate : "FDB" ; /* FDB must be a single-bit cell in the library*/
  bundle (D) {
    members (D0, D1) ;
    direction : input ;
    ...
    timing ( ) {
      ...
      ...
    }
  }
}
cell (FDX4) {
  area : 32 ;
  single_bit_degenerate : "FDB" ;
slew_type Simple Attribute

Use the `slew_type` attribute to specify the slew type for the output pins of the output or the I/O cell.

**Syntax**

```
slew_type : "nameid ";
```

**name**

An alphanumeric string identifier, enclosed in quotation marks, representing the slew type.

**Example**

```
slew_type : "slow" ;
```

switch_cell_type Simple Attribute

The `switch_cell_type` cell-level attribute specifies the type of the switch cell for direct inference.

**Syntax**

```
switch_cell_type : coarse_grain | fine_grain;
```

**Example**

```
switch_cell_type : coarse_grain ;
```

threshold_voltage_group Simple Attribute

The optional `threshold_voltage_group` attribute specifies a cell’s category based on its threshold voltage characteristics.
Syntax
threshold_voltage_group : "group_nameid" ;

group_name
A string value representing the name of the category.

Example
cell () {

  ...
  threshold_voltage_group : "high_vt_cell" ;
  ...
  threshold_voltage_group : "low_vt_cell" ;
  ...
}

timing_model_type Simple Attribute
When specified, indicates that static timing analysis tools must not automatically infer transparent level-sensitive latch devices from timing arcs defined in the cell. To indicate that transparent level-sensitive latches should be inferred for input pins, use the tlatch group.

Syntax
timing_model_type : "nameid" ;

name
Valid values are "abstracted", "extracted", and "qtm".

Example
timing_model_type : "abstracted" ;

use_for_size_only Simple Attribute
Set this attribute on a cell at the library level to specify the criteria for sizing optimization.

Syntax
use_for_size_only : valueBoolean ;

value
Valid values are true and false.

Example
library(lib1){
  cell(cell1){
Complex Attributes

This section lists, alphabetically, the complex attributes for the cell and model groups.

input_voltage_range Attribute

The input_voltage_range attribute specifies the allowed voltage range of the level-shifter input pin and the voltage range for all input pins of the cell under all possible operating conditions (defined across multiple libraries). The attribute defines two floating values: the first is the lower bound, and second is the upper bound.

The input_voltage_range syntax differs from the pin-level input_signal_level_low and input_signal_level_high syntax in the following ways:

- The input_signal_level_low and input_signal_level_high attributes are defined on the input pins under one operating condition.

- The input_signal_level_low and input_signal_level_high attributes are used to specify the partial voltage swing of an input pin (that is, to prevent from swinging from ground rail VSS to full power rail VDD). Note that input_voltage_range is not related to the voltage swing.

Note:
The input_voltage_range and output_voltage_range attributes should always be defined together.

Syntax

input_voltage_range (float, float) ;

Example

input_voltage_range (1.0, 2.0) ;
output_voltage_range Attribute

The output_voltage_range attribute is similar to the input_voltage_range attribute except that it specifies the allowed voltage range of the level shifter for the output pin instead of the input pin.

The output_voltage_range syntax differs from the pin-level output_signal_level_low and output_signal_level_high syntax in the following ways:

- The output_signal_level_low and output_signal_level_high attributes are defined on the output pins under one operating condition.
- The output_signal_level_low and output_signal_level_high attributes are used to specify the partial voltage swing of an output pin (that is, to prevent from swinging from ground rail VSS to full power rail VDD). Note that output_voltage_range is not related to the voltage swing.

Note:

The input_voltage_range and output_voltage_range attributes should always be defined together.

Syntax

output_voltage_range (float, float) ;

Example

output_voltage_range (1.0, 2.5) ;

pin_equal Complex Attribute

Use the pin_equal attribute to describe functionally equal (logically equivalent) groups of input or output pins.

Syntax

pin_equal ("name_list") ;

name_list

A list of input or output pins whose values must be equal.

Example

In the following example, input pins IP1 and IP0 are logically equivalent.

pin_equal ("IP1 IP0") ;
**pin_name_map Complex Attribute**

The `pin_name_map` attribute defines the pin names that are used to generate stimuli from the sensitization group for all timing arcs in the cell. The `pin_name_map` attribute is optional when the pin names in the cell are the same as the pin names in the sensitization master, but it is required when they are different.

If the `pin_name_map` attribute is set, the number of pins must be the same as that in the sensitization master, and all pin names should be legal pin names for the cell.

Syntax

```
pin_name_map (string..., string);
```

Example

```
pin_name_map (A, B, Z);
```

**pin_opposite Complex Attribute**

Use the `pin_opposite` attribute to describe functionally opposite (logically inverse) groups of input or output pins.

Syntax

```
pin_opposite ("name_list1", "name_list2") ;
```

**name_list1, name_list2**

A `name_list` of output pins requires the supplied output values to be opposite. A `name_list` of input pins requires the supplied input values to be opposite.

In the following example, pins IP and OP are logically inverse.

```
pin_opposite ("IP", "OP") ;
```

The `pin_opposite` attribute also incorporates the functionality of the `pin_equal` complex attribute.

In the following example, Q1, Q2, and Q3 are equal; QB1 and QB2 are equal; and the pins in the first group are opposite of the pins in the second group.

```
pin_opposite ("Q1 Q2 Q3", "QB1 QB2") ;
```

**resource_usage Complex Attribute**

Use the `resource_usage` attribute to specify the name and the number of resources the cell uses.
**Syntax**

resource_usage ( resource_nameid, number_of_resourcesint ) ;

**resource_name**

An alphanumeric identifier that matches the first argument in a max_count attribute in the library. You can specify multiple resource_usage attributes with different resource names.

**number_of_resources**

An integer representing the number of resources the cell uses.

**Example**

resource_usage (RES1, 1) ;

---

### Group Statements

This section lists, alphabetically, the group statements for the cell and model groups.

#### cell Group Example

Example 2-2 shows cell definitions that include some of the CMOS cell attributes described so far.

**Example 2-2  cell Group Example**

library (cell_example){
  date : "December 12, 2014" ;
  revision : 2.3 ;

  cell (in){
    area : 0;  /* pads do not normally consume
                internal core area*/
    cell_footprint : 5MIL ;
    pin (A) {
      direction : input;
      capacitance : 0;
    }
    pin (Z) {
      direction : output;
      function : "A";
      timing () {...}
    }
  }
  cell(inverter_med){
    area : 3;
    preferred : true;
}
/* Application tools use this inverter first during optimization */
pin (A) {
  direction : input;
  capacitance : 1.0;
}
pin (Z) {
  direction : output;
  function : "A’"
  timing () { ...} 
}
cell(and_nand){
  area : 4;
pin_opposite("Y", "Z");
pin(A) {
  direction : input
  capacitance : 1
  fanout_load : 1.0
}
pinup) {
  direction : input
  capacitance : 1
  fanout_load : 1.0
}
pin (Y) {
  direction : output
  function : "(A * B)’"
  timing() { ...} 
}
pin (Z){
  direction : output
  function : "(A * B)"
  timing() { ...} 
}
cell(buff1){
  area : 3;
pin_equal ("Y Z");
pin (A) {
  direction : input;
  capacitance : 1.0;
}
pin (Y) {
  direction : output;
  function : "A"
  timing () { ...} 
}
pin (Z) {
  direction : output;
  function : "A"
  timing () { ...} 
}
critical_area_table Group

The critical_area_table group specifies a critical area table at the cell level that is used for critical area analysis modeling. The critical_area_table group uses critical_area_lut_template as the template. The critical_area_table group contains the defect_type, related_layer, index_1, and values attributes.

Syntax

```plaintext
library(my_library) {
  critical_area_table (template_name) {
    defect_type : enum (open, short, open_and_short);
    related_layer : string;
    index_1 ("float...float");
    values ("float...float");
  }
}
```

Example

```plaintext
library(my_library) {
  critical_area_table (caa_template) {
    defect_type : short;
    related_layer : M1;
    index_1 ("0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17");
    values ("0.03, 0.08, 0.17, 0.28, 0.40, 0.54, 0.68, 0.81, 0.95, 1.09");
  }
```

defect_type Attribute

The defect_type attribute value indicates whether the critical area analysis table values are measured against a short or open electrical failure when particles fall on the wafer. The following enumerated values are accepted: short, open and open_and_short. The open_and_short attribute value specifies that the critical area analysis table is modeled for both short and open failure types. If the defect_type attribute is not specified, the default is open_and_short.

Syntax

```plaintext
defect_type : enum (open, short, open_and_short);
```
Example

library(my_library) {
  ...  
critical_area_table (ca_table) {
    defect_type : short;
    related_layer : M1;
    index_1 ("0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17");
    values ("0.03, 0.08, 0.17, 0.28, 0.40, 0.54, 0.68, 0.81, 0.95, 1.09");
  }
  ...
}

related_layer Attribute

The related_layer attribute defines the names of the layers to which the critical area analysis table values apply. All layer names must be predefined in the library's layer definitions.

Syntax

related_layer : string;

Example

library(my_library) {
  ...
  critical_area_table (ca_table) {
    defect_type : short;
    related_layer : M1;
    index_1 ("0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17");
    values ("0.03, 0.08, 0.17, 0.28, 0.40, 0.54, 0.68, 0.81, 0.95, 1.09");
  }
  ...
}

index_1 Attribute

The index_1 attribute defines the defect size diameter array in the unit of distance_unit. The attribute is optional if the values for this array are the same as that in the critical_area_lut_template.

Syntax

index_1 ("float...float");

Example

library(my_library) {  
  ...
}
critical_area_table (caa_template) {
    defect_type : short;
    related_layer : M1;
    index_1("0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17");
    values ("0.03, 0.08, 0.17, 0.28, 0.40, 0.54, 0.68, 0.81, 0.95, 1.09");
}

values Attribute

The values attribute defines critical area values for nonvia layers in the unit of distance_unit squared. For via layers, the values attribute specifies the number of single cuts on the layer.

Syntax

values ("float...float");

Example

library(my_library) {
    ...
    critical_area_table (caa_template) {
        defect_type : short;
        related_layer : M1;
        index_1("0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17");
        values ("0.03, 0.08, 0.17, 0.28, 0.40, 0.54, 0.68, 0.81, 0.95, 1.09");
    }
    ...
    ...
}

bundle Group

A bundle group uses the members complex attribute (unique to bundles) to group together in multibit cells—such as quad latches and 4-bit registers—several pins that have similar timing or functionality.

The bundle group contains the following elements:

• The members complex attribute. It must be declared first in a bundle group.
• All simple attributes that also appear in a pin group.
• The pin group statement (including all the pin group simple and complex attributes, and group statements).
Syntax

```plaintext
library (nameid) {
  cell (nameid) {
    single_bit_degenerate : string;
    bundle (string) {
      members (string) ; /*Must be declared first*/
      capacitance : float;
      ...
    }
    pin (Z0) {
      capacitance : float;
      ...
      timing () {
        ...
      }
    }
  }
}
```

Note:

*Bundle names, bundle elements, bundle members, members, and member pins are all valid terms for pin names in a bundle group.*

Simple Attributes

All pin group simple attributes are valid in a bundle group. Following are examples of three simple attributes.

```plaintext
capacitance : float;
direction : input | output | inout |internal;
function : "Boolean";
```

Complex Attribute

```plaintext
members (nameid) ;
```

Group Statement

All pin group statements are valid in a bundle group.

```plaintext
pin (nameid | name_listid) { }
```

Pin Attributes in a bundle Group

The pin group simple attributes in a bundle group define default attribute values for all pins in that bundle group. The pin attributes can also appear in a pin group within the bundle group.
capacitance Simple Attribute

Use the capacitance attribute to define the load of an input, output, inout, or internal pin.

Syntax

capacitance : value_{float} ;

value

A floating-point number in units consistent with other capacitance specifications throughout the library. Typical units of measure for capacitance include picofarads and standardized loads.

Example

The following example shows a bundle group that defines a capacitance attribute value of 1 for input pins D0, D1, D2, and D3 in bundle D:

bundle (D) {
    members(D0, D1, D2, D3) ;
    direction : input ;
    capacitance : 1 ;
}

direction Simple Attribute

The direction attribute states the direction of member pins in a bundle group.

The direction listed for this attribute should be the same as that given for the pin in the same bundle group (see the bundle Z pin in Example 2-3).

Syntax

direction : input | output | inout | internal ;

Example

In a bundle group, the direction of all pins must be the same. Example 2-3 shows two bundle groups. The first group shows two pins (Z0 and Z1) whose direction is output. The second group shows one pin (D0) whose direction is input.

Example 2-3  Direction of Pins in bundle Groups

cell(inv) {
    area : 16 ;
    cell_leakage_power : 8 ;
    bundle(Z) {
        members(Z0, Z1, Z2, Z3) ;
        direction : output ;
        function : "D" ;
    }

    pin(Z0) {

direction : output ;
timing() {
    ...
    related_pin : "D0" ;
}
}

pin(Z1) {
    direction : output ;
timing() {
    ...
    related_pin : "D1" ;
}
}

bundle(D) {
    members (D0, D1, D2, D3) ;
direction : input ;
capacitance : 1 ;
pin (D0 {
    direction : input ;
    ...
}
)
}

function Simple Attribute

The function attribute in a bundle group defines the value of an output pin or inout pin in terms of the input pins or inout pins in the cell group or model group.

Syntax

function : "Boolean expression" ;
Table 2-4 lists the Boolean operators valid in a function statement.

### Table 2-4 Valid Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`</td>
<td>invert previous expression</td>
</tr>
<tr>
<td>!</td>
<td>invert following expression</td>
</tr>
<tr>
<td>^</td>
<td>logical XOR</td>
</tr>
<tr>
<td>*</td>
<td>logical AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td>space</td>
<td>logical AND</td>
</tr>
<tr>
<td>+</td>
<td>logical OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td>0</td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>

The order of precedence of the operators is left to right, with inversion performed first, then XOR, then AND, then OR.

### Pin Names as Function Arguments

The following example describes bundle Q with the function $A \text{ OR } B$:

```markdown
bundle (Q) {
    direction : output ;
    function : "A + B" ;
}
```

A pin name beginning with a number must be enclosed in double quotation marks preceded by a backslash (**\**), as in the following example.

```markdown
function : " \"1A\" + \"1B\" " ;
```

The absence of a backslash causes the quotation marks to terminate the function string.

The following function statements all describe 2-input multiplexers. The parentheses are optional. The operators and operands are separated by spaces.

```markdown
function : "A S + B S'" ;
```
function : "A & S | B & !S" ;
function : "(A * S) + (B * S')" ;

members Complex Attribute

The members attribute lists the pin names of signals in a bundle. It provides the bundle element names, and it groups a set of pins that have similar properties. It must be the first attribute you declare in a bundle group.

Syntax

bundle (name\text{string}) {
    members (member1, member2 ... ) ;
    ... 
}

member1, member2 ...

The number of bundle members defines the width of the bundle.

Example

members (D1, D2, D3, D4) ;

If the function attribute has been defined for the bundle, the function value is copied to all bundle members.

Example

bundle(A) {
    members(A0, A1, A2, A3);
    direction : output ;
    function : "B' + C";
    ... 
}
bundle(B) {
    members(B0, B1, B2, B3);
    direction : input;
    ... 
}

The previous example shows that the members of the A bundle have these values:

A0 = B0' + C ;
A1 = B1' + C ;
A2 = B2' + C ;
A3 = B3' + C ;

Each bundle operand (B) must have the same width as the function parent bundle (A).

Example 2-4 shows how to define a bundle group in a cell with a multibit latch.
Example 2-4  Multibit Latch With Signal Bundles

cell (latch4) {
    area: 16 ;
    single_bit_degenerate : FDB ;
    pin (G) { /* active-high gate enable signal */
        direction : input ;
        ...
    }
    bundle (D) { /* data input with four member pins */
        members (D1, D2, D3, D4) ; /*must be first attribute */
        direction : input ;
    }
    bundle (Q) {
        members (Q1, Q2, Q3, Q4) ;
        direction : output ;
        function : "IQ" ;
    }
    bundle (QN) {
        members (Q1N, Q2N, Q3N, Q4N) ;
        direction : output ;
        function : "IQN" ;
    }
    latch_bank(IQ, IQN, 4) {
        enable : "G" ;
        data_in : "D" ;
    }
}

pin Group Statement in a bundle Group

You can define attribute values for specific pins or groups of pins in a pin group within a bundle group. Values in a pin group override the default attribute values defined for the bundle (described previously).

Syntax

bundle(namestring) {
    pin (namestring) {
        ... pin description ...
    }
}


The following example shows a pin group in a bundle group that defines a new capacitance attribute value for member A0 in bundle A.

Example

```plaintext
bundle (A) {
  pin (A0) {
    capacitance : 4 ;
  }
}
```

To identify the name of a pin in a pin group within a bundle group, use the full name of a pin, such as pin (A0) in the previous example.

All pin names within a single bundle group must be unique. Pin names are case-sensitive; for example, pins named A and a are different pins.

Example 2-3 on page 2-34 shows a pin group within a bundle group.

---

**bus Group**

A bus group, defined in a cell group or a model group, defines the bused pins in the library. Before you can define a bus group you must first define a type group at the library level.

From the type group you define at the library level, use the type name (bus4 in Example 2-5) as the value for the bus_type attribute in the bus group in the same library.

Example 2-5 shows a bus group in a cell group.

Example 2-5  Bused Pins

```plaintext
library (ExamBus) {
  type (bus4) {   /* bus name */
    bit_width : 4 ;  /* number of bused pins */
    ...
  }
  cell (bused cell) {
    ...
    bus (A) {
      bus_type : bus4 ; /* bus name */
      ...
    }
  }
}
```
Simple Attributes

You can use all the `pin` simple attributes in a `bus` group and the following attributes.

```plaintext
bus_type : name ;
scan_start_pin : pin_name ;
scan_pin_inverted : true | false ;
```

Group Statement

```plaintext
pin (name_string | name_list_string) { }
```

All group statements that appear in a `pin` group are valid in a `bus` group.

**bus_type Simple Attribute**

The `bus_type` attribute is a required element of all `bus` groups. The attribute defines the type of bus. It must be the first attribute declared in a `bus` group.

**Syntax**

```plaintext
bus_type : name ;
```

`name`

Define this name in the applicable `type` group in the library, as shown in Example 2-5.

**scan_start_pin Simple Attribute**

The optional `scan_start_pin` attribute specifies the scan output pin of a sequential element of a multibit scan cell, where the internal scan chain begins. This attribute applies only to output buses and bundles of multibit scan cells.

Only the following scan chains are supported:

- From the least significant bit (LSB) to the most significant bit (MSB) of the output `bus` group; and
- From the MSB to the LSB of the output `bus` group.

Therefore, for a multibit scan cell with internal scan chain, the value of the `scan_start_pin` attribute can either be the LSB, or the MSB output pin.

Specifying the LSB scan output pin as the value of the `scan_start_pin` attribute indicates that the scan signal shifts from the LSB sequential element to the MSB sequential element of the multibit scan cell.

Specifying the MSB scan output pin as the value of the `scan_start_pin` attribute indicates that the scan signal shifts from the MSB sequential element to the LSB sequential element of the multibit scan cell.
Syntax

scan_start_pin : pin_name ;

scan_pin_inverted Attribute

The optional scan_pin_inverted attribute specifies that the scan signal is inverted (after the first sequential element of the multibit scan cell). This attribute applies only to output buses and bundles of multibit scan cells. The default is false.

If you specify the scan_pin_inverted attribute value as true, you must specify the value of the signal_type attribute as test_scan_out_inverted.

If you specify the scan_pin_inverted attribute, you must specify the scan_start_pin attribute in the same bus group.

Syntax

scan_pin_inverted : true | false ;

pin Simple Attributes in a bus Group

The pin simple attributes in a bus group define default attribute values for all pins in the bus group.

Note:
- Bus names, bus members, bus pins, bused pins, pins, members, member numbers, and range of bus members are valid terms for pin names in a bus group.
- All pin group simple attributes are valid within a bus group and within a pin group in a bus group.
- The capacitance and direction attributes are frequently used in bus groups.

capacitance Simple Attribute

Use the capacitance attribute to define the load of an input, output, inout, or internal pin.

Syntax

capacitance : value_float ;

value

A floating-point number in units consistent with other capacitance specifications throughout the library. Typical units of measure for capacitance include picofarads and standardized loads.

The following example shows a bus group that defines bus A with default values assigned for direction and capacitance.
Example
bus (A) {
    bus_type : bus1 ;
    direction : input ;
    capacitance : 3 ;
}

direction Simple Attribute
The direction attribute states the direction of bus members (pins) in a bus group.
The value of the direction attribute of all bus members (pins) in a bus group must be the
same. (See Example 2-6 for a bus group with more than one pin.)

Syntax
direction : input | output | inout | internal ;

Example
direction : inout ;

pin Group Statement in a bus Group
This group defines attribute values for specific bused pins or groups of bused pins in a pin
group within a bus group. Values used in a pin group within a bus group override the
defined default bus pin attribute values described previously.

Note:
    You can use a defined bus or buses in Boolean expressions in the function attribute of
    a pin in a bus group, as shown in Example 2-6 on page 2-43.

The following example shows a bus pin group that defines a new capacitance attribute
value for member AO in bus A.

bus(A) {
    pin (AO) {
        capacitance : 4 ;
    }
}

To identify the name of a bused pin in a pin group within a bus group, use the full name of
the pin. You can identify bus member numbers as single numbers or as a range of numbers
separated by a colon. No spaces can appear between the colon and the member numbers.

Example
pin (A[0:2]) {}
bus(A)
  pin (A)
    capacitance : 4 ;
  }
}

The next example shows a pin group within a bus group that defines a new capacitance attribute value for bus members 0, 1, 2, and 3 in bus A.

bus(A)
  pin (A[0:3])
    capacitance : 4 ;
  }
}

Example Bus Description—Technology Library

Example 2-6 illustrates a complete bus description that includes a library-defined type group and cell-defined bus groups. The example also illustrates the use of bus variables in a function attribute in a pin group and in a related_pin attribute in a timing group.

Example 2-6 Bus Description

library (ExamBus) {
  date : "November 12, 2000" ;
  revision : 2.3 ;
  bus_naming_style : "%s[%d]" ;
  /* Optional; this is the default */
  type (bus4) {
    base_type : array ;/* Required */
    data_type : bit ;/* Required if base_type is array */
    bit_width : 4 ;/* Optional; default is 1 */
    bit_from : 0 ;/* Optional MSB; defaults to 0 */
    bit_to : 3 ;/* Optional LSB; defaults to 0 */
    downto : false ;/* Optional; defaults to false */
  }
  cell (bused_cell) {
    area : 10 ;
    single_bit_degenerate : FDB ;
  bus (A) {
    bus_type : bus4 ;
    direction : input ;
    capacitance : 3 ;
    pin (A[0:2])
      capacitance : 2 ;
    }
    pin (A[3])
      capacitance : 2.5 ;
    }
  }
  bus (B) {
    bus_type : bus4 ;
    direction : input ;
  }
}
capacitance : 2 ;
}
bus (E) {
direction : input ;
capacitance 2 ;
}
bus (X) {
bus_type : bus4 ;
direction : output ;
capacitance : 1 ;
pin (X[0:3]) {
function : "A & B'" ;
timing() {
   related_pin : "A B" ;
   /* A[0] and B[0] are related to X[0],
   A[1] and B[1] are related to X[1], etc. */
}
}
bus (Y) {
bus_type : bus4 ;
direction : output ;
capacitance : 1 ;
pin (Y[0:3]) {
function : "B" ;
three_state : "!E" ;
timing () {
   related_pin : "A[0:3] B E" ;
}
}
}
bus (Z) {
bus_type : bus4 ;
direction : output ;
pin (Z[0:1]) {
function : "!A[0:1]" ;
timing () {
   related_pin : "A[0:1]" ;
}
}
pin (Z[2]) {
function "A[2]" ;
timing () {
   related_pin : "A[2]" ;
}
}
pin (Z[3]) {
function : "!A[3]" ;
timing () {
   related_pin : "A[3]" ;
}
}
char_config Group

Use the char_config group to specify the characterization settings for the library cells.

Syntax

cell (cell_name) {
    char_config() {
        /* characterization configuration attributes */
        ...
    }
}

Simple Attributes

- internal_power_calculation
- three_state_disable_measurement_method
- three_state_disable_current_threshold_abs
- three_state_disable_current_threshold_rel
- three_state_disable_monitor_node
- three_state_cap_add_to_load_index
- ccs_timing_segment_voltage_tolerance_rel
- ccs_timing_delay_tolerance_rel
- ccs_timing_voltage_margin_tolerance_rel
- receiver_capacitance1_voltage_lower_threshold_pct_rise
- receiver_capacitance1_voltage_upper_threshold_pct_rise
- receiver_capacitance1_voltage_lower_threshold_pct_fall
- receiver_capacitance1_voltage_upper_threshold_pct_fall
- receiver_capacitance2_voltage_lower_threshold_pct_rise
- receiver_capacitance2_voltage_upper_threshold_pct_rise
- receiver_capacitance2_voltage_lower_threshold_pct_fall
- receiver_capacitance2_voltage_upper_threshold_pct_fall
- capacitance_voltage_lower_threshold_pct_rise
- capacitance_voltage_lower_threshold_pct_fall
- capacitance_voltage_upper_threshold_pct_rise
- capacitance_voltage_upper_threshold_pct_fall

Complex Attributes

- driver_waveform
- driver_waveform_rise
- driver_waveform_fall
- input_stimulus_transition
- input_stimulus_interval
- unrelated_output_net_capacity
- default_value_selection_method
- default_value_selection_method_rise
- default_value_selection_method_fall
- merge_tolerance_abs
- merge_tolerance_rel
merge_selection

Example

cell (cell_test) {
  char_config() {
    /* input driver for cell_test specifically */
    driver_waveform (all, input_driver_cell_test) ;
    default_value_selection_method (constraint, max) ;
    default_value_selection_method_rise(nldm_transition, min) ;
    default_value_selection_method_fall(nldm_transition, max) ;
    ...
  }
}

For more information about the char_config group and the group attributes, see “char_config Group” on page 1-28.

---

clear_condition Group

The clear_condition group is a group of attributes that specify the condition for the clear signal when a retention cell operates in the normal mode.

If the clear signal is asserted during the restore event, it needs to be active for a time longer than the restore event so that the flip-flop content is successfully overwritten. Therefore, the clear pin must be checked at the trailing edge.

Syntax

clear_condition() {
  input : "Boolean_expression" ;
  required_condition : "Boolean_expression" ;
}

Example

clear_condition() {
  input : "!RN"; /* When clear de-asserts, RET must be high to allow
                  the low value to be transferred */
  required_condition : "RET";
}

Simple Attributes

input
required_condition

input Attribute

The input attribute must be identical to the clear attribute in the ff group and defines how the asynchronous clear control is asserted.
Syntax
input : "Boolean_expression" ;

Example
input : "!RN" ;

required_condition Attribute
The required_condition attribute specifies the input condition during the active edge of the clear signal. The required_condition attribute is checked at the trailing edge of the clear signal. When the input condition is not met, the cell is in an illegal state.

Syntax
required_condition : "Boolean_expression" ;

Example
required_condition : "RET" ;

clock_condition Group
The clock_condition group is a group of attributes that specify the input conditions for correct clock signal during clock-based events.

The clock_condition group includes two classes of attributes: attributes without the _also suffix and attributes with the _also suffix. They are similar to the clocked_on and clocked_on_also attributes of the ff group.

Syntax
clock_condition() {
    clocked_on : "Boolean_expression";
    required_condition : "Boolean_expression";
    hold_state : L|H|N ;
    clocked_on_also : "Boolean_expression";
    required_condition_also : "Boolean_expression";
    hold_state_also : L|H|N;
}

Example
clock_condition() {
    clocked_on : "CK"; /* clock must be Low to go into retention mode */
    hold_state : "N"; /* when clock switches (either direction), RET must be High */
    required_condition : "RET";
}

Simple Attributes
clocked_on
clocked_on_also
required_condition
required_condition_also
hold_state
hold_state_also

clocked_on Attribute

The clocked_on attribute must be identical to the clocked_on attribute of the ff or ff_bank group.

Syntax

clocked_on : "Boolean_expression" ;

Example

clocked_on : "CK" ;

clocked_on_also Attribute

The clocked_on_also attribute must be identical to the clocked_on_also attribute of the ff or ff_bank group.

Syntax

clocked_on_also : "Boolean_expression" ;

Example

clocked_on_also : "CK" ;

required_condition Attribute

The required_condition attribute specifies the input conditions during the active edge of the clock signal. If the conditions are not met, the cell is in an illegal state.

Syntax

required_condition : "Boolean_expression" ;

Example

required_condition : "RET" ;

required_condition_also Attribute

The required_condition_also attribute specifies the input conditions during the active edge of the clock signal. If the conditions are not met, the cell is in an illegal state. It is evaluated at the rising edge of the clock signal specified by the clocked_on_also attribute. If the clocked_on_also attribute is not specified, it is evaluated at the negative edge of the clock signal specified by the clocked_on attribute.
Syntax
required_condition_also : "Boolean_expression" ;

Example
required_condition_also : "RET" ;

hold_state Attribute
The hold_state attribute specifies the values for the Boolean expression of the
clocked_on attribute during the retention mode. Valid values are L, H, or N that represent
low, high, or no-change respectively.

If retention data is restored to both master and slave latches, the hold_state is N. If
retention data is restored only to the slave latch, the hold_state attribute is L for the slave
latch to keep the data.

Syntax
hold_state : "L | H | N" ;

Example
hold_state : "L" ;

hold_state_also Attribute
The hold_state_also attribute specifies the values for the Boolean expression of the
clocked_on_also attribute during the retention mode. Valid values are L, H, or N that
represent low, high, or no-change respectively.

Syntax
hold_state_also : "L | H | N" ;

Example
hold_state_also : "L" ;

dynamic_current Group
Use the dynamic_current group to specify a current waveform vector when the power and
ground current is dependent on the logical condition of a cell. A dynamic_current group is
defined in a cell group, as shown here:

library (name) {
cell (name) {
dynamic_current () {
  when : boolean expression;
  related_inputs : input_pin_name;
  related_outputs : output_pin_name;
  typical_capacitances("float, ...");
event (mode_definition_name, event_name);
switching_group() {
    input_switching_condition(enum(rise, fall));
    output_switching_condition(enum(rise, fall));
    pg_current(pg_pin_name) {
        vector(template_name) {
            reference_time : float;
            index_output : output_pin_name;
            index_1(float);
            ...
            index_n(float);
            index_n+1("float, ...");
            values("float, ...");
        } /* vector */
        ...
    } /* pg_current */
    ...
} /* switching_group */
...
} /* dynamic_current */
...
} /* cell */

Simple Attributes
related_inputs
related_outputs
typical_capacitances

when

Group Statement
switching_group

---

**ff, latch, ff_bank, and latch_bank Groups**

The **ff, latch, ff_bank, and latch_bank** groups define sequential blocks. These groups are defined at the cell level. One or more groups can be specified within a cell group.

**reference_pin_names Variable**

The optional, user-defined **reference_pin_names** variable specifies internal reference input nodes used within the **ff, latch, ff_bank, or latch_bank** groups. If the **reference_pin_names** variable is not specified, the node names used within the **ff, latch, ff_bank, or latch_bank** group are assumed to be actual pin or bus names within the cell.

**variable1 and variable2 Variables**

The **variable1 and variable2** variables define internal reference output nodes. The **variable1 and variable2** values in those groups must be unique within a cell.
bits Variable
The bits variable defines the width of the ff_bank and latch_bank component.

related_inputs Simple Attribute
This attribute defines the input condition of input pins. If only one input is switching during the time period, the input condition is defined as “single input event.” If more than one input pin is switching, the input condition is defined as “multiple input events.”

• This attribute is required.
• A list of input pins can be specified in the attribute.
• Because “single input event” is supported, exactly one of the input pins in the list must be toggling to match the input condition.
• “Multiple input events” are not supported.
• The pins in the list can be in any order.
• Bus and bundle are supported.

Syntax
related_inputs : input_pin_name;

input_pin_name
Name of input pin.

Example
related_inputs : A ;

related_outputs Simple Attribute
The related_outputs attribute defines the output condition of specified output pins. If no toggling output occurs as a trigger event is given, the condition is called a “nonpropagating event.” If an event propagates through the cell and causes at least one output toggling, then it is called a “propagating event.”

• This attribute is optional.
• A list of output pins can be specified in the attribute.
• A “single input event” matches the output condition for all toggling output pins in the list.
• The pins in the list can be in any order.
• Bus and bundle are supported only in bit level.

• For a standard cell, if the attribute is specified, it represents a “propagating event.” Otherwise, if it is missing, it represents a “nonpropagating event.”

• There is no related_outputs attribute for macro cells. Therefore, you do not need to distinguish between nonpropagating and propagating event tables.

Syntax
related_outputs : output_pin_name ;

output_pin_name
Name of output pin.

Example
related_outputs : D ;

typical_capacitances Simple Attribute
The typical_capacitances attribute specifies the values of the capacitance for all the output pins specified in the related_outputs attribute. The values are specified in the order of the corresponding output pins specified by the related_outputs attribute. For example:

... /* the fixed capacitance of Q1 is 10.0, Q2 is 20.0, and Q3 is 30.0. */ related_outputs : "Q1 Q2 Q3"; typical_capacitances(10.0 20.0 30.0); ...

The attribute is required for cross type.

If data in the vector group is not defined as a sparse cross table, the specified values in the attribute are ignored.

Syntax
typical_capacitances ("float, ...");

float
Value of capacitance on pin.

Example
typical_capacitances (10.0 20.0);
**when Simple Attribute**

Use the `when` attribute to specify a state-dependent condition that determines whether the instantaneous power data can be accessed.

**Syntax**

```
when : boolean expression
```

*boolean expression*

Expression determines whether the instantaneous power data is accessed.

---

**switching_group Group**

Use the `switching_group` group to specify a current waveform vector when the power and ground current is dependent on pin switching conditions.

```
library (name) {
  cell (name) {
    dynamic_current () {
      ...
    
    switching_group() {
      ...
    }
  }
}
```

**Simple Attributes**

- `input_switching_condition`
- `output_switching_condition`
- `min_input_switching_count`
- `max_input_switching_count`

**Group**

`pg_current`

---

**input_switching_condition Simple Attribute**

The `input_switching_condition` attribute specifies the sense of the toggling input. If more than one `switching_group` group is specified within the `dynamic_current` group, you can place the attribute in any order.

The valid values are `rise` and `fall`. `rise` represents a rising pin and `fall` represents a falling pin.
Syntax
input_switching_condition (enum(rise, fall));

`enum(rise, fall)`
Enumerated type specifying the rise or fall condition.

Example
input_switching_condition (rise);

---

**output_switching_condition Simple Attribute**

Use the `output_switching_condition` attribute to specify the sense of the toggling output. If there is more than one `switching_group` group specified within the `dynamic_current` group, you can place the attribute in any order. The order in the list of the `output_switching_condition` attribute is mapped to the same order of output pins in the `related_outputs` attribute.

The valid values are `rise` and `fall`. `rise` represents a rising pin and `fall` represents a falling pin.

Syntax
output_switching_condition (enum(rise, fall));

`enum(rise, fall)`
Enumerated type specifying the rise or fall condition.

Example
output_switching_condition (rise, fall);

---

**min_input_switching_count Simple Attribute**

The `min_input_switching_count` attribute specifies the minimum number of bits in the input bus that are switching simultaneously. The following applies to the `min_input_switching_count` attribute:

- The count must be an integer.
- The count must be greater than 0 and less than the `max_input_switching_count` value.

Syntax
```
switching_group() {
    min_input_switching_count : integer ;
```
max_input_switching_count : integer;

...}

Example
switching_group() {
    min_input_switching_count : 1 ;
    max_input_switching_count : 3 ;
    ...
}

max_input_switching_count Attribute
The max_input_switching_count attribute specifies the maximum number of bits in the
input bus that are switching simultaneously. The following applies to the
max_input_switching_count attribute:

• The count must be an integer.
• The count must be greater than the min_input_switching_count value.
• The count within a dynamic_current should cover the total number of input bits
  specified in related_inputs.

Syntax
switching_group() {
    min_input_switching_count : integer ;
    max_input_switching_count : integer ;
    ...
}

Example
switching_group() {
    min_input_switching_count : 1 ;
    max_input_switching_count : 3 ;
    ...
}

pg_current Group
Use the pg_current group to specify current waveform data in a vector group. If all vectors
under the group are dense, data in this group is represented as a dense table. If all vectors
under the group are sparse in cross type, data in this group is represented as a sparse cross
table. If all vectors under the group are sparse in diagonal type, data in this group is
represented as a sparse diagonal table.

library (name) {

cell (name) {
    dynamic_current () {
        ...
        switching_group() {
            ...
            pg_current () {}
            ...
        }
    }
}

Group

vector

compact_ccs_power Group

The compact_ccs_power group contains a detailed description for compact CCS power data. The compact_ccs_power group includes the following optional attributes: base_curves_group, index_1, index_2, index_3 and index_4. The description for these attributes in the compact_ccs_power group is the same as in the compact_lut_template group. However, the attributes have a higher priority in the compact_ccs_power group. For more information, see “compact_lut_template Group” on page 1-26.

The index_output attribute is also optional. It is used only on cross type tables. For more information about the index_output attribute, see “index_output Simple Attribute” on page 2-59.

library (name) {
    cell(cell_name) {
        dynamic_current() {
            switching_group() {
                pg_current(pg_pin_name) {
                    compact_ccs_power (template_name) {
                        base_curves_group : bc_name;
                        index_output : pin_name;
                        index_1 ("float, ..., float");
                        index_2 ("float, ..., float");
                        index_3 ("float, ..., float");
                        index_4 ("string, ..., string");
                        values ("float | integer, ..., float | integer");
                    } /* end of compact_ccs_power */
                }
            } /* end of pg_current() */
        }
    }
}
Complex Attributes

base_curves_group : bc_name;
index_output : pin_name;
index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
index_4 ("string, ..., string");
values ("float | integer, ..., float | integer");

values Attribute

The values attribute is required in the compact_ccs_power group. The data within the quotation marks (" "), or line, represent the current waveform for one index combination. Each value is determined by the corresponding curve parameter. In the following line,

"t0, c0, 1, t1, c1, 2, t2, c2, 3, t3, c3, 4, t4, c4"

the size is 14 = 8+3*2. Therefore, the curve parameters are as follows:

"init_time, init_current, bc_id1, point_time1, point_current1, bc_id2, point_time2, point_current2, bc_id3, point_time3, point_current3, bc_id4, end_time, end_current"

The elements in the values attribute are floating-point numbers for time and current and integers for the base curve ID. The number of current waveform segments can be different for each slew and load combination, which means that each line size can be different.

Liberty syntax supports tables with varying sizes, as shown:

compact_ccs_power {template_name} {
   ...
   index_1("0.1, 0.2"); /* input_net_transition */
   index_2("1.0, 2.0"); /* total_output_net_capacitance */
   index_3 ("init_time, init_current, bc_id1, point_time1, point_current1, \
   bc_id2, point_time2, point_current2, bc_id3, ..."), \
   end_time, end_current"); /* curve_parameters */
   values ("t0, c0, 1, t1, c1, 2, t2, c2, 3, t3, c3, 4, t4, c4", \
   "t0, c0, 1, t1, c1, 2, t2, c2", \
   "t0, c0, 1, t1, c1, 2, t2, c2, 3, t3, c3", \
   "t0, c0, 1, t1, c1, 2, t2, c2, 3, t3, c3"); /* segment=3 */
}

vector Group

Use the vector group to specify the current waveform for a power and ground pin. This group represents a single current waveform based on specified input slew and output load.
• Data in this group is represented as a dense table, if a template with two
  \texttt{total_output_net_capacitance} variables is applied to the group. If a dense table is
  applied, the order of \texttt{total_output_net_capacitance} variables must map to the order
  of values in the \texttt{related_outputs} attribute.

• Data in this group is represented as a sparse cross table, if the \texttt{index_output} attribute
  is defined in the group.

• Data in this group is represented as a sparse diagonal table, if no \texttt{index_output}
  attribute is defined in the group and a template with exact one
  \texttt{total_output_net_capacitance} variable is applied to the group.

library (name) {
  cell (name) {
    dynamic_current () {
      switching_group() {
        pg_current () {} 
        vector () {
          ...
        }
      }
    }
  }
}

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
\textbf{Simple Attributes} \\
\hline
index_1 (float); \\
index_2 (float); \\
index_3 (float); \\
index_4 (float); \\
index_output : output_pin_name>; \\
reference_time: float; \\
values ("float, ..."); \\
\hline
\end{tabular}
\end{table}

\textbf{index\_1, index\_2, index\_3, and index\_4 Simple Attributes}

The index attributes specify values for variables specified in the \texttt{pg_current_template}. The index value for \texttt{input_net_transition} or \texttt{total_output_net_capacitance} is a
single floating-point number. You create a list of floating-point numbers for the index values
for time. Note the following:

• Different numbers of points are allowed for each waveform.

• If no output or only one output is specified in \texttt{related_outputs}, the table must be
dense.

• If two outputs are specified in \texttt{related_outputs}, the table can be either dense or
sparse.


If more than two outputs are specified, the table must be sparse.

For a cross-type sparse table, a fixed capacitance of all outputs must be specified in typical capacitances. The sweeping output must be specified in index_output, and the varied capacitance of that output must be specified in one of the index attributes. The specified index attribute must map to the total_output_net_capacitance variable in the template.

For a diagonal-type sparse table, capacitances of all outputs are identical and they can be specified in one of the index attributes. The specified index must map to the total_output_net_capacitance variable in the template.

index_output Simple Attribute

This attribute specifies which output capacitance is sweeping while the others are held as fixed values. This attribute is required for cross type. The attribute cannot be defined if the vector table is not defined as a sparse cross table.

Syntax

index_output : output_pin_name;

output_pin_name

Name of the pin that the output capacitance is sweeping.

Example

index_output : “QN”;

reference_time Simple Attribute

This attribute represents the time at which the input waveform crosses the reference voltage.

Syntax

reference_time : float;

float

Specifies the time at which the input waveform crosses the reference voltage.

Example

reference_time : 0.01;
values Simple Attribute
The `values` attribute defines a list of floating-point numbers that represent the dynamic current waveform of a specified power and ground pin.

Syntax
```
values:("float, ...");
```

`float`
Defines a list of floating-point numbers that represent the dynamic current waveform of a specified power and ground pin.

Example
```
values : ("0.002, 0.009, 0.134, 0.546");
```

ff Group
The `ff` group describes either a single-stage or a master-slave flip-flop in a cell or test cell. The syntax for a cell is shown here. For information about the `test_cell` group, see “test_cell Group” on page 2-118.

Syntax
```
library (namestring) {
  cell (namestring) {
    ff (variable1string, variable2string) {
      ... flip-flop description ...
    }
  }
}
```

The `variable1` value is the state of the noninverting output of the flip-flop; the `variable2` value is the state of the inverting output. The `variable1` value can be considered the 1-bit storage of the flip-flop. Valid values for `variable1` and `variable2` are anything except a pin name used in the cell being described. Both of these variables must be assigned, even if one of them is not connected to a primary output pin.

Simple Attributes
```
clear : "Boolean expression" ;
clear_preset_var1 : L | H | N | T | X ;
clear_preset_var2 : L | H | N | T | X ;
clocked_on : "Boolean expression" ;
clocked_on_also : "Boolean expression" ;
next_state : "Boolean expression" ;
preset : "Boolean expression" ;
```
clear Simple Attribute

The `clear` attribute gives the active value for the clear input.

Syntax

```
clear : "Boolean expression" ;
```

Example

```
clear : "CD’" ;
```

“Single-Stage Flip-Flop” on page 2-63 contains more information about the `clear` attribute.

clear_preset_var1 Simple Attribute

The `clear_preset_var1` attribute gives the value that `variable1` has when clear and preset are both active at the same time.

Syntax

```
clear_preset_var1 : L | H | N | T | X ;
```

Example

```
clear_preset_var1 : H ;
```

Table 2-5 shows the valid variable values for the `clear_preset_var1` simple attribute.

<table>
<thead>
<tr>
<th>Variable values</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>No change$^1$</td>
</tr>
<tr>
<td>T</td>
<td>Toggle the current value from 1 to 0, 0 to 1, or X to X$^1$</td>
</tr>
<tr>
<td>X</td>
<td>Unknown$^1$</td>
</tr>
</tbody>
</table>

1. Use these values to generate VHDL models.

“Single-Stage Flip-Flop” on page 2-63 contains more information about the `clear_preset_var1` attribute, including its function and values.
clear_preset_var2 Simple Attribute

The clear_preset_var2 attribute gives the value that variable2 has when clear and preset are both active at the same time.

Syntax

clear_preset_var2 : L | H | N | T | X ;

Example

clear_preset_var2 : L ;

"Single-Stage Flip-Flop" on page 2-63 contains more information about the clear_preset_var2 attribute, including its function and values.

clocked_on and clocked_on_also Simple Attributes

The clocked_on and clocked_on_also attributes identify the active edge of the clock signals and are required in all ff groups. For example, use clocked_on : "CP" to describe a rising-edge-triggered device and use clocked_on_also : "CP" for a falling-edge-triggered device.

Note:

A single-stage flip-flop does not use clocked_on_also. See "Single-Stage Flip-Flop" on page 2-63 for details.

When describing flip-flops that require both a master clock and a slave clock, use the clocked_on attribute for the master clock and the clocked_on_also attribute for the slave clock.

Syntax

clocked_on : "Boolean expression" ;
clocked_on_also : "Boolean expression" ;

Boolean expression

Active edge of a clock signal.

Example

clocked_on : "CP" ;
clocked_on_also : "CP’" ;

Syntax

next_state : "Boolean expression" ;

The following example shows an ff group for a single-stage D flip-flop.
The example defines two variables, IQ and IQN. The next_state equation determines the value of IQ after the next active transition of the clocked_on attribute. In this example, IQ is assigned the value of the D input.

In some flip-flops, the next state depends on the current state. In this case, the first state variable (IQ in the example) can be used in the next_state statement; the second state variable, IQN, cannot.

For example, the ff declaration for a JK flip-flop looks like this:

```plaintext
ff(IQ, IQN) {
    next_state : "(J K IQ') + (J K') + (J' K' IQ)" ;
    clocked_on : "CP" ;
}
```

The next_state and clocked_on attributes completely define the synchronous behavior of the flip-flop.

preset Simple Attribute

The preset attribute gives the active value for the preset input.

Syntax

```plaintext
preset : "Boolean expression" ;
```

Example

```plaintext
preset : "PD'" ;
```

Single-Stage Flip-Flop

A single-stage flip-flop does not use the optional clocked_on_also attribute.

The clear attribute gives the active value for the clear input. The preset attribute gives the active value for the preset input. For example, the following statement defines an active-low clear signal:

```plaintext
clear : "CD'" ;
```
Table 2-6 shows the functions of the attributes in the \texttt{ff} group for a single-stage flip-flop.

### Table 2-6  Function Table for a Single-Stage Flip-Flop

<table>
<thead>
<tr>
<th>clocked_on</th>
<th>clear</th>
<th>preset</th>
<th>variable1</th>
<th>variable2</th>
</tr>
</thead>
<tbody>
<tr>
<td>active edge</td>
<td>inactive</td>
<td>inactive</td>
<td>next_state</td>
<td>!next_state</td>
</tr>
<tr>
<td>--</td>
<td>active</td>
<td>inactive</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>--</td>
<td>inactive</td>
<td>active</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>--</td>
<td>active</td>
<td>active</td>
<td>clear_preset_var1</td>
<td>clear_preset_var2</td>
</tr>
</tbody>
</table>

The \texttt{clear_preset_var1} and \texttt{clear_preset_var2} attributes give the value that \texttt{variable1} and \texttt{variable2} have when \texttt{clear} and \texttt{preset} are both active at the same time. See Table 2-6 for the valid variable values.

If the \texttt{clear} and \texttt{preset} attributes are both included in the group, either \texttt{clear_preset_var1}, \texttt{clear_preset_var2}, or both must be defined. Conversely, if either \texttt{clear_preset_var1}, or \texttt{clear_preset_var2}, or both are included, both \texttt{clear} and \texttt{preset} must be defined.

The flip-flop cell is activated whenever the value of \texttt{clear}, \texttt{preset}, \texttt{clocked_on}, or \texttt{clocked_on_also} changes.

**Example 2-7** is an \texttt{ff} group for a single-stage D flip-flop with rising-edge sampling, negative clear and preset, and output pins set to 0 when both clear and preset are active (low).

**Example 2-7  Single-Stage D Flip-Flop**

\begin{verbatim}
ff(IQ, IQN) {
    next_state : "D" ;
    clocked_on : "CP" ;
    clear : "CD'" ;
    preset : "PD'" ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}
\end{verbatim}

**Example 2-8** is an \texttt{ff} group for a single-stage, rising-edge-triggered JK flip-flop with scan input, negative clear and preset, and output pins set to 0 when clear and preset are both active.
Example 2-8  Single-Stage JK Flip-Flop

```
ff(IQ, IQN) {
    next_state :
        "(TE*TI)+(TE'*J*K')+(TE'*J'*K'*IQ)+(TE'*J*K*IQ')" ;
    clocked_on : "CP" ;
    clear : "CD’" ;
    preset : "PD’" ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}
```

Example 2-9 is an ff group for a D flip-flop with synchronous negative clear.

Example 2-9  D Flip-Flop With Synchronous Negative Clear

```
ff (IQ, IQN) {
    next_state : "D * CLR’" ;
    clocked_on : "CP" ;
}
```

Master-Slave Flip-Flop

The syntax for a master-slave flip-flop is the same as for a single-stage device, except that it includes the clocked_on_also attribute. Table 2-7 shows the functions of the attributes in the ff group for a master-slave flip-flop.

The internal1 and internal2 variables represent the output values of the master stage, and variable1 and variable2 represent the output values of the slave stage. The variable1 and variable2 variables have the same value as internal1 and internal2, respectively, when clear and preset are both active at the same time.

Table 2-7  Function Table for a Master-Slave Flip-Flop

<table>
<thead>
<tr>
<th>Variable</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>active</td>
</tr>
<tr>
<td>preset</td>
<td>inactive</td>
</tr>
<tr>
<td>internal1</td>
<td>clear_preset_var1</td>
</tr>
<tr>
<td>internal2</td>
<td>clear_preset_var2</td>
</tr>
<tr>
<td>variable1</td>
<td>clear_preset_var1</td>
</tr>
<tr>
<td>variable2</td>
<td>clear_preset_var2</td>
</tr>
<tr>
<td>active edge</td>
<td>clocked_on</td>
</tr>
<tr>
<td></td>
<td>clocked_on_also</td>
</tr>
</tbody>
</table>

Chapter 2: cell and model Group Description and Syntax
Group Statements
Example 2-10 shows an ff group for a master-slave D flip-flop with rising-edge sampling, falling-edge data transfer, negative clear and preset, and output values set high when clear and preset are both active.

Example 2-10  Master-Slave D Flip-Flop

```vhdl
ff(IQ, IQN) {
  next_state : "D" ;
  clocked_on : "CLK" ;
  clocked_on_also : "CLKN'" ;
  clear : "CDN'" ;
  preset : "PDN'" ;
  clear_preset_var1 : H ;
  clear_preset_var2 : H ;
}
```

**next_state Simple Attribute**

Required in all ff groups, next_state is a logic equation written in terms of the cell’s input pins or the first state variable, variable1. For single-stage storage elements, the next_state attribute equation determines the value of variable1 at the next active transition of the clocked_on attribute.

For devices such as a master-slave flip-flop, the next_state equation determines the value of the master stage’s output signals at the next active transition of the clocked_on attribute.

The type of pin that appears in the Boolean expression of a next_state attribute is defined in a pin group with the nextstate_type attribute.

### ff_bank Group

An ff_bank group is defined within a cell or test_cell group, as shown in the following syntax.

The ff_bank group describes a cell that is a collection of parallel, single-bit sequential parts. Each part can share control signals with the other parts and performs an identical function. The ff_bank group is typically used to represent multibit registers in cell and test_cell groups. For information about ff_bank in test cells, see “test_cell Group” on page 2-118.

The syntax for the ff_bank group is similar to that of the ff group.

**Syntax**

```
library (namestring) {
  cell (namestring) {
    ff_bank (variable1string, variable2string, bitsinteger) {
      ... multibit flip-flop register description ...
    }
  }
}
```
Simple Attributes

clocked_on : "Boolean expression" ;
next_state : "Boolean expression" ;
clear : "Boolean expression" ;
preset : "Boolean expression" ;
clear_preset_var1 : L | H | N | T | X ;
clear_preset_var2 : L | H | N | T | X ;
clocked_on_also : "Boolean expression" ;

Example 2-11 on page 2-71 shows an ff_bank group for a multibit D flip-flop.

An input described in a pin group, such as the clk input, is fanned out to each flip-flop in the bank. Each primary output must be described in a bus or bundle group, whose function statement must include either variable1 or variable2.

clocked_on and clocked_on_also Simple Attributes

Required in all ff_bank groups, the clocked_on and clocked_on_also attributes identify the active edge of the clock signal.

When describing flip-flops that require both a master and a slave clock, use the clocked_on attribute for the master clock and the clocked_on_also attribute for the slave clock.

Syntax

clocked_on : "Boolean expression" ;
clocked_on_also : "Boolean expression" ;

Boolean expression

Active edge of the edge-triggered device.

Examples

clocked_on : "CP" ; /* rising-edge-triggered device */
clocked_on_also : "CP’" ; /* falling-edge-triggered device */

next_state Simple Attribute

Required in all ff_bank groups, the next_state attribute is a logic equation written in terms of the cell’s input pins or the first state variable, variable1. For single-stage flip-flops, the next_state attribute equation determines the value of variable1 at the next active transition of the clocked_on attribute.

For devices such as master-slave flip-flops, the next_state equation determines the value of the master stage’s output signals at the next active transition of the clocked_on attribute.

Syntax

next_state : "Boolean expression" ;
**Boolean expression**

Identifies the active edge of the clock signal.

**Example**

```
next_state : "D" ;
```

The type of a `next_state` attribute is defined in a pin group with the `nextstate_type` attribute.

**clear Simple Attribute**

The `clear` attribute gives the active value for the clear input.

**Syntax**

```
clear : "Boolean expression" ;
```

**Example**

```
clear : "CD'" ;
```

See “Single-Stage Flip-Flop” on page 2-63 for more information about the `clear` attribute.

**preset Simple Attribute**

The `preset` attribute gives the active value for the preset input.

**Syntax**

```
preset : "Boolean expression" ;
```

**Example**

```
preset : "PD'" ;
```

See “Single-Stage Flip-Flop” on page 2-63 for more information about the `preset` attribute.

**clear_preset_var1 Simple Attribute**

The `clear_preset_var1` attribute gives the value that `variable1` has when clear and preset are both active at the same time.

**Syntax**

```
clear_preset_var1 : L | H | N | T | X ;
```

**Example**

```
clear_preset_var1 : L ;
```

See “Single-Stage Flip-Flop” on page 2-63 for more information about the `clear_preset_var1` attribute, including its function and values.
Table 2-8 shows the valid variable values for the \texttt{clear\_preset\_var1} attribute.

<table>
<thead>
<tr>
<th>Variable values</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>No change$^1$</td>
</tr>
<tr>
<td>T</td>
<td>Toggle the current value from 1 to 0, 0 to 1, or X to X$^1$</td>
</tr>
<tr>
<td>X</td>
<td>Unknown$^1$</td>
</tr>
</tbody>
</table>

1. Use these values to generate VHDL models.

clear\_preset\_var2 Simple Attribute

The \texttt{clear\_preset\_var2} attribute gives the value that \texttt{variable2} has when \texttt{clear} and \texttt{preset} are both active at the same time. Table 2-8 shows the valid variable values for the \texttt{clear\_preset\_var2} attribute.

Syntax

clear\_preset\_var2 : L | H | N | T | X ;

Example

clear\_preset\_var2 : L ;

See “Single-Stage Flip-Flop” on page 2-63 for more information about the \texttt{clear\_preset\_var1} attribute, including its function and values.

Multibit Flip-Flop

The \texttt{bits} value in the \texttt{ff\_bank} definition is the number of bits in this multibit cell.

Syntax

```plaintext
library (name\_string) { cell (name\_string) { ff\_bank (variable\_1\_string, variable\_2\_string, bits\_integer) { ... multibit flip-flop register description ... } } }
```
A multibit register containing four rising-edge-triggered D flip-flops with clear and preset is shown in Figure 2-1 and Example 2-11.

Figure 2-1  Multibit Register
**Example 2-11  Multibit Register**

cell (dff4) {
  area : 1 ;
  pin (CLK) {
    direction : input ;
    capacitance : 0 ;
    min_pulse_width_low : 3 ;
    min_pulse_width_high : 3 ;
  }
  bundle (D) {
    members(D1, D2, D3, D4);
    nextstate_type : data;
    direction : input ;
    capacitance : 0 ;
    timing() {
      related_pin : "CLK" ;
      timing_type : setup_rising ;
      cell_rise(scalar) {
        values (" 1.0 ") ;
      }
      cell_fall(scalar) {
        values (" 1.0 ") ;
      }
    }
    timing() {
      related_pin : "CLK" ;
      timing_type : hold_rising ;
      cell_rise(scalar) {
        values (" 1.0 ") ;
      }
      cell_fall(scalar) {
        values (" 1.0 ") ;
      }
    }
    timing() {
      related_pin : "CLK" ;
      timing_type : recovery_rising ;
      cell_rise(scalar) {
        values (" 1.0 ") ;
      }
      cell_fall(scalar) {
        values (" 1.0 ") ;
      }
    }
  }
  pin (CLR) {
    direction : input ;
    capacitance : 0 ;
    timing() {
      related_pin : "CLK" ;
      timing_type : setup_rising ;
      cell_rise(scalar) {
        values (" 1.0 ") ;
      }
      cell_fall(scalar) {
        values (" 1.0 ") ;
      }
    }
  }
}

Chapter 2: cell and model Group Description and Syntax
Group Statements
timing() {
    related_pin     : "CLK" ;
timing_type     : recovery_rising ;
cell_rise(scalar) {
    values (" 1.0 ") ;
}
cell_fall(scalar) {
    values (" 1.0 ") ;
}
}

ff_bank (IQ, IQN, 4) {
    next_state : "D" ;
clocked_on : "CLK" ;
clear : "CLR'" ;
preset : "PRE’" ;
clear_preset_var1 : L ;
clear_preset_var2 : L ;
}

bundle (Q) {
    members(Q1, Q2, Q3, Q4);
direction : output ;
function : "(IQ)" ;
timing() {
    related_pin     : "CLK" ;
timing_type     : rising_edge ;
cell_rise(scalar) {
    values (" 2.0 ") ;
}
cell_fall(scalar) {
    values (" 2.0 ") ;
}
}

bundle (QN) {
    members(Q1N, Q2N, Q3N, Q4N);
direction : output;
function : "IQN";
timing() {
    related_pin : "CLK";
timing_type : rising_edge;
cell_rise(scalar) {
    values (" 2.0 ");
}
cell_fall(scalar) {
    values (" 2.0 ");
}
}
timing() {
    related_pin : "PRE";
timing_type : clear;
timing_sense : positive_unate;
cell_fall(scalar) {
    values (" 1.0 ");
}
}
timing() {
    related_pin : "CLR";
timing_type : preset;
timing_sense : negative_unate;
cell_rise(scalar) {
    values (" 1.0 ");
}
}
} /* end of cell dff4 */

fpga_condition Group

An fpga_condition group declares an fpga_condition group containing several fpga_condition_value groups.

Syntax

cell (name_id) {
    fpga_condition (name_id) {
        ...
    }
}

name

    Specifies the name of the fpga_condition group.

Group

fpga_condition_value
fpga_condition_value Group

The fpga_condition_value group specifies a condition.

Syntax

cell (nameid) {
    fpga_condition (condition_group_nameid) {
        fpga_condition_value (condition_nameid) {
            ...
        }
    }
}

condition_name

Specifies the name of a condition.

Simple Attribute

fpga_arc_condition

fpga_arc_condition Simple Attribute

The fpga_arc_condition attribute specifies a Boolean condition that enables the associated fpga_condition_value group.

Syntax

cell (name_string) {
    fpga_condition (nameid) {
        fpga_condition_value (condition_nameid) {
            fpga_arc_condition : conditionBoolean ;
        }
    }
}

condition

Specifies a Boolean condition. Valid values are true and false.

Example

fpga_arc_condition : true ;
**generated_clock Group**

A generated_clock group is defined within a cell group or a model group to describe a new clock that is generated from a master clock by

- Clock frequency division
- Clock frequency multiplication
- Edge derivation

**Syntax**

```
cell (name_string) {
    generated_clock (name_string) {
        ...clock data...
    }
}
```

**Simple Attributes**

- `clock_pin : "name1 [name2 name3 ... ]"`;
- `master_pin : name`;
- `divided_by : integer`;
- `multiplied_by : integer`;
- `invert : Boolean`;
- `duty_cycle : float`;

**Complex Attributes**

- `edges`
- `shifts`

**clock_pin Simple Attribute**

The clock_pin attribute identifies a pin connected to a master clock signal.

**Syntax**

```
clock_pin : "name1 [name2 name3 ... ]";
```

**Example**

```
clock_pin : "clk1 clk2 clk3";
```

**master_pin Simple Attribute**

The master_pin attribute identifies a pin connected to an input clock signal.

**Syntax**

```
master_pin : name;
```
Example
master_pin : clk;

**divided_by Simple Attribute**

The `divided_by` attribute specifies the frequency division factor, which must be a power of 2.

**Syntax**

divided_by : integer;

Example

generated_clock(genclk1) {
    clock_pin : clk1;
    master_pin : clk;
    divided_by : 2;
    invert : true;
}

This code fragment shows a clock pin (clk1) generated by dividing the original clock pin (clk) frequency by 2 and then inverting the result.

**multiplied_by Simple Attribute**

The `multiplied_by` attribute specifies the frequency multiplication factor, which must be a power of 2.

**Syntax**

multiplied_by : integer;

Example

generated_clock(genclk2) {
    clock_pin : clk1;
    master_pin : clk;
    multiplied_by : 2;
    duty_cycle : 50.0;
}

This code fragment shows a clock pin (clk1) generated by multiplying the original clock pin (clk) frequency by 2, with a duty cycle of 50.

**invert Simple Attribute**

The `invert` attribute inverts the waveform generated by multiplication or division. Set this attribute to true to invert the waveform. Set it to false if you do not want to invert the waveform.
Syntax
invert : Boolean ;

Example
invert : true;

duty_cycle Simple Attribute
The duty_cycle attribute specifies the duty cycle, in percentage, if frequency multiplication is used. This is a number between 0.0 and 100.0. The duty cycle is the high pulse width.

Syntax
duty_cycle : float ;

Example
duty_cycle : 50.0;

edges Complex Attribute
The edges attribute specifies a list of the edges from the master clock that form the edges of the generated clock. Use this option when simple division or multiplication is insufficient to describe the generated clock waveform. The number of edges must be an odd number that is greater than or equal to 3. The first edge must be greater than or equal to 1.

Syntax
edges (edge1, edge2, edge3);

Example
edges (1, 3, 5);

shifts Complex Attribute
The shifts attribute specifies the shifts (in time units) to be added to the edges specified in the edge list to generate the clock. The number of shifts must equal the number of edges. This shift modifies the ideal clock edges; it is not considered to be clock latency.

Syntax
shifts (shift1, shift2, shift3);

Example
shifts (5.0, -5.0, 0.0);

Example 2-12 shows a generated clock description.

Example 2-12 Description of a Generated Clock
cell(acell) {
  ...
}
intrinsic_parasitic Group

The intrinsic_parasitic group specifies the state-dependent intrinsic capacitance and intrinsic resistance of a cell.

Syntax

```
library( library_name ) {
    ........
    lu_table_template ( template_name ) {
        variable_1 : pg_voltage | pg_voltage_difference;
        index_1 ( "float, ..., float" );
    }
}
cell (cell_name) {
    mode_definition (mode_name) {
        mode_value (mode_value) {
            when : boolean_expression ;
            sdf_cond : boolean_expression ;
        }
    }
}
```
...}

intrinsic_parasitic () {
    mode (mode_name, mode_value) ;
    when : boolean expression ;
    intrinsic_resistance(pg_pin_name) {
        related_output : output_pin_name ;
        value : float ;
        reference_pg_pin : pg_pin_name ;
        lut_values ( template_name ) {
            index_1 ("float, ... float" );
            values ("float, ... float" );
        }
    }
    intrinsic_capacitance(pg_pin_name) {
        value : float ;
        reference_pg_pin : pg_pin_name ;
        lut_values ( template_name ) {
            index_1 ("float, ... float" );
            values ("float, ... float" );
        }
    }
}

Simple Attributes
when
reference_pg_pin

Complex Attribute
mode

Groups
intrinsic_capacitance
intrinsic_resistance
total_capacitance

when Simple Attribute
The when attribute specifies the state-dependent condition that determines whether the intrinsic parameters are accessed. The when attribute is used when all the state conditions of a cell are specified. The default intrinsic_parasitic group is not state-dependent, and is defined without the when attribute. If some of the state conditions of the cell are missing, the default intrinsic_parasitic group is used. However, if some state conditions of the cell are missing and no default state is provided, the value of the intrinsic resistance is considered to be infinite, and the value of the intrinsic capacitance is considered to be zero.
Syntax
when : boolean_expression ;

boolean expression
    Specifies the state-dependent condition.

Example
when : "A & B" ;

reference_pg_pin Simple Attribute

The reference_pg_pin attribute specifies the reference pin for the
intrinsic_resistance and intrinsic_capacitance groups. The reference pin must be
a valid PG pin.

Syntax
reference_pg_pin : pg_pin_name ;

Example
reference_pg_pin : G1 ;

mode Complex Attribute

The mode attribute pertains to an individual cell. The cell is active when the mode attribute is
instantiated with a name and a value. You can specify multiple instances of this attribute.
However, specify only one instance for each cell.

Define the mode attribute within an intrinsic_parasitic group.

Syntax
mode ( mode_name, mode_value ) ;

Example
mode (rw, read) ;

intrinsic_capacitance Group

Use this group to specify the intrinsic capacitance of a cell.

Syntax
intrinsic_parasitic () {
    intrinsic_capacitance (pg_pin_name) {
        value : float ;
        reference_pg_pin : pg_pin_name;
        lut_values ( template_name ) {
            index_1 ("float, ... float");
            values ("float, ... float");
    }
The `pg_pin_name` specifies a power and ground pin where the capacitance is derived.

You can have more than one `intrinsic_capacitance` group. You can place these groups in any order within an `intrinsic_parasitic` group.

**Simple Attributes**

**value**

**reference_pg_pin**

**Group**

**lut_values**

**value Simple Attribute**

The `value` attribute specifies the value of the intrinsic capacitance. By default, the intrinsic capacitance value is zero.

**Syntax**

```
value : float;
```

**Example**

```
value : 5;
```

**reference_pg_pin Simple Attribute**

The `reference_pg_pin` attribute specifies the reference pin for the `intrinsic_resistance` and `intrinsic_capacitance` groups. The reference pin must be a valid PG pin.

**Syntax**

```
reference_pg_pin : pg_pin_name;
```

**Example**

```
reference_pg_pin : G1;
```

**lut_values Group**

Voltage-dependent intrinsic parasitics are modeled by lookup tables. A lookup table consists of intrinsic parasitic values for different values of VDD. To use these lookup tables, define the `lut_values` group. You can add the `lut_values` group to both the `intrinsic_resistance` and `intrinsic_capacitance` groups. The `lut_values` group
uses the *variable_1* variable, which is defined within the *lu_table_template* group, at the library level. The valid values of the *variable_1* variable are *pg_voltage* and *pg_voltage_difference*.

**Syntax**

```
lut_values ( template_name ) {
  index_1 ("float, ... float");
  values ("float, ... float");
}
```

*template_name*

The name of the lookup table template.

**Example**

```
lut_values ( test_voltage ) {
  index_1 ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
  values ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
}
```

**intrinsic_resistance Group**

Use this group to specify the intrinsic resistance between a power pin and an output pin of a cell.

**Syntax**

```
intrinsic_parasitic () {
  intrinsic_resistance ( pg_pin_name ) {
    related_output : output_pin_name;
    value : float;
    reference_pg_pin : pg_pin_name;
    lut_values ( template_name ) {
      index_1 ("float, ... float");
      values ("float, ... float");
    }
  }
}
```

The *pg_pin_name* specifies a power or ground pin. You can place the *intrinsic_resistance* groups in any order within an *intrinsic_parasitic* group. If some of the *intrinsic_resistance* group is not defined, the value of resistance defaults to +infinity. The channel connection between the power and ground pins and the output pin is defined as a closed channel if the resistance value is greater than 1 megaohm. Otherwise, the channel is opened. The *intrinsic_resistance* group is not required if the channel is closed.

**Simple Attributes**

*related_output*
value
reference_pg_pin

Group
lut_values

related_output Simple Attribute
Use this attribute to specify the output pin.

Syntax
related_output : output_pin_name ;

output_pin_name
The name of the output pin.

Example
related_output : "A & B" ;

value Simple Attribute
Specifies the value of the intrinsic resistance. If this attribute is not defined, the value of the intrinsic resistance defaults to +infinity.

Syntax
value : float;

Example
value : 5;

reference_pg_pin Simple Attribute
The reference_pg_pin attribute specifies the reference pin for the intrinsic_resistance and intrinsic_capacitance groups. The reference pin must be a valid PG pin.

Syntax
reference_pg_pin : pg_pin_name ;

Example
reference_pg_pin : G1 ;
**lut_values Group**

Voltage-dependent intrinsic parasitics are modeled by lookup tables. A lookup table consists of intrinsic parasitic values for different values of VDD. To use these lookup tables, define the lut_values group. You can add the lut_values group to both the intrinsic_resistance and intrinsic_capacitance groups. The lut_values group uses the variable_1 variable, which is defined within the lu_table_template group, at the library level.

**Syntax**

```plaintext
lut_values ( template_name ) {
    index_1 ("float, ... float");
    values ("float, ... float");
}
```

*template_name*

The name of the lookup table template.

**Example**

```plaintext
lut_values ( test_voltage ) {
    index_1 ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
    values ("0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0");
}
```

---

**total_capacitance Group**

The total_capacitance group specifies the macro cell’s total capacitance on a power or ground net within the intrinsic_parasitic group. The following applies to the total_capacitance group:

- The total_capacitance group can be placed in any order if there is more than one total_capacitance group within an intrinsic_parasitic group.
- The total capacitance parasitics modeling in macro cells is not state dependent, which means that there is no state condition specified in intrinsic_parasitic.

**Syntax**

```plaintext
cell (cell_name) {
    ...
    intrinsic_parasitic () {
        total_capacitance (pg_pin_name) {
            value : float ;
        }
    }
    ...
    ...
}
```
Example

cell (my_cell) {
  ...
  intrinsic_parasitic () {
    total_capacitance (VDD) {
      value : 0.2 ;
    }
  }
  ...
  ...
}

latch Group

A latch group is defined within a cell, model, or test_cell group to describe a level-sensitive memory device. The syntax for defining a latch group within a cell group is shown here. For information about test cells, see “test_cell Group” on page 2-118.

library (name_string) {
  cell (name_string) {
    latch (variable1_string, variable2_string) {
      ... latch description ...
    }
  }
}

The variable1 value is the state of the noninverting output of the latch; the variable2 value is the state of the inverting output. The variable1 value is considered the 1-bit storage of the latch. You can name variable1 and variable2 anything except a pin name used in the cell being described. Both values are required, even if one of them is not connected to a primary output pin.

Simple Attributes

clear : "Boolean expression" ;
clear_preset_var1 : L | H | N | T | X ;
clear_preset_var2 : L | H | N | T | X ;
data_in : "Boolean expression" ;
enable : "Boolean expression" ;
enable_also : "Boolean expression" ;
preset : "Boolean expression" ;

clear Simple Attribute

The clear attribute gives the active value for the clear input.

Syntax

clear : value_Boolean ;
Example

The following example defines a low-active clear signal.

clear : "CD’" ;

clear_preset_var1 and clear_preset_var2 Simple Attributes

The clear_preset_var1 and clear_preset_var2 attributes give the value that variable1 and variable2 have when clear and preset are both active at the same time.

Syntax

clear_preset_var1 : L | H | N | T | X ;
clear_preset_var2 : L | H | N | T | X ;

Table 2-9 shows the valid values for the clear_preset_var1 and clear_preset_var2 attributes.

Table 2-9 Valid Values for the clear_preset_var1 and clear_preset_var2 Attributes

<table>
<thead>
<tr>
<th>Variable values</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>No change¹</td>
</tr>
<tr>
<td>T</td>
<td>Toggle the current value from 1 to 0, 0 to 1, or X to X¹</td>
</tr>
<tr>
<td>X</td>
<td>Unknown¹</td>
</tr>
</tbody>
</table>

¹. Use these values to generate VHDL models.

See “Single-Stage Flip-Flop” on page 2-63 for more information about the clear_preset_var1 and clear_preset_var2 attributes, including their function and values.

If you include both clear and preset, you must use either clear_preset_var1, clear_preset_var2, or both. Conversely, if you include clear_preset_var1, clear_preset_var2, or both, you must use both clear and preset.
Example
latch (I\text{Q}, I\text{QN}) \{
    clear : "S'" ;
    preset : "R'" ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}\n
data\_in Simple Attribute
The data\_in attribute gives the state of the data input, and the enable attribute gives the state of the enable input. The data\_in and enable attributes are optional, but if you use one of them, you must also use the other.

Syntax
data\_in : value\_Boolean ;

value
State of data input.

Example
data\_in : "D" ;

enable Simple Attribute
The enable attribute gives the state of the enable input, and data\_in attribute gives the state of the data input. The enable and data\_in attributes are optional, but if you use one of them, you must also use the other.

Syntax
enable : value\_Boolean ;

value
State of enable input.

Example
enable : "G" ;

enable\_also Simple Attribute
The enable\_also attribute gives the state of the enable input when you are describing master and slave cells. The enable\_also attribute is optional. If you use enable\_also, you must also use the enable and data\_in attributes.

Syntax
enable\_also : "value\_Boolean " ;
Value

State of enable input for master-slave cells.

Example

enable_also : "G" ;

preset Simple Attribute

The `preset` attribute gives the active value for the preset input.

Syntax

`preset : "valueBoolean " ;`

Example

The following example defines a low-active clear signal.

`preset : "PD’" ;`

Attribute Functions in a latch Group

The latch cell is activated whenever `clear`, `preset`, `enable`, or `data_in` changes.

Table 2-10 shows the functions of the attributes in the `latch` group.

<table>
<thead>
<tr>
<th>enable</th>
<th>clear</th>
<th>preset</th>
<th>variable1</th>
<th>variable2</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>inactive</td>
<td>inactive</td>
<td>data_in</td>
<td>!data_in</td>
</tr>
<tr>
<td>--</td>
<td>active</td>
<td>inactive</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>--</td>
<td>inactive</td>
<td>active</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>--</td>
<td>active</td>
<td>active</td>
<td>clear_preset_var1</td>
<td>clear_preset_var2</td>
</tr>
</tbody>
</table>

Example 2-13 shows a `latch` group for a D latch with active-high enable and negative clear.

Example 2-13  D Latch With Active-High Enable and Negative Clear

```plaintext
latch(IQ, IQN) {
    enable : "G" ;
    data_in : "D" ;
    clear : "CD’" ;
}
```
Example 2-14 shows a latch group for an SR latch. The enable and data_in attributes are not required for an SR latch.

Example 2-14  SR Latch

```plaintext
latch(IQ, IQN) {
    clear : "S" ;
    preset : "R" ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}
```

latch_bank Group

A latch_bank group is defined within a cell, model, or test_cell group to represent multibit latch registers. The syntax for a cell is shown here. For information about test cells, see “test_cell Group” on page 2-118.

The latch_bank group describes a cell that is a collection of parallel, single-bit sequential parts. Each part shares control signals with the other parts and performs an identical function.

An input pin that is described in a pin group, such as the clk input, is fanned out to each latch in the bank. Each primary output must be described in a bus or bundle group, and its function statement must include either variable1 or variable2.

The syntax of the latch_bank group is similar to that of the latch group (see “latch Group” on page 2-85).

Syntax

```plaintext
library (namestring) {
    cell (namestring) {
        latch_bank(variable1string, variable2string, bitsinteger)
        ... multibit latch register description ...
    }
}
```

The bits value in the latch_bank definition is the number of bits in the multibit cell.

Simple Attributes

```plaintext
enable : "Boolean expression" ;
enable_also : "Boolean expression" ;
data_in : "Boolean expression" ;
clear : "Boolean expression" ;
preset : "Boolean expression" ;
clear_preset_var1 : L | H | N | T | X ;
```
clear_preset_var2 : L | H | N | T | X ;

Example 2-15 shows a latch_bank group for a multibit register containing four rising-edge-triggered D latches.

Example 2-15 Multibit D Latch

cell (latch4) {
    area: 16;
    pin (G) {  /* gate enable signal, active-high */
        direction : input;
        ...
    }
    bundle (D) {  /* data input with four member pins */
        members(D1, D2, D3, D4);  /* must be 1st bundle attribute*/
        direction : input;
        ...
    }
    bundle (Q) {
        members(Q1, Q2, Q3, Q4);
        direction : output;
        function : "IQ" ;
        ...
    }
    bundle (QN) {
        members (Q1N, Q2N, Q3N, Q4N);
        direction : output;
        function : "IQN";
        ...
    }
    latch_bank(IQ, IQN, 4) {
        enable : "G" ;
        data_in : "D" ;
    }
    ...
}

clear Simple Attribute

The clear attribute gives the active value for the clear input.

Syntax

clear : "Boolean expression" ;

The following example defines a low-active clear signal.

clear : "CD’" ;

clear_preset_var1 and clear_preset_var2 Simple Attributes

The clear_preset_var1 and clear_preset_var2 attributes give the values that variable1 and variable2 have when clear and preset are both active at the same time.
Syntax

clear_preset_var1 : L | H | N | T | X ;
clear_preset_var2 : L | H | N | T | X ;

Example

See Table 2-9 on page 2-86 for the valid values for the clear_preset_var1 and clear_preset_var2 attributes.

See “Single-Stage Flip-Flop” on page 2-63 for more information about the clear_preset_var1 and clear_preset_var2 attributes, including their function and values.

If you include both clear and preset, you must use either clear_preset_var1, clear_preset_var2, or both. Conversely, if you include clear_preset_var1, clear_preset_var2, or both, you must use both clear and preset.

latch_bank(IQ, IQN) {
    clear : "S’" ;
    preset : "R’" ;
    clear_preset_var1 : L ;
    clear_preset_var2 : L ;
}

data_in Simple Attribute

The data_in attribute gives the state of the data input, and the enable attribute gives the state of the enable input. The enable and data_in attributes are optional, but if you use one of them, you must also use the other.

Syntax

data_in : "Boolean expression" ;

Boolean expression

State of data input.

Example

data_in : "D" ;

enable Simple Attribute

The enable attribute gives the state of the enable input, and the data_in attribute gives the state of the data input. The enable and data_in attributes are optional, but if you use one of them, you must include the other.

Syntax

enable : "Boolean expression" ;
Boolean expression

State of enable input.

Example

```
enable : "G" ;
```

preset Simple Attribute

The preset attribute gives the active value for the preset input.

Syntax

```
preset : "Boolean expression" ;
```

The following example defines a low-active clear signal.

```
preset : "PD'" ;
```

Attribute Functions in a latch_bank Group

The latch_bank cell is activated whenever the value of clear, preset, enable, or data_in attribute changes.

Figure 2-2 and Example 2-16 show a multibit register containing four high-enable D latches with the clear attribute.
Figure 2-2  Multibit Register With Latches
Example 2-16  Multibit Register With Four D Latches

```plaintext
cell (DLT2) {
/* note: 0 hold time */
area : 1 ;
single_bit_degenerate : FDB ;
pin (EN) {
  direction : input ;
capacitance : 0 ;
min_pulse_width_low : 3 ;
min_pulse_width_high : 3 ;
}
bundle (D) {
  members(DA, DB, DC, DD);
direction : input ;
capacitance : 0 ;
timing() {
  related_pin : "EN" ;
timing_type : setup_falling ;
cell_rise(scalar) {
    values (" 1.0 ") ;
  }
cell_fall(scalar) {
    values (" 1.0 ") ;
  }
}
timing() {
  related_pin : "EN" ;
timing_type : hold_falling ;
cell_rise(scalar) {
    values (" 1.0 ") ;
  }
cell_fall(scalar) {
    values (" 1.0 ") ;
  }
}
bundle (CLR) {
  members(CLRA, CLRB, CLRC, CLRD);
direction : input ;
capacitance : 0 ;
timing() {
  related_pin : "EN" ;
timing_type : recovery_falling ;
cell_rise(scalar) {
    values (" 1.0 ") ;
  }
cell_fall(scalar) {
    values (" 1.0 ") ;
  }
}
}
bundle (PRE) {
```
members(PREA, PREB, PREC, PRED);
direction : input;
capacitance : 0;
timing() {
  related_pin : "EN";
timing_type : recovery_falling;
cell_rise(scalar) {
  values (" 1.0 ");
}
cell_fall(scalar) {
  values (" 1.0 ");
}
}
latch_bank(IQ, IQN, 4) {
data_in : "D";
enable : "EN";
clear : "CLR'";
preset : "PRE'";
clear_preset_var1 : H;
clear_preset_var2 : H;
}
bundle (Q) {
members(QA, QB, QC, QD);
direction : output;
function : "IQ";
timing() {
  related_pin : "D";
cell_rise(scalar) {
    values (" 2.0 ");
  }
cell_fall(scalar) {
    values (" 2.0 ");
  }
}
timing() {
  related_pin : "EN";
timing_type : rising_edge;
cell_rise(scalar) {
    values (" 2.0 ");
}
cell_fall(scalar) {
    values (" 2.0 ");
}
}
timing() {
  related_pin : "CLR";
timing_type : clear;
timing_sense : positive_unate;
cell_fall(scalar) {
    values (" 1.0 ");
}
timing() {
    related_pin    : "PRE" ;
    timing_type    : preset ;
    timing_sense   : negative_unate ;
    cell_rise(scalar) {
        values (" 1.0 ");
    }
}

bundle (QN) {
    members(QNA, QNB, QNC, QND);
    direction    : output ;
    function     : "IQN" ;
    timing() {
        related_pin    : "D" ;
        cell_rise(scalar) {
            values (" 2.0 ");
        }
        cell_fall(scalar) {
            values (" 2.0 ");
        }
    }
    timing() {
        related_pin    : "EN" ;
        timing_type    : rising_edge ;
        cell_rise(scalar) {
            values (" 2.0 ");
        }
        cell_fall(scalar) {
            values (" 2.0 ");
        }
    }
    timing() {
        related_pin    : "CLR" ;
        timing_type    : preset ;
        timing_sense   : negative_unate ;
        cell_rise(scalar) {
            values (" 1.0 ");
        }
    }
    timing() {
        related_pin    : "PRE" ;
        timing_type    : clear ;
        timing_sense   : positive_unate ;
        cell_fall(scalar) {
            values (" 1.0 ");
        }
    }
}
} /* end of cell DLT2
leakage_current Group

A leakage_current group is defined within a cell group or a model group to specify leakage current values that are dependent on the state of the cell.

Syntax

library (name) {

  cell(cell_name) {
    ...
    leakage_current() {
      when : boolean expression;
      pg_current(pg_pin_name) {
        value : float;
      }
    }
    ...
  }
}

Simple Attributes

when
value

Complex Attribute

mode

Group

pg_current

when Simple Attribute

This attribute specifies the state-dependent condition that determines whether the leakage current is accessed.

A leakage_current group without a when attribute is defined as a default state. The default state is associated with a leakage model that does not depend on the state condition. If all state conditions of a cell are specified, a default state is not required. If some state conditions of a cell are missing, the default state is assigned. If no default state is given, the leakage current defaults to 0.0.

Syntax

when : "Boolean expression" ;

Boolean expression

  Specifies the state-dependent condition.
value Simple Attribute

When a cell has a single power and ground pin, omit the \texttt{pg\_current} group and specify the leakage current value. Otherwise, specify the value in the \texttt{pg\_current} group. Current conservation is applied for each \texttt{leakage\_current} group. The \texttt{value} attribute specifies the absolute value of leakage current on a single power and ground pin.

Syntax

\texttt{value : valuefloat ;}

\texttt{value}

A floating-point number representing the leakage current.

mode Complex Attribute

The \texttt{mode} attribute specifies the current mode of operation of the cell. Use this attribute in the \texttt{leakage\_current} group to define the leakage current in the specified mode.

Syntax

\texttt{mode (mode\_name, mode\_value) ;}

Example

\texttt{mode (rw, read) ;}

pg\_current Group

Use this group to specify a power or ground pin where leakage current is to be measured.

Syntax

\texttt{cell(cell\_name) { \ ... \ leakage\_current() { \ when : boolean expression; \ pg\_current(pg\_pin\_name) { \ value : float; \ } \}}

\texttt{pg\_pin\_name}

Specifies the power or ground pin where the leakage current is to be measured.

Simple Attribute

\texttt{value}

Use this attribute in the \texttt{pg\_current} group to specify the leakage current value when a cell has multiple power and ground pins. The leakage current is measured toward a cell. For power pins, the current is positive if it is dragged into a cell. For ground pins, the current is
negative, indicating that current flows out of a cell. If all power and ground pins are specified
within a leakage_current group, the sum of the leakage currents should be zero.

**Syntax**

```
value : valuefloat ;
```

`value`

A floating-point number representing the leakage current.

---

**gate Leakage Group**

The gate Leakage group specifies the cell’s gate leakage current on input or inout pins
within the leakage_current group in a cell. The following applies to gate Leakage groups:

- Groups can be placed in any order if there is more than one gate Leakage group within
a leakage_current group.
- The leakage current of a cell is characterized with opened outputs, which means that
modeling cell outputs do not drive any other cells. Outputs are assumed to have zero
static current during the measurement.
- A missing gate Leakage group is allowed for certain pins.
- Current conservation is applicable if it can be applied to higher error tolerance.

**Syntax**

```
gate Leakage (input_pin_name)
```

**Example**

```
cell (my_cell) {
...
leakage_current {
...
}
... 
gate Leakage (A) {
  input_low_value : -0.5 ;
  input_high_value : 0.6 ;
}
```

**Simple Attributes**

```
input_low_value
input_high_value
```
input_low_value Simple Attribute

The input_low_value attribute specifies gate leakage current on an input or inout pin when the pin is in a low state condition.

The following applies to the input_low_value attribute:

• A negative floating-point number value is required.
• The gate leakage current flow is measured from the power pin of a cell to the ground pin of its driver cell.
• The input pin is pulled up to low.
• The input_low_value attribute is not required for a gate_leakage group.

Syntax
input_low_value : float ;

Example

gate_leakage (A) {
  input_low_value : -0.5 ;
  input_high_value : 0.6 ;
}

input_high_value Simple Attribute

The input_high_value attribute specifies gate leakage current on an input or inout pin when the pin is in a high state condition.

• The gate leakage current flow is measured from the power pin of its driver cell to the ground pin of the cell itself.
• A positive floating-point number value is required.
• The input pin is pulled up to high.
• The input_high_value attribute is not required for a gate_leakage group.

Syntax
input_high_value : float ;

Example

gate_leakage (A) {
  input_low_value : -0.5 ;
}
input_high_value : 0.6 ;
}

---

**leakage_power Group**

A leakage_power group is defined within a cell group or a model group to specify leakage power values that are dependent on the state of the cell.

**Note:**

Cells with state-dependent leakage power also need the cell_leakage_power simple attribute. See "cell_leakage_power Simple Attribute" on page 2-6.

**Syntax**

```plaintext
library (name) {
    cell (name) {
        leakage_power () {
            ...
        }
    }
}
```

**Simple Attributes**

- `power_level`
- `related_pg_pin`
- `when`
- `value`

**Complex Attribute**

- `mode`

**power_level Simple Attribute**

Use this attribute to specify the power consumed by the cell.

**Syntax**

```plaintext
power_level : "name" ;
```

**name**

Name of the power rail defined in the power supply group.

**Example**

```plaintext
power_level : "VDD1" ;
```
related\_pg\_pin Simple Attribute

Use this optional attribute to associate a power and ground pin with leakage power and internal power tables. The leakage power and internal energy tables can be omitted when the voltage of a primary\_power or backup\_ground pg\_pin is at reference voltage zero, since the value of the corresponding leakage power and internal energy tables are always 0.

In the absence of a related\_pg\_pin attribute, the internal\_power/leakage\_power specifications apply to the whole cell (cell-specific power specification).

Cell-specific and pg\_pin-specific power specifications cannot be mixed; that is, when one leakage\_power (internal\_power) group has the related\_pg\_pin attribute, all the leakage\_power (internal\_power) groups must have the related\_pg\_pin attribute.

Syntax

related\_pg\_pin : pg\_pinid;

pg\_pin

The related power and ground pin name.

Example

related\_pg\_pin : G2 ;

when Simple Attribute

This attribute specifies the state-dependent condition that determines whether the leakage power is accessed.

Syntax

when : "Boolean expression" ;

Boolean expression

Name of pin or pins in a cell for which leakage\_power is different.

Table 2-11 lists the Boolean operators valid in a when statement.

Table 2-11 Valid Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>invert previous expression</td>
</tr>
<tr>
<td>!</td>
<td>invert following expression</td>
</tr>
<tr>
<td>^</td>
<td>logical XOR</td>
</tr>
</tbody>
</table>
value Simple Attribute

Use this attribute to specify the leakage power for a given state of a cell.

**Syntax**

```
value : value_float ;
```

**value**

A floating-point number representing the leakage power value.

The following example defines the leakage_power group and the cell_leakage_power simple attribute in a cell:

**Example**

```
cell () {
  ...
  leakage_power () {
    when : "A" ;
    value : 2.0 ;
  }
  cell_leakage_power : 3.0 ;
}
```

**mode Complex Attribute**

The mode attribute specifies the current mode of operation of the cell. Use this attribute in the leakage_power group to define the leakage power in the specified mode.

---

### Table 2-11 Valid Boolean Operators (Continued)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>logical AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td>space</td>
<td>logical AND</td>
</tr>
<tr>
<td>+</td>
<td>logical OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td>0</td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>
Syntax
mode (mode_name, mode_value) ;

Example
mode (rw, read) ;

lut Group
A lut group defines a single variable that is then used to represent the lookup table value in the function attribute of a pin group. The lut group applies only to FPGA libraries.

Syntax
library (name_string) {
  cell (name_string) {
    lut (name) {
      ... ;
    }
  }
}

Example
cell () {
  ...
  lut(L) {
    input_pins : "A B C D" ;
  }
  pin (Z){
    ...
    function: "L" ;
  }
}

input_pins Simple Attribute
input_pins : "name1 [name2 name3 ...]" ;

mode_definition Group
A mode_definition group defines a set of modes of operation of a cell.

The power and timing requirements vary in different modes.

For more information about the mode_definition group, see the Liberty User Guide, Vol. 1.

Syntax
cell(cell_name) {
  mode_definition (mode_definition_name) {
mode_value (mode_name) {
    when : "Boolean expression" ;
    sdf_cond : "Boolean expression" ;
}

Example

cell(example_cell) {
...
    mode_definition(rw) {
        mode_value(read) {
            when : "R";
            sdf_cond : "R == 1";
        }
        mode_value(write) {
            when : "!R";
            sdf_cond : "R == 0";
        }
    }
}

Groups
mode_value

mode_value Group
The mode_value group defines a mode, and the condition for the mode to occur. When the condition evaluates to true, the cell operates in that mode.

Simple Attributes
when
sdf_cond

Complex Attributes
pg_setting

when Simple Attribute
The when attribute specifies the logic condition for a cell to operate in a particular mode.

Syntax
when : "Boolean expression" ;

Example
when: !R;
sdf_cond Simple Attribute

The sdf_cond attribute supports Standard Delay Format (SDF) file generation and condition matching during back-annotation.

Syntax
sdf_cond : "Boolean expression" ;

Example
sdf_cond: "R == 0" ;

pg_setting Complex Attribute

In PG pin power state models, the pg_setting complex attribute specifies the power state of a cell mode by referencing a power state (psv_name) defined in the pg_settings_definition group of the cell.

You can define multiple pg_setting attributes in a mode_value group provided each of the pg_setting attributes references a power state from a different pg_settings_definition group.

Syntax
pg_setting (psd_name, psv_name);

Example
pg_setting (psd1, psv1);

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

pg_settings_definition Group

The pg_settings_definition group defines the following:

- The system power states (pg_settings_value group).
- The legality of transitions (pg_settings_transition group) between the system power states.
- The default power state of the cell.
- The legality of undefined transitions.

Syntax
cell (cell_name)
  ...
  pg_settings_definition(psd_name) {
    ...
  }

Name of the \texttt{pg\_setting\_definition} group. The system power states must be mutually exclusive.

\textbf{Example}
\begin{verbatim}
pg\_setting\_definition(psd1) {
  ...
}
\end{verbatim}

\textbf{Simple Attributes}
- \texttt{default\_pg\_setting}
- \texttt{illegal\_transition\_if\_undefined}

\textbf{Groups}
- \texttt{pg\_setting\_value}
- \texttt{pg\_setting\_transition}
- \texttt{mode\_definition}

\textbf{default\_pg\_setting Simple Attribute}
The \texttt{default\_pg\_setting} attribute specifies a default power state to match when the explicitly defined power states do not completely cover the state space of the \texttt{pg\_setting\_definition} group. The default power state also acts as the initial power state of the \texttt{pg\_setting\_definition} group.

\textbf{Syntax}
\begin{verbatim}
default\_pg\_setting : psv\_name ;
\end{verbatim}

\texttt{psv\_name} is the name of a \texttt{pg\_setting\_value} group specified in the same \texttt{pg\_setting\_definition} group.

\textbf{Example}
\begin{verbatim}
default\_pg\_setting : "psv1" ;
\end{verbatim}

\textbf{illegal\_transition\_if\_undefined Simple Attribute}
The \texttt{illegal\_transition\_if\_undefined} attribute, if set to \texttt{true}, means that all the undefined transitions in the \texttt{pg\_setting\_definition} group are illegal or invalid. By default, all undefined transitions are valid.

\textbf{Syntax}
\begin{verbatim}
illegal\_transition\_if\_undefined : true | false ;
\end{verbatim}

\textbf{Example}
\begin{verbatim}
illegal\_transition\_if\_undefined : true ;
\end{verbatim}
**pg_setting_value Group**

The `pg_setting_value` group defines a system power state named `psv_name`. Specify this group in the `pg_setting_definition` group.

**Syntax**

```
pg_setting_value (psv_name) {
    ...
}
```

**Example**

```
pg_setting_definition(psd) {
    pg_setting_value (psv1) {
        ...
    }
}
```

**Simple Attributes**

- `pg_pin_condition`
- `pg_setting_condition`

**Complex Attributes**

- `pg_pin_active_state`
- `pg_setting_active_state`

**pg_pin_condition Simple Attribute**

The `pg_pin_condition` simple attribute specifies the logic condition when the system is in the power state named `psv_name`. For a single-state PG pin, this condition evaluates to true when the PG pin is in the on state. For a multi-state PG pin, this condition evaluates to true whenever the PG pin is in the active state specified by the `pg_pin_active_state` attribute.

**Syntax**

```
pg_pin_condition : Boolean expression of pg_pin and signal pin;
```

**Example**

```
pg_pin_condition : "VDD * VDDS * !VSS";
```

**pg_setting_condition Simple Attribute**

The `pg_setting_condition` complex attribute specifies the logic condition when the system is in the power state, `psv_name`. This condition evaluates to true whenever `psd_name2` is in the active state specified by the `pg_setting_active_state` attribute.

**Syntax**

```
pg_setting_condition : Boolean expression of psd_name2 & signal pin;
```
**Example**

```
pg_setting_condition : "P1 * P2" ;
```

**pg_pin_active_state** Complex Attribute

The `pg_pin_active_state` complex attribute defines the active power supply state, `pg_state`, for a specified PG pin. The attribute only applies to a multi-state power supply. For a PG pin with a single state, the pin is active when it is on.

**Syntax**

```
pg_pin_active_state (pg_pin, pg_state) ;
```

**Example**

```
pg_pin_active_state (VDD, HV) ;
```

**pg_setting_active_state** Complex Attribute

The `pg_setting_active_state` complex attribute specifies the active system power state, `psv_name2`, of another `pg_setting_definition` group (named `psd_name2`) in the system power state, `psv_name`.

**Syntax**

```
pg_setting_active_state (psd_name2, psv_name2);
```

**Example**

```
pg_setting_active_state (psd2, psv1);
```

**pg_setting_transition** Group

The `pg_setting_transition` group specifies the legal and illegal transitions between the PG modes defined in the `pg_setting_definition` group.

**Syntax**

```
pg_setting_transition (pst_name) {
    ...
}
```

**pst_name**

Name you assign to the transition.

**Example**

```
pg_setting_transition (sleep) {
    ...
}
```
Simple Attributes

**is_illegal**

**start_setting**

**end_setting**

**is_illegal Simple Attribute**

The **is_illegal** simple attribute specifies if the transition, *pst_name*, is illegal. The default is *false*.

**Syntax**

```
is_illegal : [ true | false ];
```

**Example**

```
is_illegal : true ;
```

**start_setting Simple Attribute**

The **start_setting** simple attribute specifies the power state from where the transition, *pst_name*, starts.

**Syntax**

```
start_setting : psv_name1;
```

**Example**

```
start_setting : on;
```

**end_setting Simple Attribute**

The **end_setting** attribute specifies the power state where the transition, *pst_name*, ends.

**Syntax**

```
end_setting : psv_name2;
```

**Example**

```
end_setting : sleep;
```

---

**pg_pin Group**

Use the **pg_pin** group to specify power and ground pins. The library cells can have multiple **pg_pin** groups. A **pg_pin** group is mandatory for each cell. A cell must have at least one **primary_power** pin specified in the **pg_type** attribute and at least one **primary_ground** pin specified in the **pg_type** attribute.
Syntax

cell (name_string){
    pg_pin (pg_pin_name_string){
        voltage_name : value_id ;
        pg_type : value_enum ;
    } /* end pg_pin */
    ... 
    } /* end cell */

Simple Attributes

voltage_name
pg_type
user_pg_type
physical_connection
related_bias_pin

voltage_name Simple Attribute

Use the voltage_name attribute to specify an associated voltage. This attribute is optional in the pg_pin group of a level-shifter cell not powered by the switching power domains, where the pg_pin group has the std_cell_main_rail attribute.

Syntax

voltage_name : value_id ;

value

A voltage defined in a library-level voltage_map attribute.

Example

voltage_name : VDD1 ;

pg_type Simple Attribute

Use the optional pg_type attribute to specify the type of power and ground pin. The pg_type attribute also supports back-bias modeling. The pg_type attribute can have the following values: primary_power, primary_ground, backup_power, backup_ground, internal_power, internal_ground, pwell, nwell, deepnwell, and deppwell. The pwell and nwell values specify regular wells, and the deppwell and deepnwell values specify isolation wells.

Syntax

pg_type : value_enum ;
value

The valid values are primary_power, primary_ground, backup_power, backup_ground, internal_power, internal_ground, pwell, nwell, deepnwell, and deeppwell.

Example

pg_type : primary_power ;

Example of a 2-input NAND Cell With Virtual Bias Pins

The following example shows a 2-input NAND cell with virtual bias pins to support back-bias modeling.

library (sample_standard_cell_with_bias_pin) {
  cell ( nand2 ) {
    pg_pin ( vdd ) {
      pg_type : primary_power ;
      ...
    }
    pg_pin ( vss ) {
      pg_type : primary_ground ;
      ...
    }
    pg_pin ( vpw ) {
      pg_type : pwell ;
      ...
    }
    pg_pin ( vnw ) {
      pg_type : nwell ;
      ...
    }
    pin ( A ) {
      direction : input;
      related_power_pin : "vdd" ;
      related_ground_pin : "vss" ;
      related_bias_pin : "vpw vnw" ;
      ...
    }
    pin ( B ) {
      direction : input;
      related_power_pin : "vdd" ;
      related_ground_pin : "vss" ;
      related_bias_pin : "vpw vnw" ;
      ...
    }
    pin ( Z ) {
      direction : output;
      function : " !(A * B) " ;
      related_power_pin : "vdd" ;
      related_ground_pin : "vss" ;
Example of a Level-Shifter Cell With Virtual Bias Pins

The following example shows a level-shifter cell with virtual bias pins and two nwell regular wells for back-bias modeling.

```vhdl
library (sample_multi_rail_with_bias_pins) {
  cell (level_shifter) {
    pg_pin (vdd1) {
      pg_type : primary_power;
    }
    pg_pin (vdd2) {
      pg_type : primary_power;
    }
    pg_pin (vss) {
      pg_type : primary_ground;
    }
    pg_pin (vpw) {
      pg_type : pwell;
    }
    pg_pin (vnw1) {
      pg_type : nwell;
    }
    pg_pin (vnw2) {
      pg_type : nwell;
    }
    pin (I) {
      direction : input;
      related_power_pin : vdd1
      related_ground_pin : vss
      related_bias_pin : "vnw1 vpw"
    }
    pin (Z) {
      direction : output;
      function : "I";
      related_power_pin : "vdd2";
      related_ground_pin : "vss";
      related_bias_pin : "vnw2 vpw";
      power_down_function : "!vdd1 + !vdd2 + vss + !vnw1 + !vnw2 + vpw";
    }
  }
}
```
user_pg_type Simple Attribute

The `user_pg_type` optional attribute allows you to customize the type of power and ground pin that is used in a library. It accepts any string value, as shown:

```plaintext
pg_pin (pg_pin_name) {
    voltage_name : voltage_name;
    pg_type : primary_power | primary_ground |
               backup_power | backup_ground |
               internal_power | internal_ground;
    user_pg_type : user_pg_type_name;
}
```

Example

The following example shows a `pg_pin` library with the `user_pg_type` attribute specified.

```plaintext
pg_pin (A) {
    voltage_name : VDD1;
    pg_type : primary_power
    user_pg_type : my_pg_type;
}
```

physical_connection Simple Attribute

The `physical_connection` attribute provides two possible values: `device_layer` and `routing_pin`. The `device_layer` value specifies that the bias connection is physically external to the cell. In this case, the library provides biasing tap cells that connect through the device layers. The `routing_pin` value specifies that the bias connection is inside a cell and is exported as a physical geometry and a routing pin. Macros with pin access generally use the `routing_pin` value if the cell has bias pins with geometry that is visible in the physical view.

Example

The following example shows virtual routing pin modeling, where the bias connection is physically external to the cell.

```plaintext
pg_pin(VDDS){
    voltage_name : VDDS;
    direction : input;
    pg_type : pwell | nwell | deepnwell | deapppwell;
    physical_connection : device_layer;
}
```
related_bias_pin

The `related_bias_pin` attribute defines all bias pins associated with a power or ground pin within a cell. The `related_bias_pin` attribute is required only when the attribute is declared in a pin group but it does not specify a complete relationship between the bias pin and power and ground pin for a library cell.

The `related_bias_pin` attribute also defines all bias pins associated with a signal pin. To associate back-bias pins to signal pins, use the `related_bias_pin` attribute to specify one of the following `pg_type` values: `pwell`, `nwell`, `deeppwell`, `deepnwell`.

**Example with a Power and Ground Pin**

The following example shows the association of a back-bias pin to a power and ground pin.

```plaintext
pg_pin(signal_pin){
    related_power_pin : pg_pin_name;
    related_ground_pin : pg_pin_name;
    related_bias_pin : "bias_pin_name bias_pin_name ...";
}
```

**Example with a Signal Pin**

The following example shows the association of a back-bias pin to a signal pin.

```plaintext
pg_pin(pg_pin_name){
    related_bias_pin : "bias_pin_name bias_pin_name ...";
}
```

**is_insulated Simple Attribute**

The `is_insulated` attribute specifies that a substrate-bias PG pin has an insulated well. The default is `false` meaning the bias PG pin substrate is not an insulated well. It is defined in a `pg_pin` group with the `pg_type` attribute set to `pwell`, `nwell`, `deeppwell`, or `deepnwell`.

**Syntax**

```plaintext
is_insulated : Boolean expression;
```

**Boolean expression**

Valid values are true and false.

**Example**

```plaintext
is_insulated : true;
```
**tied_to Simple Attribute**

The optional `tied_to` attribute specifies the PG pin connected or tied to the substrate-bias PG pin.

**Syntax**

`tied_to : pgpin_name ;`

`pgpin_name`
The PG pin connected or tied to the substrate-bias PG pin.

**Example**

`tied_to : VDD1 ;`

---

**preset_condition Group**

The `preset_condition` group is a group of attributes for a condition check on the normal mode preset expression.

If preset is asserted during the restore operation, it needs to extend beyond the restore operation time period so that the flip-flop content can be successfully overwritten. Therefore, trailing-edge condition checks on preset pins might be needed.

`preset_condition() {
    input : "Boolean_expression" ;
    required_condition : "Boolean_expression" ;
}

**Simple Attributes**

`input`
`required_condition`

**input Simple Attribute**

The `input` attribute should be identical to the `preset` attribute in the `ff` group and defines how the asynchronous preset control is asserted.

**Syntax**

`input : "Boolean_expression" ;`

**required_condition Simple Attribute**

The `required_condition` attribute specifies the condition that the `input` attribute is required to be and is evaluated at the positive edge of the `clocked_on` attribute in the `clock_condition` group. If the expression evaluates to false, the cell is in an illegal state.
retention_condition Group

The retention_condition group includes attributes that specify the conditions for the retention cell to hold its state during the retention mode.

```plaintext
class retention_condition {
    power_down_function : "Boolean_expression" ;
    required_condition : "Boolean_expression" ;
}
```

**Simple Attributes**

- `power_down_function`
- `required_condition`

**power_down_function Simple Attribute**

The power_down_function attribute specifies the Boolean condition for the retention cell to be powered down, that is, the primary power to the cell is shut down. When this Boolean condition evaluates to true, it triggers the evaluation of the control input conditions specified by the `required_condition` attribute.

**Syntax**

```plaintext
power_down_function : "Boolean_expression" ;
```

**Example**

```plaintext
power_down_function : "!VDD + VSS" ;
```

**required_condition Simple Attribute**

The required_condition attribute specifies the control input conditions during the retention mode. These conditions are checked when the primary power to the retention cell is shut down. If these conditions are not met, the cell is considered to be in an illegal state.

**Note:**

Within the retention_condition group, the power_down_function attribute by itself does not specify the retention mode of the cell. The conditions specified by the required_condition attribute ensure that the retention control pin is in the correct state when the primary power to the cell is shut down.

**Syntax**

```plaintext
required_condition : "Boolean_expression" ;
```

**Example**

```plaintext
required_condition : "!CK * !RET" ;
```
statetable Group

The statetable group captures the function of more-complex sequential cells. It is defined in a cell group, model group, or test_cell group.

The purpose of this group is to define a state table. A state table is a sequential lookup table. It takes an arbitrary number of inputs and their delayed values and an arbitrary number of internal nodes and their delayed values to form an index to new internal node values.

Note:

In the statetable group, table is a keyword.

Syntax

```plaintext
statetable("input node names", "internal node names") { 
    table: "input node values : current internal values : \n    next internal values, \n    input node values : current internal values : \n    next internal values";
}
```

The following example shows a state table for a JK flip-flop with an active-low, direct-clear, and negative-edge clock:

Example

```plaintext
statetable ("J  K  CN  CD",  "IQ") { 
    table: 
}
```

test_cell Group

The test_cell group is in a cell group or model group. It models only the non-test behavior of a scan cell, which is described by an ff, ff_bank, latch, latch_bank or statetable statement and pin function attributes.

Syntax

```plaintext
library (name_string) { 
    cell (name_string) { 
        test_cell () { 
            ... test cell description ... 
        }
    }
}
```
You do not need to give the test cell a name, because the test cell takes the name of the cell being defined.

**Groups**

```
ff (variable1id, variable2id) {}  
ff_bank (variable1id, variable2string, bitsint) {}  
latch (variable1id, variable2id) {}  
latch_bank (variable1id, variable2id, bitsint) {}  
pin (nameid) {}  
statetable ("input node names", "internal node names") {}  
```

**ff Group**

For a discussion of the `ff` group syntax, see “ff Group” on page 2-60.

**ff_bank Group**

For a discussion of the `ff_bank` group syntax, see “ff_bank Group” on page 2-66.

**latch Group**

For a discussion of the `latch` group syntax, see “latch Group” on page 2-85.

**latch_bank Group**

For a discussion of the `latch_bank` group syntax, see “latch_bank Group” on page 2-89.

**pin Group in a test_cell Group**

Both test `pin` and non-test `pin` groups appear in `pin` groups within a `test_cell` group, as shown:

```
library (namestring) {  
cell (namestring) {  
test_cell (namestring) {  
pin (namestring | name_liststring) {  
... pin description ...  
}  
}  
}  
}  
```

These groups are similar to `pin` groups in a `cell` group or `model` group but can contain only direction, function, signal_type, and test_output_only attributes. They cannot contain timing, capacitance, fanout, or load information.

**Simple Attributes**

```
direction : input | output | inout ;
```
function : Boolean expression ;
signal_type: test_scan_in | test_scan_inverted | 
test_scan_out | test_scan_out_inverted | 
test_scan_enable | test_scan_enable_inverted | 
test_scan_clock | test_scan_clock_a | 
test_scan_clock_b | test_clock ;
test_output_only : true | false ;

Group
estatetable() { }

direction Attribute
The direction attribute states whether the pin being described is an input, output, or inout (bidirectional) pin. The default is input.

Syntax
direction : input | output | inout ;

Example
direction : input ;

function Attribute
The function attribute reflects only the nontest behavior of a cell.
An output pin must have either a function attribute or a signal_type attribute.

The function attribute in a pin group defines the value of an output pin or inout pin in terms of the input pins or inout pins in the cell group or model group. For more details about function, see the “function Simple Attribute” on page 2-35.

Syntax
function : "Boolean expression" ;

Boolean expression
Identifies the replaced cell.

Example
function : "IQ" ;

signal_type Attribute
In a test_cell group, signal_type identifies the type of test pin.

Syntax
signal_type : "value" ;
Descriptions of the possible values for the `signal_type` attribute follow:

`test_scan_in`
Identifies the scan-in pin of a scan cell. The scanned value is the same as the value present on the scan-in pin. All scan cells must have a pin with either the `test_scan_in` or the `test_scan_in_inverted` attribute.

`test_scan_in_inverted`
Identifies the scan-in pin of a scan cell as being of inverted polarity. The scanned value is the inverse of the value present on the scan-in pin.

For multiplexed flip-flop scan cells, the polarity of the scan-in pin is inferred from the latch or ff declaration of the cell itself. For other types of scan cells, clocked-scan, level-sensitive scan design (LSSD), and multiplexed flip-flop latches, it is not possible to give the ff or latch declaration of the entire scan cell. For these cases, you can use the `test_scan_in_inverted` attribute in the cell where the scan-in pin appears in the latch or ff declarations for the entire cell.

`test_scan_out`
Identifies the scan-out pin of a scan cell. The value present on the scan-out pin is the same as the scanned value. All scan cells must have a pin with either a `test_scan_out` or a `test_scan_out_inverted` attribute.

The scan-out pin corresponds to the output of the slave latch in the LSSD methodologies.

`test_scan_out_inverted`
Identifies the scan-out of a test cell as having inverted polarity. The value on this pin is the inverse of the scanned value.

`test_scan_enable`
Identifies the pin of a scan cell that, when high, indicates that the cell is configured in scan-shift mode. In this mode, the clock transfers data from the scan-in input to the scan-out input.

`test_scan_enable_inverted`
Identifies the pin of a scan cell that, when low, indicates that the cell is configured in scan-shift mode. In this mode, the clock transfers data from the scan-in input to the scan-out input.

`test_scan_clock`
Identifies the test scan clock for the clocked-scan methodology. The signal is assumed to be edge-sensitive. The active edge transfers data from the scan-in pin to the scan-out
pin of a cell. The sense of this clock is determined by the sense of the associated timing arcs.

test_scan_clock_a

Identifies the a clock pin in a cell that supports the single-latch LSSD, double-latch LSSD, clocked LSSD, or auxiliary clock LSSD methodologies. When the a clock is at the active level, the master latch of the scan cell can accept scan-in data. The sense of this clock is determined by the sense of the associated timing arcs.

test_scan_clock_b

Identifies the b clock pin in a cell that supports the single-latch LSSD, clocked LSSD, or auxiliary clock LSSD methodologies. When the b clock is at the active level, the slave latch of the scan-cell can accept the value of the master latch. The sense of this clock is determined by the sense of the associated timing arcs.

test_clock

Identifies an edge-sensitive clock pin that controls the capturing of data to fill scan-in test mode in the auxiliary clock LSSD methodology.

If an input pin is used in both test and nontest modes (such as the clock input in the multiplexed flip-flop methodology), do not include a signal_type statement for that pin in the test_cell pin definition.

If an input pin is used only in nontest mode and does not exist on the cell that it scans and replace, you must include a signal_type statement for that pin in the test_cell pin definition.

If an output pin is used in nontest mode, it needs a function statement. The signal_type statement is used to identify an output pin as a scan-out pin. In a test_cell group, the pin group for an output pin can contain a function statement, a signal_type attribute, or both.

You do not have to define a function or signal_type attribute in the pin group if the pin is defined in a previous test_cell group for the same cell.

Example

signal_type : "test_scan_in" ;

test_output_only Attribute

This attribute is an optional Boolean attribute that you can set for any output port described in statetable format.

For a flip-flop or latch, if a port is used for both function and test, you provide the functional description using the function attribute. If a port is used for test only, omit the function attribute.
For a state table, a port always has a functional description. Therefore, to specify that a port is for test only, set the `test_output_only` attribute to `true`.

**Syntax**

```plaintext
test_output_only : true | false ;
```

**Example**

```plaintext
test_output_only : true ;
```

**statetable Group**

For a discussion of the `statetable` group syntax, see “`statetable Group`” on page 2-118.

---

**type Group**

The `type` group, when defined within a cell, is a type definition local to the cell. It cannot be used outside of the cell.

**Syntax**

```plaintext
cell (namestring) {
    type (namestring) {
        ... type description ...
    }
}
```

**Simple Attributes**

- `base_type` : `array`
- `bit_from` : `integer`
- `bit_to` : `integer`
- `bit_width` : `integer`
- `data_type` : `bit`
- `downto` : `Boolean`

**base_type Simple Attribute**

The only valid base type value is `array`.

**Example**

```plaintext
base_type : array ;
```

**bit_from Simple Attribute**

The `bit_from` attribute specifies the member number assigned to the most significant bit (MSB) of successive array members.
Syntax

\[
\text{bit\_from} : \text{value}_{\text{int}} ;
\]

\textit{value}

Indicates the member number assigned to the MSB of successive array members. The default is 0.

**Example**

\[
\text{bit\_from} : 0 ;
\]

**bit\_to Simple Attribute**

The \textit{bit\_to} attribute specifies the member number assigned to the least significant bit (LSB) of successive array members.

Syntax

\[
\text{bit\_to} : \text{value}_{\text{int}} ;
\]

\textit{value}

Indicates the member number assigned to the LSB of successive array members. The default is 0.

**Example**

\[
\text{bit\_to} : 3 ;
\]

**bit\_width Simple Attribute**

The \textit{bit\_width} attribute specifies the integer that designates the number of bus members.

Syntax

\[
\text{bit\_width} : \text{value}_{\text{int}} ;
\]

\textit{value}

Designates the number of bus members. The default is 1.

**Example**

\[
\text{bit\_width} : 4 ;
\]

**data\_type Simple Attribute**

Only the \textit{bit} data type is supported.

**Example**

\[
\text{data\_type} : \text{bit} ;
\]
**downto Simple Attribute**

The `downto` attribute specifies a Boolean expression that indicates whether the MSB is high or low.

**Syntax**

```
downto : true | false ;
```

`true`

Indicates that member number assignment is from high to low. The default is `false` (low to high).

**Example 2-17** illustrates a type group statement in a cell.

**Example 2-17  type Group Within a Cell**

```plaintext
cell (buscell4) {  
type (BUS4) {  
    base_type : array ;  
    data_type : bit ;  
    bit_width : 4 ;  
    bit_from : 0 ;  
    bit_to : 3 ;  
    downto : true ;
  }
}
```

---

**model Group**

A model group is defined within a library group, as shown here:

**Syntax**

```
library (namestring) {  
    model (namestring) {  
      ... model description ...
    }
}
```

---

**Attributes and Values**

A model group can include all the attributes that are valid in a cell group, as well as the two additional attributes described in this section. For information about the cell group attributes, see “Attributes and Values” on page 2-2.
Simple Attribute
cell_name

Complex Attribute
short

cell_name Simple Attribute
The cell_name attribute specifies the name of the cell within a model group.

Syntax
cell_name : "name_string" ;

Example
model(modelA) {
  cell_name : "cellA";
  ...
}

short Complex Attribute
The short attribute lists the shorted ports that are connected together by a metal or poly trace. These ports are modeled within a model group.

The most common example of a shorted port is a feedthrough, where an input port is directly connected to an output port. Another example is two output ports that fan out from the same gate.

Syntax
short ("name_list_string") ;

Example
short(b, y);

Example 2-18 shows how to use a short attribute in a model group.

Example 2-18 Using the short Attribute in a model Group
model(cellA) {
  area : 0.4;
  ...
  short(b, y);
  short(c, y);
  short(b, c);
  ...
  pin(y) {
    direction : output;
    timing() {

related_pin : a;
...
}
}
}
}
pin(a) {
  direction : input;
  capacitance : 0.1;
}
pin(b) {
  direction : input;
  capacitance : 0.1;
}
pin(c) {
  direction : input;
  capacitance : 0.1;
  clock : true;
}
3

pin Group Description and Syntax

You can define a pin group within a cell, test_cell, model, or bus group.

This chapter contains

• An example of the pin group syntax showing the attribute and group statements that you can use within the pin group

• Descriptions of the attributes and groups you can use in a pin group
A pin group can include simple and complex attributes and group statements. In a cell or bus group, the syntax of a pin group is as follows:

```
library (name) {
  cell (name) {
    pin (name | name_list) {
      ... pin description ...
    }
  }
  cell (name) {
    bus (name) {
      pin (name | name_list) {
        ... pin description ...
      }
    }
  }
}
```

### Simple Attributes

Example 3-1 lists alphabetically a sampling of the attributes and groups that you can define within a pin group.

**Example 3-1  Attributes and Values in a pin Group**

```
/* Simple Attributes in a pin Group */
alive_during_power_up : true | false;
always_on : true | false;
antenna_diode_type : power | ground | power_and_ground;
antenna_diode_related_ground_pins : "ground_pin1 ground_pin2";
antenna_diode_related_power_pins : "power_pin1 power_pin2";
bit_width : integer; /* bus cells */
capacitance : float;
clock : true | false;
clock_gate_clock_pin : true | false;
clock_gate_enable_pin : true | false;
clock_gate_test_pin : true | false;
clock_gate_obs_pin : true | false;
clock_gate_out_pin : true | false;
clock_isolation_cell_clock_pin : true | false;
complementary_pin : "string";
connection_class : "name1 [name2 name3 ... ]";
direction : input | output | inout | internal;
dont_fault : sa0 | sa1 | sa01;
drive_current : float;
```
driver_type : pull_up | pull_down | open_drain | open_source | bus_hold | resistive | resistive_0 | resistive_1;
fall_capacitance : float;
fall_current_slope_after_threshold : float;
fall_current_slope_before_threshold : float;
fall_time_after_threshold : float;
fall_time_before_threshold : float;
fanout_load : float;
fault_model : "two-value string";
function : "Boolean expression";
has_builtin_pad : Boolean expression;
hysteresis : true | false;
illegal_clamp_condition : "Boolean expression";
input_map : "namestring | name_list";
input_signal_level : string;
inverted_output : true | false; /* Required in statetable cells */
is_pad : true | false;
max_capacitance : float;
max_fanout : float;
max_transition : float;
min_capacitance : float;
min_fanout : float;
min_period : float;
min_pulse_width_high : float;
min_pulse_width_low : float;
min_transition : float;
multicell_pad_pin : true | false;
nextstate_type : data | preset | clear | load | scan_in | scan_enable;
output_signal_level : string;
output_signal_level_high : float;
output_signal_level_low : float;
output_voltage : string;
pin_func_type : clock_enable | active_high | active_low | active_rising | active_falling;
prefer_tied : "0" | "1";
primary_output : true | false;
pulling_current : current value;
pulling_resistance : resistance value;
restore_action : L | H | R | F;
restore_edge_type : edge_trigger | leading | trailing;
rise_capacitance : float;
rise_current_slope_after_threshold : float;
rise_current_slope_before_threshold : float;
rise_time_after_threshold : float;
rise_time_before_threshold : float;
save_action : L | H | R | F;
signal_type : test_scan_in | test_scan_in_inverted | test_scan_out |
             test_scan_out_inverted | test_scan_enable |
             test_scan_enable_inverted | test_scan_clock |
             test_scan_clock_a | test_scan_clock_b | test_clock;
slew_control : low | medium | high | none;
state_function : "Boolean expression";
test_output_only : true | false;
three_state : "Boolean expression";
x_function : "Boolean expression";
alive_during_power_up Simple Attribute

The optional alive_during_power_up attribute specifies an active data pin that is powered by a more always-on supply. The default is false. If set to true, it indicates that the data pin is active while its related power pin is active.

You can specify this attribute only in a pin group where the isolation_cell_data_pin or the level_shifter_data_pin attribute is set to true.

Syntax
alive_during_power_up : Boolean expression ;

Boolean expression
Valid values are true and false.

Example
alive_during_power_up : true ;

always_on Simple Attribute

The always_on simple attribute models always-on cells or signal pins. Specify the attribute at the cell level to determine whether a cell is an always-on cell. Specify the attribute at the pin level to determine whether a pin is an always-on signal pin.

Syntax
always_on : Boolean expression ;

Boolean expression
Valid values are true and false.
Example
always_on : true;

antenna_diode_related_ground_pins Simple Attribute
For an antenna-diode cell, the \texttt{antenna\_diode\_related\_ground\_pins} attribute specifies the related ground pin of the cell. Apply the \texttt{antenna\_diode\_related\_ground\_pins} attribute to the input pin of the cell.

For a cell with a built-in antenna-diode pin or port, the \texttt{antenna\_diode\_related\_ground\_pins} attribute specifies the related ground pins for the antenna-diode pin. Apply the \texttt{antenna\_diode\_related\_ground\_pins} attribute to the antenna-diode pin.

Syntax
\texttt{antenna\_diode\_related\_ground\_pins} : "ground\_pin1 ground\_pin2";

Example
antenna_diode_related_ground_pins : "VSS1 VSS2";

antenna_diode_related_power_pins Simple Attribute
For an antenna-diode cell, the \texttt{antenna\_diode\_related\_power\_pins} attribute specifies the related power pin of the antenna-diode cell. Apply the \texttt{antenna\_diode\_related\_power\_pins} attribute to the input pin of the cell.

For a cell with a built-in antenna-diode pin or port, the \texttt{antenna\_diode\_related\_power\_pins} attribute specifies the related power pins for the antenna-diode pin. Apply the \texttt{antenna\_diode\_related\_power\_pins} attribute to the antenna-diode pin.

Syntax
\texttt{antenna\_diode\_related\_power\_pins} : "power\_pin1 power\_pin2";

Example
antenna_diode_related_power_pins : "VDD1 VDD2";

antenna_diode_type Simple Attribute
The \texttt{antenna\_diode\_type} attribute specifies the type of pin in a macro cell. Valid values are \texttt{power}, \texttt{ground}, and \texttt{power\_and\_ground}.

Note:
You can specify the pin-level \texttt{antenna\_diode\_type} attribute only for a macro cell.
Syntax
antenna_diode_type : power | ground | power_and_ground ;

Example
antenna_diode_type : power ;

**bit_width Simple Attribute**
The **bit_width** attribute designates the number of bus members. The default is 1.

Syntax
bit_width : integer ;

Example
bit_width : 4 ;

**capacitance Simple Attribute**
The **capacitance** attribute defines the load of an input, output, inout, or internal pin.

Syntax
capacitance : valuefloat ;

*value*

A floating-point number in units consistent with other capacitance specifications throughout the library. Typical units of measure for capacitance include picofarads and standardized loads.

Example
The following example defines the A and B pins in an AND cell, each with a capacitance of one unit.

cell (AND) {
  area : 3 ;
  pin (A,B) {
    direction : input ;
    capacitance : 1 ;
  }
}

**clamp_0_function Simple Attribute**
The **clamp_0_function** attribute specifies the input condition for the enable pins of an enable level-shifter or isolation cell when the output clamps to 0.
Syntax
clamp_0_function : "Boolean expression" ;

Example
clamp_0_function : "!EN1 * EN2" ;

clamp_1_function Simple Attribute
The clamp_1_function attribute specifies the input condition for the enable pins of an enable level-shifter or isolation cell when the output clamps to 1.

Syntax
clamp_1_function : "Boolean expression" ;

Example
clamp_1_function : "EN1 * !EN2" ;

clamp_z_function Simple Attribute
The clamp_z_function attribute specifies the input condition for the enable pins of an enable level-shifter or isolation cell when the output clamps to z.

Syntax
clamp_z_function : "Boolean expression" ;

Example
clamp_z_function : "EN1 * EN2" ;

clamp_latch_function Simple Attribute
The clamp_latch_function attribute specifies the input condition for the enable pins of an enable level-shifter or isolation cell when the output clamps to the previously latched value.

Syntax
clamp_latch_function : "Boolean expression" ;

Example
clamp_latch_function : "!EN1 * !EN2" ;

Note:
The Boolean expressions of the clamp_0_function, clamp_1_function, clamp_z_function, clamp_latch_function, and illegal Clamp Condition attributes must be mutually exclusive.
clock Simple Attribute
The clock attribute indicates whether an input pin is a clock pin.

Syntax
clock : true | false ;

The true value specifies the pin as a clock pin. The false value specifies the pin as not a clock pin, even though it might have the clock characteristics.

Example
The following example defines pin CLK2 as a clock pin.

pin(CLK2) {
  direction : input ;
  capacitance : 1.0 ;
  clock : true ;
}

clock_gate_clock_pin Simple Attribute
The clock_gate_clock_pin attribute identifies an input pin connected to a clock signal.

Syntax
clock_gate_clock_pin : true | false ;

A true value labels the pin as a clock pin. A false value labels the pin as not a clock pin.

Example
clock_gate_clock_pin : true ;

clock_gate_enable_pin Simple Attribute
The clock_gate_enable_pin attribute identifies an input port connected to an enable signal for nonintegrated clock-gating cells and integrated clock-gating cells.

Syntax
clock_gate_enable_pin : true | false ;

A true value labels the input port pin connected to an enable signal for nonintegrated and integrated clock-gating cells. A false value labels the input port pin connected to an enable signal as not for nonintegrated and integrated clock-gating cells.

Example
clock_gate_enable_pin : true ;
For nonintegrated clock-gating cells, you can set the `clock_gate_enable_pin` attribute to true on only one input port of a 2-input AND, NAND, OR, or NOR gate. If you do so, the other input port is the clock.

**clock_gate_test_pin Simple Attribute**

The `clock_gate_test_pin` attribute identifies an input pin connected to a `test_mode` or `scan_enable` signal.

**Syntax**

```plaintext
clock_gate_test_pin : true | false ;
```

A true value labels the pin as a test (`test_mode` or `scan_enable`) pin. A false value labels the pin as not a test pin.

**Example**

```plaintext
clock_gate_test_pin : true ;
```

**clock_gate_obs_pin Simple Attribute**

The `clock_gate_obs_pin` attribute identifies an output pin connected to an observability signal.

**Syntax**

```plaintext
clock_gate_obs_pin : true | false ;
```

A true value labels the pin as an observability pin. A false value labels the pin as not an observability pin.

**Example**

```plaintext
clock_gate_obs_pin : true ;
```

**clock_gate_out_pin Simple Attribute**

The `clock_gate_out_pin` attribute identifies an output port connected to an `enable_clock` signal.

**Syntax**

```plaintext
clock_gate_out_pin : true | false ;
```

A true value labels the pin as an out (`enable_clock`) pin. A false value labels the pin as not an out pin.

**Example**

```plaintext
clock_gate_out_pin : true ;
```
clock_isolation_cell_clock_pin Simple Attribute

The `clock_isolation_cell_clock_pin` attribute identifies an input clock pin of a clock-isolation cell. The default is `false`.

Syntax

clock_isolation_cell_clock_pin : true | false ;

Example

clock_isolation_cell_clock_pin : true ;

complementary_pin Simple Attribute

The `complementary_pin` attribute supports differential I/O. Differential I/O assumes the following:

- When the noninverting pin equals 1 and the inverting pin equals 0, the signal gets logic 1.
- When the noninverting pin equals 0 and the inverting pin equals 1, the signal gets logic 0.

Use the `complementary_pin` attribute to identify the differential input inverting pin with which the noninverting pin is associated and from which it inherits timing information and associated attributes.

For information on the `connection_class` attribute, see “connection_class Simple Attribute” on page 3-11.

Syntax

complementary_pin : "value_string" ;

value

Identifies the differential input data inverting pin whose timing information and associated attributes the noninverting pin inherits. Only one input pin is modeled at the cell level. The associated differential inverting pin is defined in the same pin group as the noninverting pin.

For details on the `fault_model` attribute that you use to define the value when both the complementary pins are driven to the same value, see “fault_model Simple Attribute” on page 3-22.

Example

cell (diff_buffer) {
  ...
  pin (A) { /* noninverting pin */
    direction : input ;
  }
}
complementary_pin : ("DiffA") /* inverting pin */
}
}

**connection_class Simple Attribute**
The `connection_class` attribute defines design rules for connections between cells. Only pins with the same connection class can be legally connected.

**Syntax**
connection_class : "name1 [name2 name3 ... ]" ;

**name**
A name or names of your choice for the connection class. You can assign multiple connection classes to a pin by separating the connection class names with spaces.

**Example**
connection_class : "internal" ;

**data_in_type Simple Attribute**
In a pin group, the `data_in_type` attribute specifies the type of input data defined by the `data_in` attribute in a latch or latch_bank group.

**Note:**
The Boolean expression of the `data_in` attribute must include the pin with the `data_in_type` attribute.

**Syntax**
data_in_type : data | preset | clear | load ;

data
  Identifies the pin as a synchronous data pin. This is the default value.

preset
  Identifies the pin as a synchronous preset pin.

clear
  Identifies the pin as a synchronous clear pin.

load
  Identifies the pin as a synchronous load pin.
Example

cell(new_cell) {
  latch (IQ, IQN) {
    enable : "(!ENN)"
    data_in : "D"
    clear : "(!RN)"
  }
  pin(D) {
    direction : input,
    data_in_type : preset,
    ...
  }
  ...
}

direction Simple Attribute

The direction attribute declares a pin as being an input, output, inout (bidirectional), or internal pin. The default is input.

Syntax

direction : input | output | inout | internal

Example

In the following example, both A and B in the AND cell are input pins; Y is an output pin.

cell (AND) {
  area : 3
  pin (A,B) {
    direction : input;
  }
  pin (Y) {
    direction : output;
  }
}

dont_fault Simple Attribute

The dont_fault attribute is a string ("stuck at") that you can set on a library cell or pin.

Syntax

dont_fault : sa0 | sa1 | sa01

Example

dont_fault : sa0;

The dont_fault attribute can also be defined in the cell group.
**drive_current Simple Attribute**

The `drive_current` attribute defines the drive current strength for the pad pin.

**Syntax**

```plaintext
drive_current : value_float ;
```

**value**

A floating-point number that represents the drive current the pad supplies in the units defined with the `current_unit` library-level attribute.

**Example**

```plaintext
drive_current : 5.0 ;
```

**driver_type Simple Attribute**

The `driver_type` attribute tells the VHDL library generator to use a special pin-driving configuration for the pin during simulation.

To support pull-up and pull-down circuit structures, the Liberty models for I/O pad cells support pull-up and pull-down driver information using the `driver_type` attribute with the values `pull_up` or `pull_down`. Liberty syntax also supports conditional (programmable) pull-up and pull-down driver information for I/O pad cells. For more information about programmable driver types, see “Programmable Driver Type Functions” on page 3-16.

**Syntax**

```plaintext
driver_type : pull_up | pull_down | open_drain | open_source | bus_hold | resistive | resistive_0 | resistive_1 ;
```

**pull_up**

The pin is connected to power through a resistor. If it is a three-state output pin, it is in the Z state and its function is evaluated as a resistive 1 (H). If it is an input or inout pin and the node to which it is connected is in the Z state, it is considered an input pin at logic 1 (H). For a pull-up cell, the pin constantly stays at logic 1 (H).

**pull_down**

The pin is connected to ground through a resistor. If it is a three-state output pin, it is in the Z state and its function is evaluated as a resistive 0 (L). If it is an input or inout pin and the node to which it is connected is in the Z state, it is considered an input pin at logic 0 (L). For a pull-down cell, the pin constantly stays at logic 0 (L).
open_drain
The pin is an output pin without a pull-up transistor. Use this driver type only for off-chip output or inout pins representing pads. The pin goes to high impedance (Z) when its function is evaluated as logic 1.

open_source
The pin is an output pin without a pull-down transistor. Use this driver type only for off-chip output or inout pins representing pads. The pin goes to high impedance (Z) when its function is evaluated as logic 0.

bus_hold
The pin is a bidirectional pin on a bus holder cell. The pin holds the last logic value present at that pin when no other active drivers are on the associated net. Pins with this driver type cannot have function or three_state statements.

resistive
The pin is an output pin connected to a controlled pull-up or pull-down transistor with a control port EN. When EN is disabled, the pull-up or pull-down transistor is turned off and has no effect on the pin. When EN is enabled, a functional value of 0 evaluated at the pin is turned into a weak 0, and a functional value of 1 is turned into a weak 1, but a functional value of Z is not affected.

resistive_0
The pin is an output pin connected to power through a pull-up transistor that has a control port EN. When EN is disabled, the pull-up transistor is turned off and has no effect on the pin. When EN is enabled, a functional value of 1 evaluated at the pin turns into a weak 1, but a functional value of 0 or Z is not affected.

resistive_1
The pin is an output pin connected to ground through a pull-down transistor that has a control port EN. When EN is disabled, the pull-down transistor is turned off and has no effect on the pin. When EN is enabled, a functional value of 0 evaluated at the pin turns into a weak 0, but a functional value of 1 or Z is not affected.
Table 3-1 lists the driver types, their signal mappings, and the applicable pin types.

**Table 3-1  Pin Driver Types**

<table>
<thead>
<tr>
<th>Driver type</th>
<th>Signal mapping</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>pull_up</td>
<td>01Z-&gt;01H</td>
<td>in, out</td>
</tr>
<tr>
<td>pull_down</td>
<td>01Z-&gt;01L</td>
<td>in, out</td>
</tr>
<tr>
<td>open_drain</td>
<td>01Z-&gt;0ZZ</td>
<td>out</td>
</tr>
<tr>
<td>open_source</td>
<td>01Z-&gt;0Z1Z</td>
<td>out</td>
</tr>
<tr>
<td>bus_hold</td>
<td>01Z-&gt;01S</td>
<td>inout</td>
</tr>
<tr>
<td>resistive</td>
<td>01Z-&gt;LHZ</td>
<td>out</td>
</tr>
<tr>
<td>resistive_0</td>
<td>01Z-&gt;0HZ</td>
<td>out</td>
</tr>
<tr>
<td>resistive_1</td>
<td>01Z-&gt;L1Z</td>
<td>out</td>
</tr>
</tbody>
</table>

Keep the following concepts in mind when interpreting Table 3-1.

- The signal modifications a **driver_type** attribute defines divide into two categories, transformation and resolution.
  - Transformation specifies an actual signal transition from 0/1 to L/H/Z. This signal transition performs a function on an input signal and requires only a straightforward mapping.
  - Resolution resolves the value Z on an existing circuit node without actually performing a function and implies a constant (0/1) signal source as part of the resolution.

  In Table 3-1, the **pull_up**, **pull_down**, and **bus_hold** driver types define a resolution scheme. The remaining driver types define transformations.

Example 3-2 describes an output pin with a pull-up resistor and the bidirectional pin on a **bus_hold** cell.

**Example 3-2  Pin Driver Type Specifications**

```plaintext
cell (bus) {
  pin(Y) {
    direction : output ;
    driver_type : pull_up ;
    pulling_resistance : 10000 ;
  }
```

Bidirectional pads often require one driver type for the output behavior and another driver type associated with the input behavior. In such a case, define multiple driver types in one driver_type attribute, as shown here:

```markdown
driver_type : open_drain pull_up;
```

**Note:**

An n-channel open-drain pad is flagged with `open_drain`, and a p-channel open-drain pad is flagged with `open_source`.

### Programmable Driver Type Functions

Liberty syntax also supports conditional (programmable) pull-up and pull-down driver information for I/O pad cells. The programmable pin syntax has been extended to driver_type attribute values, such as `bus_hold`, `open_drain`, `open_source`, `resistive`, `resistive_0`, and `resistive_1`.

### Syntax

The following syntax supports programmable driver types in I/O pad cell models. Unlike the nonprogrammable driver type support, the programmable driver type support allows you to specify more than one driver type within a pin.

```markdown
pin (pin_name) { /* programmable driver type pin */
... pull_up_function : "function string";
pull_down_function : "function string";
bus_hold_function : "function string";
open_drain_function : "function string";
open_source_function : "function string";
resistive_function : "function string";
resistive_0_function : "function string";
resistive_1_function : "function string";
... }
```

The functions in Table 3-2 have been introduced on top of (as an extension of) the existing driver_type attribute to support programmable pins. These driver type functions help
model the programmable driver types. The same rules that apply to nonprogrammable driver types also apply to these functions.

Table 3-2  Programmable Driver Type Functions

<table>
<thead>
<tr>
<th>Programmable driver type</th>
<th>Applicable on pin types</th>
</tr>
</thead>
<tbody>
<tr>
<td>pull_up_function</td>
<td>Input, output and inout</td>
</tr>
<tr>
<td>pull_down_function</td>
<td>Input, output and inout</td>
</tr>
<tr>
<td>bus_hold_function</td>
<td>Inout</td>
</tr>
<tr>
<td>open_drain_function</td>
<td>Output and inout</td>
</tr>
<tr>
<td>open_source_function</td>
<td>Output and inout</td>
</tr>
<tr>
<td>resistive_function</td>
<td>Output and inout</td>
</tr>
<tr>
<td>resistive_0_function</td>
<td>Output and inout</td>
</tr>
<tr>
<td>resistive_1_function</td>
<td>Output and inout</td>
</tr>
</tbody>
</table>

Example

The following example models a programmable driver type in a I/O pad cell.

```
library(cond_pull_updown_example) {
  delay_model : table_lookup;

time_unit : 1ns;
voltage_unit : 1V;
capacitive_load_unit (1.0, pf);
current_unit : 1mA;

  cell(conditional_PU_PD) {
    dont_touch : true;
    dont_use : true;
    pad_cell : true;
    pin(IO) { 
      drive_current : 1;
      min_capacitance : 0.001;
      min_transition : 0.0008;
      is_pad : true;
      direction : inout;
      max_capacitance : 30;
      max_fanout : 2644;
      function : "((A*ETM')+(TA*ETM))";
      three_state : "((TEN*ETM')+(EN*ETM))";
      pull_up_function : "(!P1 * !P2)";
    }
  }
}```
pull_down_function : "( P1 * P2)" ;
capacitance : 2.06649 ;
timing() {
  related_pin : "IO A ETM TEN TA" ;
cell_rise(scalar) {
    values("0" ) ;
  }
  rise_transition(scalar) {
    values("0" ) ;
  }
  cell_fall(scalar) {
    values("0" ) ;
  }
  fall_transition(scalar) {
    values("0" ) ;
  }
}
timing() {
  timing_type : three_state_disable;
  related_pin : "EN ETM TEN" ;
cell_rise(scalar) {
    values("0" ) ;
  }
  rise_transition(scalar) {
    values("0" ) ;
  }
  cell_fall(scalar) {
    values("0" ) ;
  }
  fall_transition(scalar) {
    values("0" ) ;
  }
}

pin(ZI) {
  direction : output;
  function : "IO" ;
timing() {
  related_pin : "IO" ;
cell_rise(scalar) {
    values("0" ) ;
  }
  rise_transition(scalar) {
    values("0" ) ;
  }
  cell_fall(scalar) {
    values("0" ) ;
  }
  fall_transition(scalar) {
    values("0" ) ;
  }
}
pin(A) {
  direction : input;
  capacitance : 1.0;
}

pin(EN) {
  direction : input;
  capacitance : 1.0;
}

pin(TA) {
  direction : input;
  capacitance : 1.0;
}

pin(TEN) {
  direction : input;
  capacitance : 1.0;
}

pin(ETM) {
  direction : input;
  capacitance : 1.0;
}

pin(P1) {
  direction : input;
  capacitance : 1.0;
}

pin(P2) {
  direction : input;
  capacitance : 1.0;
}

} /* End cell conditional_PU_PD */
} /* End Library */

driver_waveform Simple Attribute

The driver_waveform attribute specified at the pin level is the same as the
driver_waveform attribute specified at the cell level. For more information, see
“driver_waveform Simple Attribute” on page 2-11.

driver_waveform_rise and driver_waveform_fall
Simple Attributes

The driver_waveform_rise and driver_waveform_fall attributes specified at the pin
level are the same as the driver_waveform_rise and driver_waveform_fall attributes
specified at the cell level. For more information, see “driver_waveform_rise and
driver_waveform_fall Simple Attributes” on page 2-12.

fall_capacitance Simple Attribute

Defines the load for an input and inout pin when its signal is falling.
Setting a value for the fall_capacitance attribute requires that a value for rise_capacitance also be set, and setting a value for rise_capacitance attribute requires that a value for the fall_capacitance also be set.

**Syntax**

```plaintext
fall_capacitance : float ;
```

**float**

A floating-point number that represents the internal fanout of the input pin. Typical units of measure for fall_capacitance include picofarads and standardized loads.

**Example**

The following example defines the A and B pins in an AND cell, each with a fall_capacitance of one unit, a rise_capacitance of two units, and a capacitance of two units.

```plaintext
cell (AND) {
    area : 3 ;
    pin (A,B) {
        direction   : input ;
        fall_capacitance : 1 ;
        rise_capacitance : 2 ;
        capacitance : 2 ;
    }
}
```

**fall_current_slope_after_threshold Simple Attribute**

The fall_current_slope_after_threshold attribute represents a linear approximation of the change in current with respect to time, from the point at which the rising transition reaches the threshold to the end of the transition.

**Syntax**

```plaintext
fall_current_slope_after_threshold : value_float ;
```

**value**

A floating-point number that represents the change in current.

**Example**

```plaintext
fall_current_slope_after_threshold : 0.07 ;
```
**fall_current_slope_before_threshold Simple Attribute**

The `fall_current_slope_before_threshold` attribute represents a linear approximation of the change in current with respect to time from the beginning of the falling transition to the threshold point.

**Syntax**

```markdown
fall_current_slope_before_threshold : value_float ;
```

- **value**
  - A floating-point number that represents the change in current.

**Example**

```markdown
fall_current_slope_before_threshold : -0.14 ;
```

**fall_time_after_threshold Simple Attribute**

The `fall_time_after_threshold` attribute gives the time interval from the threshold point of the falling transition to the end of the transition.

**Syntax**

```markdown
fall_time_after_threshold : value_float ;
```

- **value**
  - A floating-point number that represents the time interval.

**Example**

```markdown
fall_time_after_threshold : 1.8 ;
```

**fall_time_before_threshold Simple Attribute**

The `fall_time_before_threshold` attribute gives the time interval from the beginning of the falling transition to the point at which the threshold is reached.

**Syntax**

```markdown
fall_time_before_threshold : value_float ;
```

- **value**
  - A floating-point number that represents the time interval.

**Example**

```markdown
fall_time_before_threshold : 0.55 ;
```
fanout_load Simple Attribute

The **fanout_load** attribute gives the internal fanout load for an input pin.

Syntax

```plaintext
fanout_load : value_float ;

value

A floating-point number that represents the internal fanout of the input pin. There are no fixed units for **fanout_load**. Typical units are standard loads or pin count.
```

Example

```plaintext
pin (B) {
    direction : input ;
    fanout_load : 2.0 ;
}
```

fault_model Simple Attribute

The differential I/O feature enables an input noninverting pin to inherit the timing information and all associated attributes of an input inverting pin in the same pin group designated with the **complementary_pin** attribute.

The **fault_model** attribute defines a two-value string when both differential inputs are driven to the same value. The first value represents the value when both input pins are at logic 0, and the second value represents the value when both input pins are at logic 1.

For details on the **complementary_pin** attribute, see “**complementary_pin Simple Attribute**” on page 3-10.

Syntax

```plaintext
fault_model : "two-value string" ;

two-value string

Two values that define the value of the differential signals when both inputs are driven to the same value. The first value represents the value when both input pins are at logic 0; the second value represents the value when both input pins are at logic 1. Valid values for the two-value string are any two-value combinations made up of 0, 1, and x.

If you do not enter a **fault_model** attribute value, the signal pin value goes to x when both input pins are 0 or 1.
```

Example

```plaintext
cell (diff_buffer) {
    ...
    pin (A) { /* noninverting pin |
```
direction : input;
complementary_pin : ("DiffA")
fault_model : "1x";
}
}

**function Simple Attribute**

The function attribute describes the value of a pin or bus.

**Pin Names as function Statement Arguments**

The function attribute in a pin group defines the value of an output pin or inout pin in terms of the input pins or inout pins in the cell group.

**Syntax**

```
function : "Boolean expression" ;
```

Table 3-3 lists the valid Boolean operators in a function statement. The precedence of the operators is left to right, with inversion performed first, then XOR, then AND, then OR.

**Table 3-3 Valid Boolean Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*</code></td>
<td>invert previous expression</td>
</tr>
<tr>
<td><code>!</code></td>
<td>invert following expression</td>
</tr>
<tr>
<td><code>^</code></td>
<td>logical XOR</td>
</tr>
<tr>
<td><code>*</code></td>
<td>logical AND</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>logical AND</td>
</tr>
<tr>
<td>space</td>
<td>logical AND</td>
</tr>
<tr>
<td><code>+</code></td>
<td>logical OR</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>1</code></td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td><code>0</code></td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>

The following example describes pin Q with the function A OR B.
**Example**

```plaintext
pin(Q) {
    direction : output;
    function : "A + B";
}
```

**Note:**
Pin names beginning with a number, and pin names containing special characters, must be enclosed in double quotation marks preceded by a backslash (\), as shown here:

```plaintext
function : " "1A" + "1B" ";
```

The absence of a backslash causes the quotation marks to terminate the function statement.

The following function statements all describe 2-input multiplexers.

```plaintext
function : "A S + B S'";
function : "A & S | B & !S";
function : "(A * S) + (B * S')";
```

The parentheses are optional. The operators and operands are separated by spaces.

**Grouped Pins in a function Statement**

Grouped pins can be used as variables in the function attribute statement. See “bundle Group” on page 2-32 and “bus Group” on page 2-39. Also see the “Defining Core Cells” and “Defining Sequential Cells” chapters in the Liberty User Guide, Vol. 1.

In function attribute statements that use bus or bundle names, all the variables must be either buses or bundles of the same width, or a single bus pin.

The range for the buses or bundles is valid if the range you define contains the same number of members (pins) as the other buses or bundles in the same expression. You can reverse the bus order by listing the member numbers in reverse (high:low) order.

When the function attribute of a cell with group input pins is a combinational logic function of grouped variables only, the logic function is expanded to apply to each set of output grouped pins independently. For example, if A, B, and Z are defined as buses of the same width and the function statement for output Z is

```plaintext
function : "(A & B)";
```

the function for Z[0] is interpreted as

```plaintext
function : "(A[0] & B[0])";
```

and the function for Z[1] is interpreted as

```plaintext
```
If a bus and a single pin are in the same function attribute, the single pin is distributed across all members of the bus. For example, if A and Z are buses of the same width, B is a single pin, and the function statement for the Z output is

```
function : "(A & B)" ;
```

the function for Z[0] is interpreted as

```
function : "(A[0] & B)" ;
```

Likewise, the function for Z[1] is interpreted as

```
```

**has_builtin_pad Simple Attribute**

Use this attribute in the case of an FPGA containing an ASIC core connected to the chip’s port. When set to true, this attribute specifies that an output pin has a built-in pad, which prevents pads from being inserted on the net connecting the pin to the chip’s port.

**Syntax**

```
has_builtin_pad : Boolean ;
```

**Example**

```
has_builtin_pad : true ;
```

**has_pass_gate Simple Attribute**

The has_pass_gate simple Boolean attribute can be defined in a pin group to indicate whether the pin is internally connected to at least one pass gate.

**Syntax**

```
has_pass_gate : Boolean expression ;
```

**Boolean expression**

Valid values are true and false.

**hysteresis Simple Attribute**

The hysteresis attribute allows the pad to accommodate longer transition times, which are more subject to noise problems.

**Syntax**

```
hysteresis : true | false ;
```
When the attribute is set to true, the voltages are actual transition points. When the hysteresis attribute is omitted, the value is assumed to be false and no hysteresis occurs.

Example

```plaintext
hysteresis : true ;
```

illegal_clamp_condition Simple Attribute

The illegal_clamp_condition attribute specifies the invalid condition for the enable pins of an enable level-shifter or isolation cell. If the illegal_clamp_condition attribute is not specified, the invalid condition does not exist.

Syntax

```plaintext
illegal_clamp_condition : "Boolean expression" ;
```

Example

```plaintext
illegal_clamp_condition : "EN1 \* EN2" ;
```

input_map Simple Attribute

The input_map attribute maps the input, internal, or output pin names to input and internal node names defined in the statetable group.

Syntax

```plaintext
input_map : nameid ;
```

name

A string representing a name or a list of port names, separated by spaces, that correspond to the input pin names, followed by the internal node names.

Example

```plaintext
input_map : " D G R Q " ;
```

input_signal_level Simple Attribute

The input_signal_level attribute describes the voltage levels in the pin group of a cell with multiple power supplies.

The input_signal_level attribute is used for an input or inout pin definition. The output_signal_level attribute is used for an output or inout pin definition. If the input_signal_level or output_signal_level attribute is missing, you can apply the default power supply name to the cell.
To model CCS noise stages in multivoltage designs, use the `output_signal_level` and `input_signal_level` attributes to specify internal power supplies in the `ccsn_first_stage` and `ccsn_last_stage` groups, respectively.

**Syntax**

```plaintext
input_signal_level: name;
output_signal_level: name;
```

*name*

A string representing the name of the power supply already defined at the library level.

**Example**

```plaintext
input_signal_level: VDD1;
output_signal_level: VDD2;
```

**input_threshold_pct_fall** Simple Attribute

Use the `input_threshold_pct_fall` attribute to set the value of the threshold point on an input pin signal falling from 1 to 0. You can specify this attribute at the library level to set a default for all the pins.

**Syntax**

```plaintext
input_threshold_pct_fall: trip_pointfloat;
```

*trip_point*

A floating-point number between 0.0 and 100.0 that specifies the threshold point of an input pin signal falling from 1 to 0. The default is 50.0.

**Example**

```plaintext
input_threshold_pct_fall: 60.0;
```

**input_threshold_pct_rise** Simple Attribute

Use the `input_threshold_pct_rise` attribute to set the value of the threshold point on an input pin signal rising from 0 to 1. You can specify this attribute at the library level to set a default for all the pins.

**Syntax**

```plaintext
input_threshold_pct_rise: trip_pointfloat;
```

*trip_point*

A floating-point number between 0.0 and 100.0 that specifies the threshold point of an input pin signal rising from 0 to 1. The default is 50.0.
Example
input_threshold_pct_rise : 40.0 ;

input_voltage Simple Attribute
You can define a special set of voltage thresholds in the library group with the input_voltage or output_voltage attribute. You can then apply the default voltage ranges in the group to selected cells with the input_voltage or output_voltage attribute in the pin definition.

Syntax
input_voltage : nameid ;
output_voltage : nameid ;

name
A string representing the name of the voltage range group defined at the library level. The input_voltage attribute is used for an input pin definition, and the output_voltage attribute is used for an output pin definition.

Example
input_voltage : CMOS_SCHMITT ;
output_voltage : GENERAL ;

internal_node Simple Attribute
The internal_node attribute describes the sequential behavior of an internal pin or an output pin. It indicates the relationship between an internal node in the statetable group and a pin of a cell. Each output or internal pin with the internal_node attribute can also have the optional input_map attribute.

Syntax
internal_node : pin_nameid ;

pin_name
Name of either an internal or output pin.

Example
internal_node : IQ ;

inverted_output Simple Attribute
Except in statetable cells, where it is required, the inverted_output attribute is an optional Boolean attribute that can be set for any output port. Set this attribute to true if the output from the pin is inverted. Set it to false if the output is not inverted.
Syntax

\texttt{inverted_output : Boolean expression ;}

Example

\texttt{inverted_output : true}

\textbf{is\_analog Attribute}

The \texttt{is\_analog} attribute identifies an analog signal pin as analog so it can be recognized by tools. The valid values for \texttt{is\_analog} are \texttt{true} and \texttt{false}. Set the \texttt{is\_analog} attribute to \texttt{true} at the pin level to specify that the signal pin is analog.

Syntax

The syntax for the \texttt{is\_analog} attribute is as follows:

\begin{verbatim}
    cell (cell_name) {
        ...
        pin (pin_name) {
            is\_analog: true | false ;
        ...
    }
\end{verbatim}

Example

The following example identifies the pin as an analog signal pin.

\begin{verbatim}
    pin(Analog) {
        direction : input;
        capacitance : 1.0 ;
        is\_analog : true;
    }
\end{verbatim}

\textbf{is\_pad Simple Attribute}

The \texttt{is\_pad} attribute indicates which pin represents the pad. The valid values are \texttt{true} and \texttt{false}. You can also specify the \texttt{is\_pad} attribute on PG pins.

Syntax

\texttt{is\_pad : Boolean expression ;}

This attribute must be used on at least one pin with a \texttt{pad\_cell} attribute.

Example

\begin{verbatim}
    cell(INBUF) {
        ...
        pad\_cell : true ;
        ...
    }
\end{verbatim}
pin(PAD) {
   direction : input ;
   is_pad : true ;
   ...
}

is_pll_reference_pin Attribute

The `is_pll_reference_pin` Boolean attribute tags a pin as a reference pin on the phase-locked loop. In a phase-locked loop cell group, the `is_pll_reference_pin` attribute should be set to true in only one input pin group.

Syntax

cell (cell_name) {
   is_pll_cell : true;
   pin (ref_pin_name) {
      is_pll_reference_pin : true;
      direction : output;
      ...
   }
   ...
}

Example

cell(my_pll) {
   is_pll_cell : true;
   pin( REFCLK ) {
      direction : input;
      is_pll_reference_pin : true;
   }
   ...
}

is_pll_feedback_pin Attribute

The `is_pll_feedback_pin` Boolean attribute tags a pin as a feedback pin on a phase-locked loop. In a phase-locked loop cell group, the `is_pll_feedback_pin` attribute should be set to true in only one input pin group.

Syntax

cell (cell_name) {
   is_pll_cell : true;
   pin (ref_pin_name) {
      is_pll_reference_pin : true;
      direction : output;
      ...
   }
   ...
}
pin (\textit{feedback\_pin\_name}) { 
    \textit{is\_pll\_feedback\_pin} : true;
    direction : output;
    ...
}

\textbf{Example}

\texttt{cell(my\_pll) { 
    \textit{is\_pll\_cell} : true;
    
    \texttt{pin( REFCLK ) { 
        direction : input;
        \textit{is\_pll\_reference\_pin} : true;
        
        \texttt{pin( FBKCLK ) { 
            direction : input;
            \textit{is\_pll\_feedback\_pin} : true;
            
            ...
        }
        ...
    }
    }
    }
}

\textbf{is\_pll\_output\_pin Attribute}

The \textit{is\_pll\_output\_pin} Boolean attribute tags a pin as an output pin on a phase-locked loop. In a phase-locked loop cell group, the \textit{is\_pll\_output\_pin} attribute should be set to true in one or more output pin groups.

\textbf{Syntax}

\texttt{cell (cell\_name) { 
    \textit{is\_pll\_cell} : true;
    \texttt{pin (ref\_pin\_name) { 
        \textit{is\_pll\_reference\_pin} : true;
        direction : output;
        ...
    }
    ...
    \texttt{pin (output\_pin\_name) { 
        \textit{is\_pll\_output\_pin} : true;
        direction : output;
        ...
    }
    }
}

\textbf{Example}

\texttt{cell(my\_pll) { 
    \textit{is\_pll\_cell} : true;
    
    \texttt{pin( REFCLK ) { 
        direction : input;
        
        \texttt{pin( FBKCLK ) { 
            direction : input;
            \textit{is\_pll\_feedback\_pin} : true;
            
            ...
        }
        ...
    }
    }
    }
}
is_pll_reference_pin : true;
}
...
pin (OUTCLK_1x) {
direction : output;
is_pll_output_pin : true;
timing() { /* Timing Arc */
related_pin: "REFCLK";
timing_type: combinational_rise;
timing_sense: positive_unate;
...
}

timing() { /* Timing Arc */
related_pin: "REFCLK";
timing_type: combinational_fall;
timing_sense: positive_unate;
...
} /* End pin group */
} /* End cell group */

**is_unbuffered Simple Attribute**

The `is_unbuffered` attribute specifies the pin as unbuffered. You can specify this optional attribute on the pins of any library cell. The default is `false`.

**Syntax**

```plaintext
is_unbuffered : Boolean expression ;
```

**Boolean expression**

Valid values are `true` and `false`.

**isolation_cell_enable_pin Simple Attribute**

The `isolation_cell_enable_pin` attribute specifies the enable input pin of an isolation cell including a clock isolation cell. You can also use this attribute to model isolation enable pins of low-power complex macro cells. For more information about isolation cells, see "**is_isolation_cell Simple Attribute**" on page 2-17.

**Syntax**

```plaintext
isolation_cell_enable_pin : boolean_expression ;
```

**Boolean expression**

Valid values are `true` and `false`. 
Example

isolation_cell_enable_pin : true ;

**isolation_cell_data_pin Simple Attribute**

The isolation_cell_data_pin attribute identifies the data pin of any isolation cell. You can also use this attribute to model isolation pins of low-power complex macro cells.

The valid values of this attribute are true or false. If this attribute is not specified, all the input pins of the isolation cell are considered to be data pins.

**Syntax**

isolation_cell_data_pin : boolean_expression ;

**Boolean expression**

Valid values are true and false.

Example

isolation_cell_data_pin : true ;

**permit_power_down Simple Attribute**

The permit_power_down attribute identifies the power pin of an isolation cell, that can be powered down. You can also use this attribute to model isolation power pins of low-power complex macro cells.

The valid values of this attribute are true or false. The default is true, meaning that the power pin is allowed to power down in the isolation mode.

**Syntax**

permit_power_down : boolean_expression ;

**Boolean expression**

Valid values are true or false.

Example

permit_power_down : true ;

**alive_during_partial_power_down Simple Attribute**

The alive_during_partial_power_down attribute indicates that the pin with this attribute is active while the isolation cell is partially powered down, and the UPF isolation supply set is used for the power reference instead of the power and ground rails. You can also use this attribute in low-power complex macro cells.
The valid values of this attribute are true and false. The default is true.

Syntax
alive_during_partial_power_down : boolean_expression ;

Boolean expression
Valid values are true or false.

Example
alive_during_partial_power_down : true ;

is_isolated Simple Attribute
The is_isolated attribute indicates that a pin, bus, or bundle of a cell is internally isolated and does not require the insertion of an external isolation cell. The attribute is applicable to pins of macro cells and I/O pad cells.

The valid values are true and false. The default is false.

Syntax
is_isolated : boolean_expression ;

Boolean expression
Valid values are true or false.

Example
is_isolated : true ;

isolation_enable_condition Simple Attribute
The isolation_enable_condition attribute specifies the isolation condition for internally-isolated pins, buses, and bundles of a cell. When this attribute is defined in a pin group, the corresponding Boolean expression can include only input and inout pins. Do not include the output pins of an internally isolated cell in the Boolean expression.

The attribute is applicable to pins of macro cells and I/O pad cells.

When the isolation_enable_condition attribute is defined in a bus or bundle group, the corresponding Boolean expression can include pins, and buses and bundles of the same bit-width. For example, when the Boolean expression includes a bus and a bundle, both of them must have the same bit-width.

Pins, buses, and bundles that have the isolation_enable_condition attribute must also have the always_on attribute.
Syntax

isolation_enable_condition : Boolean expression;

Example

isolation_enable_condition : "en";

**level_shifter_data_pin Simple Attribute**

The `level_shifter_enable_pin` attribute specifies the input data pin on a level shifter cell. You can also use this attribute in low-power complex macro modeling.

For more information about level-shifter cells, see “is_level_shifter Simple Attribute” on page 2-17.

Syntax

level_shifter_enable_pin : boolean_expression;

*Boolean expression*

Valid values are `true` and `false`.

Example

level_shifter_enable_pin : true;

**level_shifter_enable_pin Simple Attribute**

The `level_shifter_enable_pin` attribute specifies the enable input pin on a level shifter cell. You can also use this attribute in low-power complex macro modeling.

For more information about level-shifter cells, see “is_level_shifter Simple Attribute” on page 2-17.

Syntax

level_shifter_enable_pin : boolean_expression;

*Boolean expression*

Valid values are `true` and `false`.

Example

level_shifter_enable_pin : true;

**map_to_logic Simple Attribute**

The `map_to_logic` attribute specifies which logic level to tie a pin when a power-switch cell functions as a normal cell. For more information about power-switch cells, see “power_gating_cell Simple Attribute” on page 2-20.
Syntax
map_to_logic : boolean_expression ;

Boolean expression
Valid values are 1 and 0.

Example
map_to_logic : 1 ;

max_capacitance Simple Attribute
The max_capacitance attribute defines the maximum total capacitive load an output pin can drive. Define this attribute only for an output or inout pin.

Syntax
max_capacitance : value ;

value
A floating-point number that represents the capacitive load.

Example
max_capacitance : 1 ;

max_fanout Simple Attribute
The max_fanout attribute defines the maximum fanout load that an output pin can drive.

Syntax
max_fanout : value_float ;

value
A floating-point number that represents the number of fanouts the pin can drive. There are no fixed units for max_fanout. Typical units are standard loads or pin count.

Example
In the following example, pin X can drive a fanout load of no more than 11.0.

pin (X) {
    direction : output ;
    max_fanout : 11.0 ;
}
max_transition Simple Attribute

The `max_transition` attribute defines a design rule constraint for the maximum acceptable transition time of an input or output pin.

Syntax

```
max_transition : value_float ;
```

`value`

A floating-point number in units consistent with other time values in the library.

Example

The following example shows a `max_transition` time of 4.2:

```
max_transition : 4.2 ;
```

min_capacitance Simple Attribute

The `min_capacitance` attribute defines the minimum total capacitive load an output pin should drive. Define this attribute only for an output or inout pin.

Syntax

```
min_capacitance : value_float ;
```

`value`

A floating-point number that represents the capacitive load.

Example

```
min_capacitance : 1 ;
```

min_fanout Simple Attribute

The `min_fanout` attribute defines the minimum fanout load that an output pin should drive. There are no fixed units for `min_fanout`. Typical units are standard loads or pin count.

Syntax

```
min_fanout : value_float ;
```

`value`

A floating-point number that represents the minimum number of fanouts the pin can drive.
Example

In the following example, pin X can drive a fanout load of no less than 3.0.

```plaintext
pin (X) {
  direction : output ;
  min_fanout : 3.0 ;
}
```

**min_period Simple Attribute**

Placed on the clock pin of a flip-flop or latch, the `min_period` attribute specifies the minimum clock period required for the input pin.

**Syntax**

```plaintext
min_period : value\text{float} ;
value
```

A floating-point number indicating a time unit.

**Example**

```plaintext
pin (CLK4) {
  direction : input ;
  capacitance : 1 ;
  clock : true ;
  min_period : 26.0 ;
}
```

**min_pulse_width_high Simple Attribute**

The VHDL library generator uses the optional `min_pulse_width_high` and `min_pulse_width_low` attributes for simulation.

**Syntax**

```plaintext
min_pulse_width_high : value\text{float} ;
value
```

A floating-point number defined in units consistent with other time values in the library. It gives the minimum length of time the pin must remain at logic 1 (`min_pulse_width_high`) or logic 0 (`min_pulse_width_low`).

**Example**

The following example shows both attributes on a clock pin, indicating the minimum pulse width for a clock pin.

```plaintext
pin (CLK) {
  direction : input ;
  min_fanout : 3.0 ;
  min_period : 26.0 ;
  min_pulse_width_high : 2.0 ;
  min_pulse_width_low : 1.0 ;
}
```
capacitance : 1 ;
min_pulse_width_high : 3 ;
min_pulse_width_low : 3 ;
}

**min_pulse_width_low Simple Attribute**

For information about using the `min_pulse_width_low` attribute, see the description of the `min_pulse_width_high Simple Attribute`.

**multicell_pad_pin Simple Attribute**

The `multicell_pad_pin` attribute indicates which pin on a cell should be connected to another cell to create the correct configuration.

**Syntax**

```plaintext
multicell_pad_pin : true | false ;
```

Use this attribute for all pins on a pad cell or auxiliary pad cell that are connected to another cell.

**Example**

```plaintext
multicell_pad_pin : true ;
```

**nextstate_type Simple Attribute**

In a pin group, the `nextstate_type` attribute defines the type of the `next_state` attribute. You define a `next_state` attribute in an `ff` group or an `ff_bank` group.

**Note:**

Specify a `nextstate_type` attribute to ensure that the synchronous set (or synchronous reset) pin and the D pin of a sequential cell are not swapped when the design is instantiated.

**Syntax**

```plaintext
nextstate_type : data | preset | clear | load | scan_in | scan_enable ;
```

where

- **data**
  
  Identifies the pin as a synchronous data pin. This is the default value.

- **preset**
  
  Identifies the pin as a synchronous preset pin.
clear
  Identifies the pin as a synchronous clear pin.
load
  Identifies the pin as a synchronous load pin.
scan_in
  Identifies the pin as a synchronous scan-in pin.
scan_enable
  Identifies the pin as a synchronous scan-enable pin.

Any pin with the nextstate_type attribute must be in the next_state function. A
consistency check is also made between the pin’s nextstate_type attribute and the
next_state function.

Example 2-11 on page 2-71 shows a nextstate_type attribute in a bundle group.

**output_signal_level Simple Attribute**
See “input_signal_level Simple Attribute” on page 3-26.

**output_signal_level_high Simple Attribute**
The output_signal_level_high and output_signal_level_low attributes specify the
actual output voltages of an output pin after a transition.

You can also define the output_signal_level_low and output_signal_level_high
attributes in timing arcs. See “output_signal_level_high Simple Attribute” on page 3-98.

**output_signal_level_low Simple Attribute**
See “output_signal_level_high Simple Attribute” on page 3-40.

**output_voltage Simple Attribute**
See “input_voltage Simple Attribute” on page 3-28.

**pg_function Simple Attribute**
The pg_function attribute is used for the coarse-grain switch cells’ virtual VDD output pins
to represent the propagated power level through the switch as a function of input pg_pins.
This is a logical buffer and is useful where the VDD and VSS connectivity might be
erroneously reversed.
Syntax
pg_function : "function_string" ;

Example
pg_function : "VDD" ;

pin_func_type Simple Attribute
The pin_func_type attribute describes the functionality of a pin.

Syntax
pin_func_type : clock_enable | active_high | active_low |
active_rising | active_falling ;

- clock_enable
  Enables the clocking mechanism.

- active_high and active_low
  Describes the clock active edge or the level of the enable pin of the latches.

- active_rising and active_falling
  Describes the clock active edge or level of the clock pin of the flip-flops.

Example
pin_func_type : clock_enable ;

power_down_function Simple Attribute
The power_down_function string attribute specifies the Boolean condition under which the cell's output pin is switched off by the state of the power and ground pins (when the cell is in off mode due to the external power pin states). If power_down_function is evaluated as 1, X is assumed on the pin, meaning that the pin is assumed to be in an unknown state.

You specify the power_down_function attribute for combinational and sequential cells. For simple or complex sequential cells, power_down_function also determines the condition of the cell's internal state. Knowing the sequential cell's internal state is necessary for formal verification tools when they perform equivalence checking because the internal state is what is verified.

Syntax
power_down_function : function_string ;

Example
power_down_function : "!VDD + VSS";
prefer_tied Simple Attribute

The prefer_tied attribute describes an input pin of a flip-flop or latch. It indicates what the library developer wants this pin connected to.

Syntax

prefer_tied : "0" | "1" ;

Example

The following example shows a prefer_tied attribute on a test-enable pin.

```plaintext
pin(TE) {
    direction : input;
    prefer_tied : "0" ;
}
```

primary_output Simple Attribute

The primary_output attribute describes the primary output pin of a device that has more than one output pin for a particular phase of the output signal. When set to true, it indicates that one of the output pins is the primary output pin.

Syntax

primary_output : true | false ;

pulling_current Simple Attribute

The pulling_current attribute defines the current-drawing capability of a pull-up or pull-down device on a pin. This attribute can be used for pins with the driver_type attribute set to pull_up or pull_down.

Syntax

pulling_current : current value ;

current value

If you characterize your pull-up or pull-down devices in terms of the current drawn during nominal operating conditions, use pulling_current instead of pulling_resistance.

Example

```plaintext
pin(Y) {
    direction : output ;
    driver_type : pull_up ;
    pulling_resistance : 1000 ;
    ...
}
```
pulling_resistance Simple Attribute

The `pulling_resistance` attribute defines the resistance strength of a pull-up or pull-down device on a pin. This attribute can be used for pins with the `driver_type` attribute set to `pull_up` or `pull_down`.

**Syntax**

```plaintext
pulling_resistance : resistance value ;
```

**resistance value**

The resistive strength of the pull-up or pull-down device.

**Example**

```plaintext
pin(Y) {
  direction : output ;
  driver_type : pull_up ;
  pulling_resistance : 1000 ;
  ...
}
```

pulse_clock Simple Attribute

Use the `pulse_clock` attribute to model edge-derived clocks at the pin level.

**Syntax**

```plaintext
pulse_clock : pulse_typeenum ;
```

**pulse_type**

The valid values are `rise_triggered_high_pulse`, `rise_triggered_low_pulse`, `fall_triggered_high_pulse`, and `fall_triggered_low_pulse`.

**Example**

```plaintext
pin(Y) {
  ...
  pulse_clock : rise_triggered_low_pulse ;
  ...
}
```

related_ground_pin Simple Attribute

The `related_power_pin` and `related_ground_pin` attributes are defined at the pin level for output, input, and inout pins. The `related_power_pin` and `related_ground_pin` attributes are used to associate a predefined power and ground pin with the signal pin, in which they are defined. This behavior only applies to standard cells. For special cells, you must specify this relationship explicitly.
The `pg_pin` groups are mandatory for each cell. Because a cell must have at least one `primary_power` and at least one `primary_ground` pin, a default `related_power_pin` and `related_ground_pin` always exists in any cell.

**Syntax**

```
related_ground_pin : pg_pin_name ;
```

*pg_pin_name*

Name of the related ground pin.

**Example**

```
pin(Y) {
    ... related_ground_pin : G1 ;
    ...}
```

**related_power_pin Simple Attribute**

For details about the `related_ground_pin` attribute, see “related_ground_pin Simple Attribute” on page 3-43.

**Syntax**

```
related_power_pin : pg_pin_nameid ;
```

*pg_pin_name*

Name of the related power pin.

**Example**

```
pin(Y) {
    ... related_power_pin : P1 ;
    ...}
```

**restore_action Simple Attribute**

The `restore_action` attribute specifies where the restore event occurs with respect to the restore control signal. Valid values are L (low), H (high), R (rise), and F (fall).

**Syntax**

```
restore_action : L | H | R | F ;
```

**Example**

```
restore_action : "R" ;
```
restore_condition Simple Attribute

The `restore_condition` attribute specifies the input condition during the restore event in a retention cell. This condition is checked at the value of the `restore_action` attribute. When the condition is not met, the cell is in an illegal state.

Syntax

```
restore_condition : "Boolean_expression" ;
```

Example

```
restore_condition : "!CK" ;
```

restore_edge_type Simple Attribute

The `restore_edge_type` attribute specifies the type of the edge of the restore signal where the output of the master-slave latch is restored. The `restore_edge_type` attribute supports the following edge-types: `edge_trigger`, `leading`, and `trailing`.

The `edge-trigger` edge-type specifies that the flip-flop data is restored immediately at the restore signal edge and can also begin normal operation immediately.

The `leading` edge-type specifies that the flip-flop data is available at leading edge of the restore signal. The flip-flop can begin normal operation after the trailing edge of the restore signal.

The `trailing` edge-type specifies that the flip-flop data is available at the trailing edge of the restore signal. The flip-flop also can begin normal operation after the trailing edge of the restore signal.

The default of the `restore_edge_type` attribute is `leading`.

Syntax

```
restore_edge_type : edge_trigger | leading | trailing ;
```

Example

```
restore_edge_type : "leading" ;
```

rise_capacitance Simple Attribute

Defines the load for an input or an inout pin when its signal is rising.

Setting a value for the `rise_capacitance` attribute requires that a value for `fall_capacitance` attribute also be set, and setting a value for `fall_capacitance` requires that a value for the `rise_capacitance` also be set.

Syntax

```
rise_capacitance : float ;
```
**float**

A floating-point number in units consistent with other capacitance specifications throughout the library. Typical units of measure for `rise_capacitance` include picofarads and standardized loads.

**Example**

The following example defines the A and B pins in an AND cell, each with a `fall_capacitance` of one unit, a `rise_capacitance` of two units, and a `capacitance` of two units.

```plaintext
cell (AND) {
  area : 3 ;
  pin (A,B) {
    direction   : input ;
    fall_capacitance : 1 ;
    rise_capacitance : 2 ;
    capacitance : 2 ;
  }
}
```

**rise_current_slope_after_threshold Simple Attribute**

The `rise_current_slope_after_threshold` attribute represents a linear approximation of the change in current over time from the point at which the rising transition reaches the threshold to the end of the transition.

**Syntax**

```plaintext
rise_current_slope_after_threshold : value float ;
```

**value**

A negative floating-point number that represents the change in current.

**Example**

```plaintext
rise_current_slope_after_threshold : -0.09 ;
```

**rise_current_slope_before_threshold Simple Attribute**

The `rise_current_slope_before_threshold` attribute represents a linear approximation of the change in current over time, from the beginning of the rising transition to the threshold point.

**Syntax**

```plaintext
rise_current_slope_before_threshold : value float ;
```
value
A positive floating-point number that represents the change in current.

Example
rise_current_slope_before_threshold : 0.18;

rise_time_after_threshold Simple Attribute
The rise_time_after_threshold attribute gives the time interval from the threshold point of the rising transition to the end of the transition.

Syntax
rise_time_after_threshold : valuefloat;

value
A floating-point number that represents the time interval for the rise transition from threshold to finish (after).

Example
rise_time_after_threshold : 2.4;

rise_time_before_threshold Simple Attribute
The rise_time_before_threshold attribute gives the time interval from the beginning of the rising transition to the point at which the threshold is reached.

Syntax
rise_time_before_threshold : valuefloat;

value
A floating-point number that represents the time interval for the rise transition from start to threshold (before).

Example
rise_time_before_threshold : 0.8;

save_action Simple Attribute
The save_action attribute specifies where the save event occurs with respect to the save signal. Valid values are L (low), H (high), R (rise), and F (fall). For level-sensitive latches (L or H), the save event occurs at the trailing edge of the save signal.
Syntax
save_action : L | H | R | F ;

Example
save_action : "R" ;

**save_condition Simple Attribute**

The `save_condition` attribute specifies the input condition during the save event in a retention cell. This condition is checked at a value specified by the `save_action` attribute. When the condition is not met, the cell is in an illegal state.

Syntax
save_condition : "Boolean_expression" ;

Example
save_condition : "!CK" ;

**signal_type Simple Attribute**

In a `test_cell` group, the `signal_type` attribute identifies the type of test pin.

Syntax
signal_type : test_scan_in | test_scan_in_inverted |
  test_scan_out | test_scan_out_inverted |
  test_scan_enable |
  test_scan_enable_inverted |
  test_scan_clock | test_scan_clock_a |
  test_scan_clock_b | test_clock ;

test_scan_in

Identifies the scan-in pin of a scan cell. The scanned value is the same as the value present on the scan-in pin. All scan cells must have a pin with either the `test_scan_in` or the `test_scan_in_inverted` attribute.

test_scan_in_inverted

Identifies the scan-in pin of a scan cell as having inverted polarity. The scanned value is the inverse of the value present on the scan-in pin.

For multiplexed flip-flop scan cells, the polarity of the scan-in pin is inferred from the latch or ff declaration of the cell itself. For other types of scan cells, clocked-scan, LSSD, and multiplexed flip-flop latches, it is not possible to give the ff or latch declaration of the entire scan cell. For these cases, you can use the `test_scan_in_inverted` attribute in the cell where the scan-in pin appears in the latch or ff declarations for the entire cell.
test_scan_out

Identifies the scan-out pin of a scan cell. The value present on the scan-out pin is the same as the scanned value. All scan cells must have a pin with either a test_scan_out or a test_scan_out_inverted attribute.

The scan-out pin corresponds to the output of the slave latch in the LSSD methodologies.

test_scan_out_inverted

Identifies the scan-out pin of a test cell as having inverted polarity. The value on this pin is the inverse of the scanned value.

test_scan_enable

Identifies the pin of a scan cell that, when high, indicates that the cell is configured in scan-shift mode. In this mode, the clock transfers data from the scan-in input to the scan-out input.

test_scan_enable_inverted

Identifies the pin of a scan cell that, when low, indicates that the cell is configured in scan-shift mode. In this mode, the clock transfers data from the scan-in input to the scan-out input.

test_scan_clock

Identifies the test scan clock for the clocked-scan methodology. The signal is assumed to be edge-sensitive. The active edge transfers data from the scan-in pin to the scan-out pin of a cell. The sense of this clock is determined by the sense of the associated timing arcs.

test_scan_clock_a

Identifies the a clock pin in a cell that supports a single-latch LSSD, double-latch LSSD, clocked LSSD, or auxiliary clock LSSD methodology. When the a clock is at the active level, the master latch of the scan cell can accept scan-in data. The sense of this clock is determined by the sense of the associated timing arcs.

test_scan_clock_b

Identifies the b clock pin in a cell that supports the single-latch LSSD, clocked LSSD, or auxiliary clock LSSD methodology. When the b clock is at the active level, the slave latch of the scan-cell can accept the value of the master latch. The sense of this clock is determined by the sense of the associated timing arcs.

test_clock

Identifies an edge-sensitive clock pin that controls the capturing of data to fill scan-in test mode in the auxiliary clock LSSD methodology.
If an input pin is used in both test and nontest modes (such as the clock input in the multiplexed flip-flop methodology), do not include a signal_type statement for that pin in the test_cell pin definition.

If an input pin is used only in test mode and does not exist on the cell that it scans and replaces, you must include a signal_type statement for that pin in the test_cell pin definition.

If an output pin is used in nontest mode, it needs a function statement. The signal_type statement is used to identify an output pin as a scan-out pin. In a test_cell group, the pin group for an output pin can contain a function statement, a signal_type attribute, or both.

Note:
You do not have to define a function or signal_type attribute in the pin group if the pin is defined in a previous test_cell group for the same cell.

Example
signal_type : test_scan_in ;

slew_control Simple Attribute
The slew_control attribute provides increasing levels of slew-rate control to slow down the transition rate. This attribute associates a coarse measurement of slew-rate control with the output pad cell.

Syntax
slew_control : low | medium | high | none ;

low, medium, high
Provides increasingly higher levels of slew-rate control.

none
Indicates that no slew-rate control is applied. If you do not use slew_control, none is the default.

This attribute limits peak noise by smoothing out fast output transitions, thus decreasing the possibility of a momentary disruption in the power or ground planes.

slew_lower_threshold_pct_fall Simple Attribute
Use the slew_lower_threshold_pct_fall attribute to set the value of the lower threshold point used in modeling the delay of a pin transitioning from 1 to 0. You can specify this attribute at the library level to set a default for all the pins.
Syntax
slew_lower_threshold_pct_fall : trip_pointvalue ;

trip_point
A floating-point number between 0.0 and 100.0 that specifies the lower threshold point used to model the delay of a pin falling from 1 to 0. The default value is 20.0.

Example
slew_lower_threshold_pct_fall : 30.0 ;

slew_lower_threshold_pct_rise Simple Attribute
Use the slew_lower_threshold_pct_rise attribute to set the value of the lower threshold point used in modeling the delay of a pin transitioning from 0 to 1. You can specify this attribute at the library level to set a default for all the pins.

Syntax
slew_lower_threshold_pct_rise : trip_pointvalue ;

trip_point
A floating-point number between 0.0 and 100.0 that specifies the lower threshold point used to model the delay of a pin rising from 0 to 1. The default value is 20.0.

Example
slew_lower_threshold_pct_rise : 30.0 ;

slew_upper_threshold_pct_fall Simple Attribute
Use the slew_upper_threshold_pct_fall attribute to set the value of the upper threshold point used in modeling the delay of a pin falling from 1 to 0. You can specify this attribute at the library level to set a default for all the pins.

Syntax
slew_upper_threshold_pct_fall : trip_pointvalue ;

trip_point
A floating-point number between 0.0 and 100.0 that specifies the upper threshold point used to model the delay of a pin transitioning from 1 to 0. The default value is 80.0.

Example
slew_upper_threshold_pct_fall : 70.0 ;
### Slew Upper Threshold Pct Rise Simple Attribute

Use the `slew_upper_threshold_pct_rise` attribute to set the value of the upper threshold point used to model the delay of a pin rising from 0 to 1. You can specify this attribute at the library level to set a default for all the pins.

**Syntax**

```plaintext
slew_upper_threshold_pct_rise : trip_pointvalue ;
```

**trip_point**

A floating-point number between 0.0 and 100.0 that specifies the upper threshold point used to model the delay of a pin transitioning from 0 to 1. The default value is 80.0.

**Example**

```plaintext
slew_upper_threshold_pct_rise : 70.0 ;
```

### State Function Simple Attribute

Use this attribute to define output logic. Ports in the `state_function` Boolean expression must be either input, three-state inout, or ports with an `internal_node` attribute. If the output logic is a function of only the inputs (IN), the output is purely combinational (for example, feedthrough output). A port in the `state_function` expression refers only to the non-three-state functional behavior of that port. An inout port in the `state_function` expression is treated only as an input port.

**Syntax**

```plaintext
state_function : "Boolean expression" ;
```

**Example**

```plaintext
state_function : "EN*X" ;
```

For a list of Boolean operators, see Table 3-3 on page 3-23.

### Std Cell Main Rail Simple Attribute

The `std_cell_main_rail` Boolean attribute is defined in a `primary_power` power pin. When the attribute is set to true, the power and ground pin is used to determine which side of the voltage boundary the power and ground pin is connected.

**Syntax**

```plaintext
std_cell_main_rail : true | false ;
```

**Example**

```plaintext
std_cell_main_rail : true ;
```
switch_function Simple Attribute

The `switch_function` string attribute identifies the condition when the attached design partition is turned off by the input `switch_pin`.

For a coarse-grain switch cell, the `switch_function` attribute can be defined at both controlled power and ground pins (virtual VDD and virtual VSS for `pg_pin`) and the output pins.

When the `switch_function` attribute is defined in the controlled power and ground pin, it is used to specify the Boolean condition under which the cell switches off (or drives an X to) the controlled design partitions, including the traditional signal input pins only (with no related power pins to this output).

Syntax

```
switch_function : function_string;
```

Example

```
switch_function : "CTL";
```

switch_pin Simple Attribute

The `switch_pin` attribute is a pin-level Boolean attribute. When it is set to `true`, it is used to identify the pin as the switch pin of a coarse-grain switch cell.

Syntax

```
switch_pin : valueBoolean;
```

Example

```
switch_pin : true;
```

test_output_only Simple Attribute

This attribute can be set for any output port described in statetable format.

For a flip-flop or latch, if a port is used for both function and test, provide the functional description using the `function` attribute. If a port is used for test only, omit the `function` attribute.

For a state table, a port always has a functional description. To specify that a port is for test only, set the `test_output_only` attribute to `true`.

Syntax

```
test_output_only : true | false;
```

Example

```
pin (scout) {
```
direction : output ;
signal_type : test_scan_out ;
test_output_only : true ;
}

three_state Simple Attribute
The three_state attribute defines a three-state output pin in a cell.

Syntax
three_state : "Boolean expression" ;

Boolean expression
An equation defining the condition that causes the pin to go to the high-impedance state. The syntax of this equation is the same as the syntax of the function attribute statement described in “function Simple Attribute” on page 3-23. The three_state attribute can be used in both combinational and sequential pin groups, with bus or bundle variables.

Example
three_state : "!E" ;

For a list of Boolean operators, see Table 3-3 on page 3-23.

x_function Simple Attribute
The x_function attribute describes the X behavior of an output or inout pin. X is a state other than 0, 1, or Z.

Syntax
x_function : "Boolean expression" ;

Example
x_function : "!an * ap" ;

Complex Attributes
This section describes the complex attributes you can use in a pin group.

fall_capacitance_range Complex Attribute
The fall_capacitance_range attribute specifies a range of values for pin capacitance during fall transitions.
Syntax
fall_capacitance_range (value_1float, value_2float) ;

value_1, value_2
Positive floating-point numbers that specify the range of values.

Example
fall_capacitance_range (0.0, 0.0) ;

power_gating_pin Complex Attribute

Note:
The power_gating_pin attribute has been replaced by the retention_pin attribute.
See “retention_pin Complex Attribute” on page 3-55.

The power_gating_pin attribute specifies a pair of pin values for a power-switch cell. The first value represents the power gating pin class. The second value specifies which logic level (default) the power-switch cell is tied to when the power-switch cell is functioning in normal mode. For more information about specifying power-switch cells, see “power_gating_cell Simple Attribute” on page 2-20.

Syntax
power_gating_pin ("power_pin_[1-5]", enumerated_type) ;

value_1
A string that represents one of five predefined classes of power gating pins: power_pin_[1-5].

value_2
An integer that specifies the default logic level for the pin when the power-switch cell functions as a normal cell.

Example
power_gating_pin ( "power_pin_1", 0 ) ;

retention_pin Complex Attribute

The retention_pin complex attribute identifies the retention pins of a retention cell. The attribute defines the following information:

• pin class
  Valid values:
  ◦ restore
Restores the state of the cell.

- save

Saves the state of the cell.

- save_restore

Saves and restores the state of the cell.

- disable value

Defines the value of the retention pin when the cell works in normal mode. The valid values are 0 and 1.

Syntax

```
retention_pin (pin_class, disable_value);
```

Example

```
retention_pin (save | restore | save_restore, enumerated_type);
```

rise_capacitance_range Complex Attribute

The `rise_capacitance_range` attribute specifies a range of values for pin capacitance during rise transitions.

Syntax

```
rise_capacitance_range (value_1float, value_2float);
```

Example

```
rise_capacitance_range (0.0, 0.0);
```

Group Statements

You can use the following group statements in a `pin` group:

```
ccsn_first_stage () {}
ccsn_last_stage () {}
dc_current () {}
electromigration () {}
input_ccb (string) {}
input_signal_swing () {}
internal_power () {}
max_capacitance () {}
max_transition () {}
```
Use the ccsn_first_stage group to specify CCS noise for the first stage of the channel-connected block (CCB).

A ccsn_first_stage or ccsn_last_stage group contains the following information:

- A set of channel-connected block parameters: the is_needed, is_inverting, stage_type, miller_cap_rise, miller_cap_fall, attributes
- The optional when and mode attributes for conditional data modeling
- The optional output_signal_level or input_signal_level attribute to model CCS noise stages of channel-connected blocks with internal power supplies
- A two-dimensional DC current table: dc_current group
- Two timing tables for rising and falling transitions: output_current_rise group, output_current_fall group
- Two noise tables for low and high propagated noise: propagated_noise_low group, propagated_noise_high group

Note that if the ccsn_first_stage and ccsn_last_stage groups are defined inside pin-level groups, then the ccsn_first_stage group can only be defined in an input pin or an inout pin, and the ccsn_last_stage group can only be defined in an output pin or an inout pin.

**Syntax**

```plaintext
library (name) {
...
  cell (name) {
    pin (name) {
      ...
      ccsn_first_stage () {
        is_needed : boolean;
        is_inverting : boolean;
        stage_type : stage_type_value;
        miller_cap_rise : float;
        miller_cap_fall : float;
      }
    }
  }
}...
```
dc_current (dc_current_template)
    index_1("float, ..."),
    index_2("float, ..."),
    values("float, ..."),
}

output_voltage_rise ()
    vector (output_voltage_template_name) {
        index_1(float);
        index_2(float);
        index_3("float, ..."),
        values("float, ..."),
    }
    ...
}

output_voltage_fall () {
    vector (output_voltage_template_name) {
        index_1(float);
        index_2(float);
        index_3("float, ..."),
        values("float, ..."),
    }
    ...
}

propagated_noise_low () {
    vector (propagated_noise_template_name) {
        index_1(float);
        index_2(float);
        index_3(float);
        index_4("float, ..."),
        values("float, ..."),
    }
    ...
}

propagated_noise_high () {
    vector (propagated_noise_template_name) {
        index_1(float);
        index_2(float);
        index_3(float);
        index_4("float, ..."),
        values("float, ..."),
    }
    ...
}

when : boolean expression;
Simple Attributes

is_inverting
is_needed
is_pass_gate
miller_cap_fall
miller_cap_rise
stage_type
when

Complex Attribute

mode

Group Statements

dc_current
output_voltage_fall
output_voltage_rise
propagated_noise_low
propagated_noise_rise

is_inverting Simple Attribute

Use the is_inverting attribute to specify whether the channel-connecting block is inverting. This attribute is mandatory if the is_needed attribute value is true. If the channel-connecting block is inverting, set the attribute to true. Otherwise, set the attribute to false. This attribute is different from the timing sense of a timing arc, which might consist of multiple channel-connecting blocks.

Syntax

is_inverting : valueBoolean ;

value

Valid values are true and false. Set the value to true when the channel-connecting block is inverting.

Example

is_inverting : true ;

is_needed Simple Attribute

Use the is_needed attribute to specify whether composite current source (CCS) noise modeling data is required.

Syntax

is_needed : valueBoolean ;
value

Valid values are true and false. The default is true. Set the value to false for cells such as diodes, antennas, and clod cells that do not need current-based data.

Example
is_needed : true ;

is_pass_gate Simple Attribute

The is_pass_gate attribute is defined in a ccsn_*_stage group, such as the ccsn_first_stage group, to indicate that the ccsn_*_stage information is modeled as a pass gate. The attribute is optional and the default is false.

Syntax
is_pass_gate : Boolean expression ;

Boolean expression

Valid values are true and false.

miller_cap_fall Simple Attribute

Use the miller_cap_fall attribute to specify the Miller capacitance value for the channel-connecting block.

Syntax
miller_cap_fall : valuefloat ;

value

A floating-point number representing the Miller capacitance value. The value must be greater or equal to zero.

Example
miller_cap_fall : 0.00084 ;

miller_cap_rise Simple Attribute

Use the miller_cap_rise attribute to specify the Miller capacitance value for the channel-connecting block.

Syntax
miller_cap_rise : valuefloat ;
value

A floating-point number representing the Miller capacitance value. The value must be greater or equal to zero.

Example
miller_cap_rise : 0.00055 ;

**mode Attribute**

The pin-based mode attribute is provided in the ccsn_first_stage and ccsn_last_stage groups for conditional data modeling. If the mode attribute is specified, mode_name and mode_value must be predefined in the mode_definition group at the cell level.

**stage_type Simple Attribute**

Use the stage_type attribute to specify the stage type of the channel-connecting block output voltage.

Syntax
stage_type : valueenum ;

value

The valid values are pull_up, in which the output voltage of the channel-connecting block is always pulled up (rising); pull_down, in which the output voltage of the channel-connecting block is always pulled down (falling); and both, in which the output voltage of the channel-connecting block is pulled up or down.

Example
stage_type : pull_up ;

**when Simple Attribute**

The when attribute is defined in both the pin-level and the timing-level ccsn_first_stage and ccsn_last_stage groups. Use this attribute to specify the condition under which the channel-connecting block data is applied.

Syntax
when : valueboolean ;

value

Result of a Boolean expression.
**mode Complex Attribute**

Use the `mode` attribute in the `ccsn_first_stage` and `ccsn_last_stage` groups to specify the noise parameters for a particular mode.

**Syntax**

```plaintext
mode (mode_definition_name, mode_value) ;
```

**Example**

```plaintext
mode (rw, read) ;
```

**dc_current Group**

Use the `dc_current` group to specify the input and output voltage values of a two-dimensional current table for a channel-connecting block.

**Syntax**

```plaintext
dc_current( dc_current_templateid ) { }
index_1 ("float, ..., float") ;
index_2 ("float, ..., float") ;
values ("float, ..., float") ;
```

**dc_current_template**

The name of the dc current lookup table.

Use `index_1` to represent the input voltage and `index_2` to represent the output voltage. The `values` attribute of the group lists the relative channel-connecting block DC current values in library units measured at the channel-connecting block output node.

**output_voltage_fall Group**

Use the `output_voltage_fall` group to specify vector groups that describe three-dimensional `output_voltage` tables of the channel-connecting block whose output node’s voltage values are falling.

```plaintext
output_voltage_fall ( ) { }

vector (output_voltage_template_name) {
    index_1(float);
    index_2(float);
    index_3("float, ...");
    values("float, ...");
```

**Complex Attributes**

- `index_1`
- `index_2`
index_3
values

Specify the following attributes in the vector group: The index_1 attribute lists the input_net_transition (slew) values in library time units. The index_2 attribute lists the total_output_net_capacitance (load) values in library capacitance units. The index_3 attribute lists the sampling time values in library time units. The values attribute lists the voltage values, in library voltage units, that are measured at the channel-connecting block output node.

output_voltage_rise Group

Use the output_voltage_rise group to specify vector groups that describe three-dimensional output_voltage tables of the channel-connecting block whose output node’s voltage values are rising.

For details, see the output_voltage_fall group description.

propagated_noise_high Group

The propagated_noise_high group uses vector groups to specify the three-dimensional output_voltage tables of the channel-connecting block whose output node’s voltage values are rising.

propagated_noise_high ( ) {

    vector (output_voltage_template_name) {
        index_1(float);
        index_2(float);
        index_3(float);
        index_4(“float, ...”);
        values(“float, ...”);
    }

Complex Attributes

index_1
index_2
index_3
index_4
values

Specify the following attributes in the vector group: The index_1 attribute lists the input_noise_height values in library voltage units. The index_2 attribute lists the input_noise_width values in library time units. The index_3 attribute lists the total_output_net_capacitance values in library capacitance units. The index_4 attribute lists the sampling time values in library time units. The values attribute lists the voltage values, in library voltage units, that are measured at the channel-connecting block output node.
propagated_noise_low Group

Use the `propagated_noise_low` group to specify the three-dimensional output_voltage tables of the channel-connecting block whose output node’s voltage values are falling.

For details, see the "propagated_noise_high Group" on page 3-63.

ccsn_last_stage Group

Use the `ccsn_last_stage` group to specify composite current source (CCS) noise for the last stage of the channel-connecting block.

For details, see “ccsn_first_stage Group” on page 3-57.

char_config Group

Use the `char_config` group in the `pin` group to specify the characterization settings of the library-cell pins.

**Syntax**

```
pin(pin_name) {
    char_config() {
        /* characterization configuration attributes */
    }
    ... ...
}
```

**Simple Attributes**

- `internal_power_calculation`
- `three_state_disable_measurement_method`
- `three_state_disable_current_threshold_abs`
- `three_state_disable_current_threshold_rel`
- `three_state_disable_monitor_node`
- `three_state_cap_add_to_load_index`
- `ccs_timing_segment_voltage_tolerance_rel`
- `ccs_timing_delay_tolerance_rel`
- `ccs_timing_voltage_margin_tolerance_rel`
- `receiver_capacitance1_voltage_lower_threshold_pct_rise`
- `receiver_capacitance1_voltage_upper_threshold_pct_rise`
- `receiver_capacitance1_voltage_lower_threshold_pct_fall`
- `receiver_capacitance1_voltage_upper_threshold_pct_fall`
- `receiver_capacitance2_voltage_lower_threshold_pct_rise`
- `receiver_capacitance2_voltage_upper_threshold_pct_rise`
- `receiver_capacitance2_voltage_lower_threshold_pct_fall`
- `receiver_capacitance2_voltage_upper_threshold_pct_fall`
- `capacitance_voltage_lower_threshold_pct_rise`
- `capacitance_voltage_lower_threshold_pct_fall`
capacitance_voltage_upper_threshold_pct_rise
capacitance_voltage_upper_threshold_pct_fall

**Complex Attributes**
driver_waveform
driver_waveform_rise
driver_waveform_fall
input_stimulus_transition
input_stimulus_interval
unrelated_output_net_capacitance
default_value_selection_method
default_value_selection_method_rise
default_value_selection_method_fall
merge_tolerance_abs
merge_tolerance_rel
merge_selection

**Example**
pin(pin1) {
  char_config() {
    driver_waveform_rise(delay, input_driver_rise);
  }
  ...
} /* pin */

For more information about the char_config group and the group attributes, see "char_config Group" on page 1-28.

electromigration Group
An electromigration group is defined in a pin group, as shown here:

library (name) {
  cell (name) {
    pin (name) {
      electromigration () {
        ... electromigration description ...
      }
    }
  }
}

**Simple Attributes**
related_pin : "name | name_list" ;/* path dependency */
related_bus_pins : "list of pins" ;/* list of pin names */
when : Boolean expression ;
Complex Attributes

```c
index_1 ("float, ..., float") ; /* optional */
index_2 ("float, ..., float") ; /* optional */
values ("float, ..., float") ;
```

Group Statement

```c
em_max_toggle_rate (em_template_name) {} 
```

related_pin Simple Attribute

The `related_pin` attribute associates the electromigration group with a specific input pin. The input pin’s input transition time is used as a variable in the electromigration lookup table.

If more than one input pin is specified in this attribute, the weighted input transition time of all input pins specified is used to index the electromigration table.

The pin or pins in the `related_pin` attribute denote the path dependency for the electromigration group. A particular electromigration group is accessed if the input pin or pins named in the `related_pin` attribute cause the corresponding output pin named in the `pin` group to toggle. All functionally related pins must be specified in a `related_pin` attribute if you specify two-dimensional tables.

Syntax

```c
related_pin : "name | name_list"
```

`name | name_list`

Name of input pin or pins.

Example

```c
related_pin : "A B" ;
```

related_bus_pins Simple Attribute

The `related_bus_pins` attribute associates the electromigration group with the input pin or pins of a specific bus group. The input pin’s input transition time is used as a variable in the electromigration lookup table.

If more than one input pin is specified in this attribute, the weighted input transition time of all input pins specified is used to index the electromigration table.

Syntax

```c
related_bus_pins : "name1 [name2 name3 ... ] " ;
```
Example

related_bus_pins : "A" ;

The pin or pins in the related_bus_pins attribute denote the path dependency for the electromigration group. A particular electromigration group is accessed if the input pin or pins named in the related_bus_pins attribute cause the corresponding output pin named in the pin group to toggle. All functionally related pins must be specified in a related_bus_pins attribute if two-dimensional tables are being used.

when Simple Attribute

The when attribute defines the enabling condition for the check in logic expression format.

Syntax

when : "Boolean expression" ;

For a list of Boolean operators, see Table 3-4 on page 3-77.

Example

when : "SE" ;

index_1 and index_2 Complex Attributes

You can use the index_1 optional attribute to specify the breakpoints of the first dimension of an electromigration table used to characterize cells for electromigration within the library. You can use the index_2 optional attribute to specify breakpoints of the second dimension of an electromigration table used to characterize cells for electromigration within the library.

You can overwrite the values entered for the em_lut_template group's index_1 by entering a value for the em_max_toggle_rate group's index_1. You can overwrite the value entered for the em_lut_template group's index_2 by entering a value for the em_max_toggle_rate group's index_2.

Syntax

index_1 ("float, ..., float") ; /* optional */
index_2 ("float, ..., float") ; /* optional */

float

For index_1, the floating-point numbers that specify the breakpoints of the first dimension of the electromigration table used to characterize cells for electromigration within the library. For index 2, the floating-point numbers that specify the breakpoints for the second dimension of the electromigration table used to characterize cells for electromigration within the library.
Example
index_1 ("0.0, 5.0, 20.0")
index_2 ("0.0, 1.0, 2.0");

values Complex Attribute
You use this complex attribute to specify the nets’ maximum toggle rates.

Syntax
values : ("float, ..., float") ;

float
Floating-point numbers that specify the net’s maximum toggle rates. The number can be
a list of \textit{nindex}_1 positive floating-point numbers if the table is one-dimensional and can
be \textit{nindex}_1 \times \textit{nindex}_2 positive floating-point numbers if the table is two-dimensional,
where \textit{nindex}_1 is the size of \textit{index}_1 and \textit{nindex}_2 is the size of \textit{index}_2, specified
for these two indexes in the \texttt{em\_max\_toggle\_rate} group or in the \texttt{em\_lut\_template}
group.

Example (One-Dimensional Table)
values : ("1.5, 1.0, 0.5") ;

Example (Two-Dimensional Table)
values : ("2.0, 1.0, 0.5", "1.5, 0.75, 0.33","1.0, 0.5, 0.15") ;

\texttt{em\_max\_toggle\_rate} Group
The \texttt{em\_max\_toggle\_rate} group is a pin-level group that is defined within the
\texttt{electromigration} group.

library (name) {
  cell (name) {
    pin (name) {
      electromigration () {
        \texttt{em\_max\_toggle\_rate} (\texttt{em\_template\_name}) {
          ... \texttt{em\_max\_toggle\_rate\_description}...
        }
      }
    }
  }
}

Simple Attribute
current\_type

Complex Attributes
index\_1 : ("float, ..., float") ; /*this attribute is optional*/
current_type Simple Attribute

The optional current_type attribute specifies the type of current for the
em_max_toggle_rate lookup table. Valid values are average, rms, and peak.

Syntax

current_type: average | rms | peak ;

Example

current_type: average ;

input_ccb Group

In referenced CCS noise modeling, use the input_ccb group to specify the CCS noise for
an input channel-connected block (CCB). You must name the input_ccb group so that it
can be referenced.

The input_ccb group includes all the attributes and subgroups of the “ccsn_first_stage
Group” on page 3-57. The input_ccb group also includes the related_ccb_node simple
attribute.

Syntax

input_ccb (input_ccb_name1) {
    related_ccb_node : "spice_node_name1";
    ...
}

Example

input_ccb("CCB_B") {
    related_ccb_node : "net1:15";
    ...
}

Simple Attributes

related_ccb_node

related_ccb_node Simple Attribute

The related_ccb_node attribute specifies the SPICE node in the subcircuit netlist is used
for the dc_current table characterization. The attribute is defined in the input_ccb group
of an input pin and the output_ccb group of an output pin.
output_ccb Group

In the referenced CCS noise model, use the `output_ccb` group to specify the CCS noise for an output channel-connected block (CCB). You must name the `output_ccb` group so that it can be referenced. For more information about the `output_ccb` group, see “input_ccb Group” on page 3-69.

The `output_ccb` group includes all the attributes and subgroups of the “ccsn_last_stage Group” on page 3-64.

internal_power Group

An `internal_power` group is defined in a `pin` group, as shown here:

```plaintext
library (name) {
  cell (name) {
    pin (name) {
      internal_power () {
        ... internal power description ...
      }
    }
  }
}
```

Note:

Either braces `{}` or quotation marks `" "` are valid syntax for values specified in internal power tables.

Simple Attributes

equal_or_opposite_output
falling_together_group
power_level
related_pin
related_pg_pin
rising_together_group
switching_interval
switching_together_group
when

Complex Attribute

mode

Group Statements

domain
fall_power (template name) {}
Syntax for One-Dimensional, Two-Dimensional, and Three-Dimensional Tables

You can define a one-, two-, or three-dimensional table in the internal_power group in either of the following ways:

- Using the power group
- Using a combination of the related_pin attribute, the fall_power group, and the rise_power group
- Using a combination of the related_pin attribute, the power group, and the equal_or_opposite attribute.

Note: Either braces {} or quotation marks " " are valid syntax for values specified in internal power tables.

This is the syntax for a one-dimensional table using the power group:

```plaintext
text
```

This is the syntax for a one-dimensional table using fall_power, and rise_power:

```plaintext
text
```

This is the syntax for a two-dimensional table using the power group:

```plaintext
text
```

This is the syntax for a two-dimensional table using the related_pin attribute and the fall_power and rise_power groups:

```plaintext
text
```
related_pin : "name | name_list" ;
fall_power (template name) {
  values ("float, ..., float");
}
rise_power (template name) {
  values ("float, ..., float");
}

This is the syntax for a three-dimensional table using the power group:

internal_power() {
  power (template name) {
    values ("float, ..., float");
  }
}

This is the syntax for a three-dimensional table using the related_pin attribute, power group, and the equal_or_opposite attribute:

internal_power() {
  related_pin : "name | name_list" ;
  power (template name) {
    values ("float, ..., float");
  }
  equal_or_opposite_output : "name | name_list" ;
}

equal_or_opposite_output Simple Attribute

The equal_or_opposite_output attribute designates optional output pin or pins whose capacitance is used to access a three-dimensional table in the internal_power group.

Syntax

equal_or_opposite_output : "name | name_list" ;

name | name_list

The name of the output pin or pins.

Note:
This pin (or pins) has to be functionally equal to or opposite of the pin named in this pin group.

Example

equal_or_opposite_output : "Q" ;

Note:
The output capacitance of this pin (or pins) is used as the total output2_net_capacitance variable in the internal power lookup table.
falling_together_group Simple Attribute

The falling_together_group attribute identifies the list of two or more input or output pins that share logic and are falling together during the same time period. This time period is set with the switching_interval attribute; see “switching_interval Simple Attribute” on page 3-75 for details.

Together, the falling_together_group and switching_interval attribute settings determine the level of power consumption.

Syntax
falling_together_group : "list of pins" ;

list of pins
The names of the input or output pins that share logic and are falling during the same time period.

Example
cell (foo) {
  pin (A) {
    internal_power () {
      falling_together_group : "B C D" ;
      rising_together_group : "E F G" ;
      switching_interval : 10.0 ;
      rise_power () {
        ...
      }
      fall_power () {
        ...
      }
    }
  }
}

power_level Simple Attribute

This optional attribute is used for multiple power supply modeling. In the internal_power group at the pin level, you can specify the power level used to characterize the lookup table.

Syntax
power_level : "name" ;

name
Name of the power rail defined in the power supply group.

Example
power_level : "VDD1" ;
related_pin Simple Attribute

This attribute is used only in three-dimensional tables. It associates the internal_power group with a specific input or output pin. If related_pin is an output pin, it must be functionally equal to or opposite of the pin in that pin group.

If related_pin is an input pin, the pin’s input transition time is used as a variable in the internal power lookup table.

If related_pin is an output pin, the pin’s capacitance is used as a variable in the internal power lookup table.

Syntax

related_pin : "name | name_list" ;

name | name_list

The name of the input or output pin or pins.

Example

related_pin : "A B" ;

The pin or pins in the related_pin attribute denote the path dependency for the internal_power group. A particular internal_power group is accessed if the input pin or pins named in the related_pin attribute cause the corresponding output pin named in the pin group to toggle. All functionally related pins must be specified in a related_pin attribute if two-dimensional tables are being used.

related_pg_pin Simple Attribute

Use this optional attribute to associate a power and ground pin with leakage power and internal power tables. The leakage power and internal energy tables can be omitted when the voltage of a primary_power or backup_ground pg_pin is at reference voltage zero, since the value of the corresponding leakage power and internal energy tables are always zero.

In the absence of a related_pg_pin attribute, the internal_power or leakage_power specifications apply to the whole cell (cell-specific power specification).

Cell-specific and pg_pin-specific power specifications cannot be mixed; that is, when one internal_power group has the related_pg_pin attribute, all the internal_power groups must have the related_pg_pin attribute.

Syntax

related_pg_pin : pg_pin;

where pg_pin is the name of the related PG pin.
Example
related_pg_pin : G2 ;

rising_together_group Simple Attribute
The rising_together_group attribute identifies the list of two or more input or output pins that share logic and are rising during the same time period. This time period is defined with the switching_interval attribute; see "switching_interval Simple Attribute" on page 3-75 for details.

Together, the rising_together_group attribute and switching_interval attribute settings determine the level of power consumption.

Syntax
rising_together_group : "list of pins" ;

list of pins
The names of the input or output pins that share logic and are rising during the same time period.

Example
cell (new_cell) {
  pin (A) {
    internal_power () {
      falling_together_group : "B C D" ;
      rising_together_group : "E F G" ;
      switching_interval : 10.0 ;
      rise_power () {
        ...
      }
      fall_power () {
        ...
      }
    }
  }
}

switching_interval Simple Attribute
The switching_interval attribute defines the time interval during which two or more pins that share logic are falling, rising, or switching (either falling or rising) during the same time period.

This attribute is set together with the falling_together_group, rising_together_group, or switching_together_group attribute. Together with one of these attributes, the switching_interval attribute defines a level of power consumption.
For details about the attributes that are set together with the `switching_interval` attribute, see “falling_together_group Simple Attribute” on page 3-73, “rising_together_group Simple Attribute” on page 3-75, and “switching_together_group Simple Attribute” on page 3-76.

**Syntax**

```
switching_interval : valuefloat ;
```

**value**

A floating-point number that represents the time interval during which two or more pins that share logic are transitioning together.

**Example**

```
pin (Z) {
  direction : output;
  internal_power () {
    switching_together_group : "A B";
    /*if pins A, B, and Z switch*/ ;
    switching_interval : 5.0;
    /* switching within 5 time units */;
    power () {
      ...
    }
  }
}
```

**switching_together_group Simple Attribute**

The `switching_together_group` attribute identifies a list of two or more input or output pins that share logic, are either falling or rising during the same time period, and are not affecting the power consumption.

The time period is defined with the `switching_interval` attribute. See “switching_interval Simple Attribute” on page 3-75 for details.

**Syntax**

```
switching_together_group : "list of pins" ;
```

**list of pins**

The names of the input or output pins that share logic, are either falling or rising during the same time period, and are not affecting power consumption.

**when Simple Attribute**

The `when` attribute specifies the state-dependent condition that determines whether this power table is accessed.
You can use the *when* attribute to define one-, two-, or three-dimensional tables in the *internal_power* group. You can also use the *when* attribute in the *power*, *fall_power*, and *rise_power* groups.

**Note:**
If you want to use the same Boolean expression for multiple *when* statements in an *internal_power* group, you must specify a different power rail for each *internal_power* group.

**Syntax**

```
when : "Boolean expression" ;
```

*Boolean expression*

The name or names of the input and output pins with corresponding Boolean operators.

*Table 3-4* lists the Boolean operators valid in a *when* statement.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>invert previous expression</td>
</tr>
<tr>
<td>!</td>
<td>invert following expression</td>
</tr>
<tr>
<td>^</td>
<td>logical XOR</td>
</tr>
<tr>
<td>*</td>
<td>logical AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td>space</td>
<td>logical AND</td>
</tr>
<tr>
<td>+</td>
<td>logical OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td>0</td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>

The order of precedence of the operators is left to right, with inversion performed first, then XOR, then AND, then OR.

**Example**

```
when : "A B" ;
```
mode Complex Attribute

The `mode` attribute specifies the current mode of operation of the cell. Use this attribute in the `internal_power` group to define the internal power in the specified mode.

**Syntax**

```
mode (mode_name, mode_value) ;
```

**Example**

```
mode (rw, read) ;
```

fall_power Group

The `fall_power` group defines the power associated with a fall transition on a pin. You specify a `fall_power` group in an `internal_power` group in a `pin` group, as shown here.

```
cell (name_string) {
    pin (name_string) {
        internal_power () {
            fall_power (template name) {
                ... fall power description ...
            }
        }
    }
}
```

**Complex Attributes**

```
index_1 ("float, ..., float") ; /* lookup table */
index_2 ("float, ..., float") ; /* lookup table */
index_3 ("float, ..., float") ; /* lookup table */
values ("float, ..., float") ; /* lookup table */
```

**float**

Floating-point numbers that identify the amount of energy per fall transition the cell consumes internally.

You convert the `values` attribute to power consumption by multiplying the unit by the factor transition or per-unit time, as follows:

- nindex_1 floating-point numbers if the table is one-dimensional
- nindex_1 * nindex_2 floating-point numbers if the table is two-dimensional
- nindex_1 * nindex_2 * nindex_3 floating-point numbers if the table is three-dimensional

nindex_1, nindex_2, and nindex_3 are the size of index_1, index_2, and index_3 in this group or in the `power_lut_template` group it inherits. Quotation marks (" " ) enclose a group. Each group represents a row in the table.
This power is accessed when the pin has a fall transition. If you have a fall_power group, you must have a rise_power group.

Example 3-3 on page 3-81 shows cells that contain internal power information in the pin group.

**power Group**

Use the power group to define power when the rise power equals the fall power for a particular pin. Specify a power group within an internal_power group in a pin group at the cell level.

**Syntax**

```plaintext
library (name) {
  cell (name) {
    pin (name) {
      internal_power () {
        power (template name) {
          ... power template description ...
        }
      }
    }
  }
}
```

**Complex Attributes**

- `index_1 ("float, ..., float")`
- `index_2 ("float, ..., float")`
- `index_3 ("float, ..., float")`
- `values ("float, ..., float")`

**float**

Floating-point numbers that identify the amount of energy per transition, either rise or fall, the cell consumes internally.

You convert the values attribute to power consumption by multiplying the unit by the factor transition or per-unit time, as follows:

- nindex_1 floating-point numbers if the table is one-dimensional
- nindex_1 x nindex_2 floating-point numbers if the table is two-dimensional
- nindex_1 x nindex_2 x nindex_3 floating-point numbers if the table is three-dimensional

nindex_1, nindex_2, and nindex_3 are the size of index_1, index_2, and index_3 in this group or in the power_lut_template group it inherits. Quotation marks (" ") enclose a group. Each group represents a row in the table.
This power is accessed when the pin has a rise transition or fall transition. The values in the table specify the average power per transition.

Example 3-3 on page 3-81 shows cells that contain power information in the internal_power group in a pin group.

rise_power Group

The rise_power group defines the power associated with a rise transition on a pin. Specify the rise_power group in an internal_power group in a pin group.

Syntax

cell (name) {
    pin (name) {
        internal_power () {
            rise_power (template name) {
                ... rise power description ...
            }
        }
    }
}

Complex Attributes

index_1 ("float, ..., float") ;
index_2 ("float, ..., float") ;
index_3 ("float, ..., float") ;
values ("float, ..., float") ;

float

Floating-point numbers that identify the amount of energy per rise transition the cell consumes internally.

You convert the values attribute to power consumption by multiplying the unit by the factor transition or per-unit time, as follows:

- nindex_1 floating-point numbers if the table is one-dimensional
- nindex_1 x nindex_2 floating-point numbers if the table is two-dimensional
- nindex_1 x nindex_2 x nindex_3 floating-point numbers if the table is three-dimensional

The nindex_1, nindex_2, and nindex_3 number are the size of the index_1, index_2, and index_3 attribute values in this group or in the power_lut_template group it inherits. Quotation marks (" ") enclose a group. Each group represents a row in the table.

This power is accessed when the pin has a rise transition.

Example 3-3 shows cells that contain internal power information in the pin group.
Example 3-3  A Library With Internal Power

library(internal_power_example) {
  ...
  power_lut_template(output_by_cap1_cap2_and_trans) {
    variable_1 : total_output1_net_capacitance;
    variable_2 : equal_or_opposite_output_net_capacitance;
    variable_3 : input_transition_time;
    index_1 ("0.0, 5.0, 20.0");
    index_2 ("0.0, 5.0, 20.0");
    index_3 ("0.0, 1.0, 2.0");
  }
  power_lut_template(output_by_cap_and_trans) {
    variable_1 : total_output_net_capacitance;
    variable_2 : input_transition_time;
    index_1 ("0.0, 5.0, 20.0");
    index_2 ("0.0, 1.0, 2.0");
  }
  power_lut_template(input_by_trans) {
    variable_1 : input_transition_time;
    index_1 ("0.0, 1.0, 2.0");
  }
  ...
  cell(AN2) {
    pin(Z) {
      direction : output;
      internal_power() {
        power (output_by_cap1_cap2_and_trans) {
          values ("2.2, 3.7, 4.3", "1.7, 2.1, 3.5", "1.0, 1.5, 2.8");
        }
        related_pin : "A B";
      }
      ...
    }
    pin(A) {
      direction : input;
      ...
    }
    pin(B) {
      direction : input;
      ...
    }
  }
  cell(FLOP1) {
    pin(CP) {
      direction : input;
      internal_power() {
        power (input_by_trans) {
          values ("1.5, 2.5, 4.7");
        }
      }
    }
    pin(D) {
      direction : input;
      ...
    }
    pin(S) {
      ...
max_cap Group

The max_cap group defines the frequency-based maximum capacitance information for the output and inout pins.

Syntax

library (name) {
  cell (name) {
    pin (name) {
      direction : input;
      ...
    }
    pin(R) {
      direction : input;
      ...
    }
    pin(Q) {
      direction : output;
      internal_power() {
        power (output_by_cap1_cap2_and_trans) {
          values ("2.2, 3.7, 4.3", "1.7, 2.1, 3.5", "1.0, 1.5, 2.8", \\
          "2.1, 3.6, 4.2", "1.6, 2.0, 3.4", "0.9, 1.5, 2.7", \\
          "2.0, 3.5, 4.1", "1.5, 1.9, 3.3", "0.8, 1.4, 2.6");
        }
        when : "S' + R'";
        equal_or_opposite_output : "QN";
        related_pin : "CP";
      }
    }
    internal_power() {
      power (output_by_cap_and_trans) {
        values ("1.8, 3.4, 4.0", "1.5, 1.9, 3.3", "0.8, 1.3, 2.5");
      }
      related_pin : "S R";
    }
    ...
    }
    pin(QN) {
      direction : output;
      internal_power() {
        rise_power (output_by_cap_and_trans) {
          values ("0.5, 0.9, 1.3", "0.3, 0.7, 1.1", "0.2, 0.5, 0.9");
        }
        fall_power (output_by_cap_and_trans) {
          values ("0.1, 0.7, 0.9", "-0.1, 0.2, 0.4", "-0.2, 0.2, 0.3");
        }
        related_pin : "S R";
      }
      ...
      }
    }
  }
  ...
}
max_cap (template name) {
  ... capacitance description ...
}
}
}

template_name

A value representing the name of a maxcap_lut_template group. You need to specify or remove an attribute from the group according to the template. The supported attributes for the template are frequency and input_transition_time. Because input_transition_time is the second index attribute, a related_pin attribute is required to inform the transition of the corresponding input pin. The template can be one-dimensional or two-dimensional. A one-dimensional template does not allow the related_pin attribute. A two-dimensional template requires the related_pin attribute.

Example
max_cap ( ) {
  maxcap_lut_template(maxcap_table) {
    variable_1 : frequency ;
    variable_2 : input_transition_time ;
    index_1("0.01, 0.1, 1.0");
    index_2("0, 0.5, 1.5, 2.0");
  }
  ...
  pin(Y) {
    direction : output   ;
    max_fanout : 7 ;
    function : "(A+B)'";
    max_cap(maxcap_table) {
      values("1.1, 1.2, 1.3, 1.4", "1.2, 1.3, 1.4, 1.5", "1.3, 1.4, 1.5, 1.6");
      related_pin : A ;
    }
  }
  ...
}

max_trans Group

Use the max_trans group to describe the maximum transition information for output and inout pins.

Syntax
library (name) {
  cell (name) {
    pin (name) {
      max_trans (template_nameid) {
        ... transition description ...
      }
    }
  }
}
template_name
A value representing the name of a maxtrans_lut_template group.

Example
def max_trans() {
    ...
}

min_pulse_width Group
In a pin, bus, or bundle group, the min_pulse_width group models the enabling conditional minimum pulse width check. In the case of a pin, the timing check is performed on the pin itself, so the related pin must be the same.

Syntax

def pin() {
    ...
    def min_pulse_width() {
        constraint_high : value;
        constraint_low : value;
        when : "Boolean expression";
        /* enabling condition */
        sdf_cond : "Boolean expression";
        /* in SDF syntax */
    }
}

Example
def pin(A) {
    ...
    def min_pulse_width() {
        constraint_high : 3.0;
        constraint_low : 3.5;
        when : "SE";
        sdf_cond : "SE == 1'B1";
    }
}

For an example that shows how to specify a lookup table with the timing_type attribute and min_pulse_width and minimum_period values, see Example 3-4 on page 3-106.
Simple Attributes

constraint_high
constraint_low
when
sdf_cond

Complex Attributes

mode

constraint_high Simple Attribute

The constraint_high attribute defines the minimum length of time the pin must remain at logic 1. You define a value for either constraint_high, constraint_low, or both in the min_pulse_width group.

Syntax
constraint_high : valuefloat ;

value
A nonnegative number.

Example
constraint_high : 3.0 ; /* min_pulse_width_high */

when Simple Attribute

The when attribute defines the enabling condition for the check in Synopsys logic expression format.

Syntax
when : "Boolean expression" ;

For a list of Boolean operators, see Table 3-4 on page 3-77.

Example
when : "SE" ;

constraint_low Simple Attribute

The constraint_low attribute defines the minimum length of time the pin must remain at logic 0. You define a value for either constraint_low, constraint_high, or both in the min_pulse_width group.
Syntax
constraint_low : value\_float;

value
  A nonnegative number.

Example
constraint_low : 3.5; /* min\_pulse\_width\_low */

sdf\_cond Simple Attribute
The sdf\_cond attribute defines the enabling condition for the check in Open Verilog International (OVI) Standard Delay Format (SDF) 2.1 syntax.

Syntax
sdf\_cond : "Boolean expression";

Boolean expression
  An SDF condition expression.

Example
sdf\_cond : "SE == 1'\!B1";

mode Complex Attribute
The mode complex attribute uses the mode\_name to reference a mode\_value group defined in the cell.

In PG pin power states modeling, when you specify the pg\_setting attribute in the referenced mode\_value group, the min\_pulse\_width group where the mode is referenced becomes power-state aware.

Syntax
pin (pin\_name) {  
  min\_pulse\_width(min\_pulse\_width\_name) {  
    mode(mode\_def\_name, mode\_name);
  }
}

Example
mode (power\_state, active);
minimum_period Group

In a pin, bus, or bundle group, the minimum_period group models the enabling conditional minimum period check. In the case of a pin, the check is performed on the pin itself, so the related pin must be the same.

If the pin group contains a minimum_period group and a min_period attribute, the min_period attribute is ignored.

Syntax

minimum_period() {
    constraint : value ;
    when : "Boolean expression" ;
    sdf_cond : "Boolean expression" ;
}

For an example that shows how to specify a lookup table with the timing_type attribute and min_pulse_width and minimum_period values, see Example 3-4 on page 3-106.

Simple Attributes

constraint
when
sdf_cond

Complex Attributes

mode

constraint Simple Attribute

This required attribute defines the minimum clock period for the pin.

Syntax

constraint : value_float ;

value
    A nonnegative number.

Example

constraint : 9.5 ;
when Simple Attribute

This required attribute defines the enabling condition for the check in logic expression format.

Syntax
when : "Boolean expression" ;

For a list of Boolean operators, see Table 3-4 on page 3-77.

Example
when : "SE" ;

sdf_cond Simple Attribute

This required attribute defines the enabling condition for the check in OVI SDF 2.1 syntax.

Syntax
sdf_cond : "Boolean expression" ;

Boolean expression
An SDF condition expression.

Example
sdf_cond : "SE == 1'b1" ;

mode Complex Attribute

The mode attribute uses the mode_name to reference a mode_value group defined in the cell. When you specify the pg_setting attribute in the referenced mode_value group, the minimum_period group where the mode is referenced becomes power-state aware.

Syntax
pin (pin_name) {
  minimum_period(minimum_period_name){
    mode(mode_def_name, mode_name);
  }
}

Example
mode (power_state, active);
**receiver_capacitance Group**

Use the `receiver_capacitance` group to specify capacitance values for composite current source (CCS) receiver modeling at the pin level.

**Syntax**
```
library (namestring) {
  cell (namestring) {
    pin (namestring) {
      receiver_capacitance (){
        ... description ...
      }
    }
  }
}
```

**Groups**

For two-segment receiver capacitance model

- `receiver_capacitance1_fall`
- `receiver_capacitance1_rise`
- `receiver_capacitance2_fall`
- `receiver_capacitance2_rise`

For multisegment receiver capacitance model

- `receiver_capacitance_fall`
- `receiver_capacitance_rise`

**receiver_capacitance1_fall Group**

In the two-segment receiver capacitance model, you can define the `receiver_capacitance1_fall` group at the pin level and the timing level. Define the `receiver_capacitance1_fall` group at the pin level to reference a lookup table template. For information about using the group at the timing level, see “receiver_capacitance1_fall Group” on page 3-138.

**Syntax**
```
receiver_capacitance1_fall (lu_template_name) {
  lu_template_name
    The name of a template.
}
```

**Complex Attribute**

*values*
Example
receiver_capacitance () {
  receiver_capacitance1_rise (LTT1) {
    values (0.0, 0.0, 0.0, 0.0) ;
  }
  receiver_capacitance1_fall (LTT1) {
    ...
  }
  ...
}

receiver_capacitance1_rise Group
For information about using the receiver_capacitance1_rise group, see the description of the “receiver_capacitance1_fall Group.”

receiver_capacitance2_fall Group
For information about using the receiver_capacitance2_fall group, see the description of “receiver_capacitance1_fall Group.”

receiver_capacitance2_rise Group
For information about using the receiver_capacitance2_rise group, see the description of the “receiver_capacitance1_fall Group.”

receiver_capacitance_fall Group
In the multi-segment receiver capacitance model, you can define the receiver_capacitance_fall group in the pin-level receiver_capacitance group and in the timing group. Define the receiver_capacitance_fall group to reference a lookup table template. For information about using the group at the timing level, see “receiver_capacitance_fall Group” on page 3-138.

Syntax
receiver_capacitance_fall (lu_template_name) {}

lu_template_name
  The name of a template.

Simple Attribute
segment

segment Simple Attribute
  The segment attribute specifies the segment that the receiver_capacitance_rise or the receiver_capacitance_fall group represents. The values vary from 1 to N.
Complex Attribute

values

values Complex Attribute

Use this attribute to specify the receiver capacitance values in voltage rise or fall segments.

Example

receiver_capacitance () {
  receiver_capacitance_rise (LTT1) {
    segment: 4;
    values (0.0, 0.0, 0.0, 0.0);
  }
  receiver_capacitance_fall (LTT1) {
    ...
  }
  ...
}

receiver_capacitance_rise Group

For information about using the receiver_capacitance_rise group, see the description of the "receiver_capacitance_fall Group."

Complex Attribute

mode

mode Attribute

The mode attribute specifies the current mode of the cell. If the mode attribute is specified, the mode_name and mode_value must be predefined in the mode_definition group at the cell level.

Simple Attributes

when

char_when

when Attribute

The when attribute specifies the Boolean condition for the input state of the receiver capacitance timing arc.

char_when Attribute

The char_when attribute specifies the input state at which the receiver capacitance timing arc was characterized.
You must specify the `char_when` attribute in a pin-level `receiver_capacity` group that represents a default condition timing arc, that is, does not have the conditional `when` attribute.

---

**timing Group in a pin Group**

A timing group is defined within a pin group. The groups and attributes of the timing group are listed in the alphabetical order.

To identify timing arcs, you can name a timing group.

**Syntax**

```plaintext
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing (name_string){
        ... timing description ...
      }
    }
  }
}
```
Simple Attributes

clock_gating_flag : true|false ;
default_timing : true|false ;
fpga_arc_condition : "Boolean expression" ;
interdependence_id : integer ;
output_signal_level_high : float ;
output_signal_level_low : float ;
related_output_pin : name ;
related_pin : " name1 [name2 name3 ... ] " ;
sdf_cond : "SDF expression" ;
sdf_cond_end : "SDF expression" ;
sdf_cond_start : "SDF expression" ;
sdf_edges : SDF edge type ;
timing_sense : positive_unate| negative_unate| non_unate ;
timing_type : combinational | combinational_rise |
        combinational_fall | three_state_disable |
        three_state_disable_rise | three_state_disable_fall |
        three_state_enable | three_state_enable_rise |
        three_state_enable_fall | rising_edge | falling_edge |
        preset | clear | hold_rising | hold_falling |
        setup_rising | setup_falling | recovery_rising |
        recovery_falling | skew_rising | skew_falling |
        removal_rising | removal_falling | min_pulse_width |
        min_clock_tree_path | max_clock_tree_path |
        non_seq_setup_rising | non_seq_setup_falling |
        non_seq_scale_rising | non_seq_scale_falling |
        nochange_high_high | nochange_high_low |
        nochange_low_high | nochange_low_low ;
when : "Boolean expression" ;
when_end : "Boolean expression" ;
when_start : "Boolean expression" ;
Complex Attributes

`active_input_ccb(string, string) ;`
`active_output_ccb(string) ;`
`function`
`mode`
`pin_name_map`
`propagating_ccb(string, string) ;`
`reference_input`
`wave_rise`
`wave_fall`
`wave_rise_time_interval`
`wave_fall_time_interval`

Group Statements

`cell_degradation () { }`
`cell_fall () { }`
`cell_rise () { }`
`char_config () { }`
`fall_constraint () { }`
`fall_propagation () { }`
`fall_transition () { }`
`ocv_sigma_cell_fall () { }`
`ocv_sigma_cell_rise () { }`
`ocv_sigma_fall_constraint () {}`
`ocv_sigma_fall_transition () {}`
`ocv_sigma_rise_constraint () {}`
`ocv_sigma_rise_transition () {}`
`ocv_sigma_retaining_fall () {}`
`ocv_sigma_retaining_rise () {}`
`ocv_sigma_retain_fall_slew () {}`
`ocv_sigma_retain_rise_slew () {}`
`ocv_mean_shift_cell_rise(){}
`ocv_mean_shift_cell_fall(){}
`ocv_mean_shift_fall_transition(){}
`ocv_mean_shift_fall_constraint(){}
`ocv_mean_shift_rise_constraint(){}
`ocv_mean_shift_rise_transition(){}
`ocv_mean_shift_retaining_rise(){}
`ocv_mean_shift_retaining_fall(){}
`ocv_mean_shift_retain_rise_slew(){}
`ocv_mean_shift_retain_fall_slew(){}
`ocv_mean_shift_rise(){}
`ocv_mean_shift_fall(){}
`ocv_mean_shift_rise_transition(){}
`ocv_mean_shift_fall_transition(){}
`ocv_mean_shift_rise_constraint(){}
`ocv_mean_shift_fall_constraint(){}
`ocv_mean_shift(cell, string) ;
`ocv_mean_shift_fall(cell, string) ;
`ocv_mean_shift_fall_transition(cell, string) ;
`ocv_mean_shift_fall_constraint(cell, string) ;
`ocv_mean_shift_rise(cell, string) ;
`ocv_mean-shift_fall(cell, string) ;
`ocv_mean_shift_rise_transition(cell, string) ;
`ocv_mean_shift_rise_constraint(cell, string) ;
`ocv_mean_shift_retaining_fall(cell, string) ;
`ocv_mean_shift_retaining_rise(cell, string) ;
`ocv_mean_shift_retain_fall_slew(cell, string) ;
`ocv_mean_shift_retain_rise_slew(cell, string) ;
`ocv_mean_shift_rise(cell, string) ;
`ocv_mean_shift_fall(cell, string) ;
`ocv_mean_shift_rise_transition(cell, string) ;
`ocv_mean_shift_fall_transition(cell, string) ;
`ocv_mean_shift_rise_constraint(cell, string) ;
`ocv_mean_shift_fall_constraint(cell, string) ;

Chapter 3: pin Group Description and Syntax

Group Statements

3-94
clock_gating_flag Simple Attribute

Use this attribute to indicate that a constraint arc is for a clock gating relation between the data and clock pin, instead of a constraint found in standard sequential devices, such as registers and latches.

Syntax

clock_gating_flag : Boolean ;

Boolean

Valid values are true and false. The value true is applicable only when the value of the timing_type attribute is setup, hold, or nochange. When not defined for a timing arc, the value false is assumed, indicating the timing arc is part of a standard sequential device.
Example

clock_gating_flag : true ;

**default_timing Simple Attribute**

The `default_timing` attribute allows you to specify one timing arc as the default in the case of multiple timing arcs with `when` statements.

**Syntax**

default_timing : Boolean expression ;

**Example**

default_timing : true ;

**fpga_arc_condition Simple Attribute**

The `fpga_arc_condition` attribute specifies a Boolean condition that enables a timing arc.

**Syntax**

fpga_arc_condition : conditionBoolean ;

**condition**

Specifies a Boolean condition. Valid values are true and false.

**Example**

fpga_arc_condition : "!A" ;

**interdependence_id Simple Attribute**

Use pairs of `interdependence_id` attributes to identify interdependent pairs of setup and hold constraint tables. Interdependence data is supported in conditional constraint checking; the value of the `interdependence_id` attribute increases independently for each condition. Interdependence data can be specified in pin, bus, and bundle groups.

**Syntax**

interdependence_id : "name_enum" ;

**name**

Valid values are 1, 2, 3, and so on.

**Examples**

timing()
    related_pin : CLK ;
    timing_type: setup_rising ;
interdependence_id : 1 ;
...
timing()
  related_pin : CLK ;
timing_type: setup_rising ;
timing_interdependence_id = 2 ;
...
pin (D_IN) {
  ...
  /* original nonconditional setup/hold constraints */
  setup/hold constraints
  /* new interdependence data for nonconditional constraint
  checking */
  setup/hold, timing_interdependence_id = 1
  setup/hold, timing_interdependence_id = 2
  setup/hold, timing_interdependence_id = 3

  /* original setup/hold constraints for conditional
  <condition_a> */
  setup/hold when <condition_a> *
  /* new interdependence data for <condition_a> constraint
  checking */
  setup/hold when <condition_a>, timing_interdependence_id = 1
  setup/hold when <condition_a>, timing_interdependence_id = 2
  setup/hold when <condition_a>, timing_interdependence_id = 3

  /* original setup/hold constraints for conditional
  <condition_b> */
  setup/hold when <condition_b> *
  /* new interdependence data for <condition_b> constraint
  checking */
  setup/hold when <condition_b>, timing_interdependence_id = 1
  setup/hold when <condition_b>, timing_interdependence_id = 2
  setup/hold when <condition_b>, timing_interdependence_id = 3
}

Guidelines:

• To prevent potential backward-compatibility issues, interdependence data cannot be the
  first timing arc in the pin group.

• The interdependence_id attribute only supports the following timing types:
  setup_rising, setup_falling, hold_rising, and hold_falling. If you set this
  attribute on other timing types, an error is reported.

• You must specify setup and hold interdependence data in pairs; otherwise an error is
  reported. If you define one setup_rising timing arc with interdependence_id: 1; on
  a pin, you must also define a hold_rising timing arc with interdependence_id: 1;
for that pin. The `interdependence_id` can be a random integer, but it must be found in a pair of timing arcs. These timing types are considered as pairs: \texttt{setup_rising} with \texttt{hold_rising} and \texttt{setup_falling} with \texttt{hold_falling}.

- For each set of conditional constraints (nonconditional categorized as a special condition), a timing arc with a specific `interdependence_id` must be unique in a pin group.

- For each set of conditional constraints, the `interdependence_id` must start from 1, and if there is multiple interdependence data defined, the values for the `interdependence_id` must be in consecutive order. That is, 1, 2, 3 is allowed, but 1, 2, 4 is not.

**output_signal_level_high Simple Attribute**

The optional `output_signal_level_high` and `output_signal_level_low` attributes specify the actual output voltages of an output pin after a transition through a timing arc.

The `output_signal_level_low` attribute specifies the minimum voltage after a falling transition to state 0.

The `output_signal_level_high` attribute specifies the maximum voltage value after a rising transition to state 1.

Define the `output_signal_level_low` and `output_signal_level_high` attributes in the timing group when the following occur together:

- The cell output exhibits a partial voltage swing (and not a rail-to-rail swing).
- The voltages are different in different timing arcs.

**Syntax**

\begin{verbatim}
output_signal_level_high : float ;
output_signal_level_low : float ;
\end{verbatim}

**Example**

\begin{verbatim}
output_signal_level_high : 0.75;
output_signal_level_low : 0.15;
\end{verbatim}

**output_signal_level_low Simple Attribute**

See “output_signal_level_high Simple Attribute” on page 3-98.
related_output_pin Simple Attribute

The `related_output_pin` attribute specifies the output or inout pin used to describe a load-dependent constraint. This is an attribute in the `timing` group of the output or inout pin. The pin defined must be a pin in the same cell, and its direction must be either output or inout.

Syntax

```plaintext
related_output_pin : name ;
```

Example

```plaintext
related_output_pin : Z ;
```

related_pin Simple Attribute

The `related_pin` attribute defines the pin or pins representing the start point of the timing arc. It is required in all `timing` groups.

Syntax

```plaintext
related_pin : "name1 [name2 name3 ... ]" ;
```

In a cell with input pin A and output pin B, define A and its relationship to B in the `related_pin` attribute statement in the `timing` group that describes pin B.

Example

```plaintext
pin (B) {  
    direction : output ;
    function : "A'";
    timing () { 
        related_pin : "A" ;
        ... timing information ... 
    } 
}
```

The `related_pin` attribute statement can also serve as a shortcut for two identical timing arcs for a cell. For example, in a 2-input NAND gate with identical delays from both input pins to the output pin, you only need to define only one timing arc with two related pins.

Example

```plaintext
pin (Z) {  
    direction : output; 
    function : "(A * B)'" ;
    timing () { 
        related_pin : "A B" ;
        ... timing information ... 
    } 
}
```
When a bus name appears in a `related_pin` attribute, the bus members or range of members is distributed across all members of the parent bus. The width of the bus or the range must be the same as the width of the parent bus.

Pin names used in a `related_pin` statement can start with a nonalphabetic character.

**Example**

```verbatim
related_pin : "A 1B 2C" ;
```

**Note:**

It is not necessary to use the escape character, \ (backslash), with nonalphabetic characters.

---

**sdf_cond Simple Attribute**

The `sdf_cond` attribute is defined in the state-dependent `timing` group to support SDF file generation and condition matching during back-annotation.

**Syntax**

```verbatim
sdf_cond : "SDF expression" ;
```

**SDF expression**

A string that represents a Boolean description of the state dependency of the delay. Use a Boolean description that conforms to the valid syntax defined in the OVI SDF, which is different from the Boolean expression. For a complete description of the valid syntax for these expressions, see the OVI specification for SDF, V1.0.

**Example**

```verbatim
sdf_cond : "b == 1'b1" ;
```

---

**sdf_cond_end Simple Attribute**

The `sdf_cond_end` attribute defines a timing-check condition specific to the end event in VHDL models. The expression must conform to OVI SDF 2.1 timing-check condition syntax.

**Syntax**

```verbatim
sdf_cond_end : "SDF expression" ;
```

**SDF expression**

An SDF expression containing names of input, output, inout, and internal pins.

**Example**

```verbatim
sdf_cond_end : "SIG_0 == 1'b1" ;
```
sdf_cond_start Simple Attribute

The sdf_cond_start attribute defines a timing-check condition specific to the start event in full-timing gate-level simulation (FTGS) models. The expression must conform to OVI SDF 2.1 timing-check condition syntax.

Syntax
sdf_cond_start : "SDF expression" ;

SDF expression
An SDF expression containing names of input, output, inout, and internal pins.

Example
sdf_cond_start : "SIG_2 == 1'b1" ;

sdf_edges Simple Attribute

The sdf_edges attribute defines the edge specification on both the start pin and the end pin. The default is noedge.

Syntax
sdf_edges : sdf_edge_type;

sdf_edge_type
The valide edge types are: noedge, start_edge, end_edge, or both_edges. The default is noedge.

Example
sdf_edges : both_edges;
sdf_edges : start_edge ; /* edge specification on starting pin */
sdf_edges : end_edge ; /* edge specification on end pin */

sensitization_master Simple Attribute

The sensitization_master attribute defines the sensitization group specific to the current timing group to generate stimulus for characterization. The attribute is optional when the sensitization master used for the timing arc is the same as that defined in the current cell. It is required when they are different. Any sensitization group name predefined in the current library is a valid attribute value.

Syntax
sensitization_master : sensitization_group_name;
sensitization_group_name
A string identifying the sensitization group name predefined in the current library.

Example
sensitization_master : sensi_2in_1out;

timing_sense Simple Attribute
The timing_sense attribute describes the way an input pin logically affects an output pin.

Syntax
timing_sense : positive_unate | negative_unate | non_unate ;

positive_unate
Combines incoming rise delays with local rise delays and compares incoming fall delays with local fall delays.

negative_unate
Combines incoming rise delays with local fall delays and compares incoming fall delays with local rise delays.

non_unate
Combines local delays with the worst-case incoming delay value. The non-unate timing sense represents a function whose output value change cannot be determined from the direction of the change in the input value.

The timing_sense is derived from the logic function of a pin. For example, the value derived for an AND gate is positive_unate, the value for a NAND gate is negative_unate, and the value for an XOR gate is non_unate.

A function is unate if a rising (or falling) change on a positive unate input variable causes the output function variable to rise (or fall) or not change. A rising (or falling) change on a negative unate input variable causes the output function variable to fall (or rise) or not change. For a nonunate variable, further state information is required to determine the effects of a particular state transition.

You can specify half-unate sequential timing arcs if the timing_type value is either rising_edge or falling_edge and the timing_sense value is either positive_unate or negative_unate.

• In the case of rising_edge and positive_unate values, only the cell_rise and rise_transition information is required.
• In the case of rising_edge and negative_unate values, only the cell_fall and fall_transition information is required.
• In the case of falling_edge and positive_unate values, only the cell_rise and rise_transition information is required.

• In the case of falling_edge and negative_unate values, only the cell_fall and fall_transition information is required.

Do not define the timing_sense value of a pin, except when you need to override the derived value or when you are characterizing a noncombinational gate such as a three-state component. For example, you might want to define the timing sense manually when you model multiple paths between an input pin and an output pin, such as in an XOR gate.

It is possible that one path is positive unate while another is negative unate. In this case, the first timing arc is given a positive_unate designation and the second is given a negative_unate designation.

Timing arcs with a timing type of clear or preset require a timing_sense attribute.

If related_pin is an output pin, you must define a timing_sense attribute for that pin.

timing_type Simple Attribute

The timing_type attribute distinguishes between combinational and sequential cells by defining the type of timing arc. If this attribute is not assigned, the cell is considered combinational.

Syntax

timing_type : combinational | combinational_rise | combinational_fall | three_state_disable | three_state_disable_rise | three_state_disable_fall | three_state_enable | three_state_enable_rise | three_state_enable_fall | rising_edge | falling_edge | preset | clear | hold_rising | hold_falling | setup_rising | setup_falling | recovery_rising | recovery_falling | skew_rising | skew_falling | removal_rising | removal_falling | min_pulse_width | minimum_period | max_clock_tree_path | min_clock_tree_path | non_seq_setup_rising | non_seq_setup_falling | non_seq_hold_rising | non_seq_hold_falling | nochange_high_high | nochange_high_low | nochange_low_high | nochange_low_low ;
Combinational Timing Arcs

The timing type and timing sense define the signal propagation pattern. The default timing type is combinational. Table 3-5 shows the timing type and timing sense values for combinational timing arcs.

Table 3-5  timing_type and timing_sense Values for Combinational Timing Arcs

<table>
<thead>
<tr>
<th>Timing type</th>
<th>Timing sense</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive_Unate</td>
</tr>
<tr>
<td>combinational</td>
<td>R-&gt;R,F-&gt;F</td>
</tr>
<tr>
<td>combinational_rise</td>
<td>R-&gt;R</td>
</tr>
<tr>
<td>combinational_fall</td>
<td>F-&gt;F</td>
</tr>
<tr>
<td>three_state_disable</td>
<td>R-&gt;[0Z,1Z]</td>
</tr>
<tr>
<td>three_state_enable</td>
<td>R-&gt;[Z0,Z1]</td>
</tr>
<tr>
<td>three_state_disable_rise</td>
<td>R-&gt;0Z</td>
</tr>
<tr>
<td>three_state_disable_fall</td>
<td>R-&gt;1Z</td>
</tr>
<tr>
<td>three_state_enable_rise</td>
<td>R-&gt;Z1</td>
</tr>
<tr>
<td>three_state_enable_fall</td>
<td>R-&gt;Z0</td>
</tr>
</tbody>
</table>

Sequential Timing Arcs

rising_edge

Identifies a timing arc whose output pin is sensitive to a rising signal at the input pin.

falling_edge

Identifies a timing arc whose output pin is sensitive to a falling signal at the input pin.

preset

Preset arcs affect only the rise arrival time of the arc's endpoint pin. A preset arc implies that you are asserting a logic 1 on the output pin when the designated related_pin is asserted.
clear

Clear arcs affect only the fall arrival time of the arc’s endpoint pin. A clear arc implies that you are asserting a logic 0 on the output pin when the designated related_pin is asserted.

hold_rising

Designates the rising edge of the related pin for the hold check.

hold_falling

Designates the falling edge of the related pin for the hold check.

setup_rising

Designates the rising edge of the related pin for the setup check on clocked elements.

setup_falling

Designates the falling edge of the related pin for the setup check on clocked elements.

recovery_rising

Uses the rising edge of the related pin for the recovery time check. The clock is rising-edge-triggered.

recovery_falling

Uses the falling edge of the related pin for the recovery time check. The clock is falling-edge-triggered.

skew_rising

The timing constraint interval is measured from the rising edge of the reference pin (specified in related_pin) to a transition edge of the parent pin of the timing group. The intrinsic_rise value is the maximum skew time between the reference pin rising and the parent pin rising. The intrinsic_fall value is the maximum skew time between the reference pin falling and the parent pin falling.

skew_falling

The timing constraint interval is measured from the falling edge of the reference pin (specified in related_pin) to a transition edge of the parent pin of the timing group. The intrinsic_rise value is the maximum skew time between the reference pin falling and the parent pin rising. The intrinsic_fall value is the maximum skew time between the reference pin falling and the parent pin falling.

removal_rising

Used when the cell is a low-enable latch or a rising-edge-triggered flip-flop. For active-low asynchronous control signals, define the removal time with the
The `intrinsic_rise` attribute. For active-high asynchronous control signals, define the removal time with the `intrinsic_fall` attribute.

`removal_falling`

Used when the cell is a high-enable latch or a falling-edge-triggered flip-flop. For active-low asynchronous control signals, define the removal time with the `intrinsic_rise` attribute. For active-high asynchronous control signals, define the removal time with the `intrinsic_fall` attribute.

`min_pulse_width`

This value lets you specify the minimum pulse width for a clock pin. The timing check is performed on the pin itself, so the related pin should be the same. You need to specify both rise and fall constraints to calculate the high and low pulse widths.

`minimum_period`

This value lets you specify the minimum period for a clock pin. The timing check is performed on the pin itself, so the related pin should be the same. You need to specify both rise and fall constraints to calculate the minimum clock period. Rise constraint is characterization data when the clock waveform has a rising start edge. Fall constraint is characterization data when the start edge of a waveform is falling.

`max_clock_tree_path`

Used in timing groups under a clock pin. Defines the maximum clock tree path constraint.

`min_clock_tree_path`

Used in timing groups under a clock pin. Defines the minimum clock tree path constraint.

**Example**

Example 3-4 shows how to specify a lookup table with the `timing_type` attribute and `min_pulse_width` and `minimum_period` values. The `rise_constraint` group defines the rising pulse width constraint for `min_pulse_width`, and the `fall_constraint` group defines the falling pulse width constraint. For `minimum_period`, the `rise_constraint` group is used to model the period when the pulse is rising and the `fall_constraint` group is used to model the period when the pulse is falling. You can specify the `rise_constraint` group, the `fall_constraint` group, or both groups.

**Example 3-4  Example Library With timing_type Statements**

```c
library(example) {
  technology (cmos) ;
  delay_model : table_lookup ;

  /* 2-D table template */
```
lu_table_template ( mpw ) {
  variable_1 : constrained_pin_transition;
  /* You can replace the constrained_pin_transition value with related_pin_transition, but you cannot specify both values. */
  variable_2 : related_out_total_output_net_capacitance;
  index_1("1, 2, 3");
  index_2("1, 2, 3");
}
/* 1-D table template */
lu_table_template( f_ocap ) {
  variable_1 : total_output_net_capacitance;
  index_1( "0.0000, 1.0000" );
}

cell( test ) {
  area : 200.000000;
  dont_use : true;
  dont_touch : true;

  pin ( CK ) {
    direction : input;
    rise_capacitance : 0.00146468;
    fall_capacitance : 0.00145175;
    capacitance : 0.00146468;
    clock : true;

    timing ( mpw_constraint) {
      related_pin : "CK";
      timing_type : min_pulse_width;
      related_output_pin : "Z";

      fall_constraint ( mpw) {
        index_1("0.1, 0.2, 0.3");
        index_2("0.1, 0.2");
        values( "0.10 0.11", "0.12 0.13", "0.14 0.15");
      }

      rise_constraint ( mpw) {
        index_1("0.1, 0.2, 0.3");
        index_2("0.1, 0.2");
        values( "0.10 0.11", "0.12 0.13", "0.14 0.15");
      }

      timing ( mpw_constraint) {
        related_pin : "CK";
        timing_type : minimum_period;
        related_output_pin : "Z";
        fall_constraint ( mpw) {
        }
      }
    }
  }
}
Nonsequential Timing Arcs

In some nonsequential cells, the setup and hold timing constraints are specified on the data pin with a nonclock pin as the related pin. It requires the signal of a pin to be stable for a specified period of time before and after another pin of the same cell range state so that the cell can function as expected.

**non_seq_setup_rising**

Defines (with **non_seq_setup_falling**) the timing arcs used for setup checks between pins with nonsequential behavior. The related pin in a timing arc is used for the timing check.

**non_seq_setup_falling**

Defines (with **non_seq_setup_rising**) the timing arcs used for setup checks between pins with nonsequential behavior. The related pin in a timing arc is used for the timing check.

**non_seq_hold_rising**

Defines (with **non_seq_hold_falling**) the timing arcs used for hold checks between pins with nonsequential behavior. The related pin in a timing arc is used for the timing check.

**non_seq_hold_falling**

Defines (with **non_seq_hold_rising**) the timing arcs used for hold checks between pins with nonsequential behavior. The related pin in a timing arc is used for the timing check.
No-Change Timing Arcs

This feature models the timing requirement of latch devices with latch-enable signals. The four no-change timing types define the pulse waveforms of both the constrained signal and the related signal in standard CMOS and nonlinear CMOS delay models. The information is used in static timing verification during synthesis.

nochange_high_high (positive/positive)
Indicates a positive pulse on the constrained pin and a positive pulse on the related pin.

nochange_high_low (positive/negative)
Indicates a positive pulse on the constrained pin and a negative pulse on the related pin.

nochange_low_high (negative/positive)
Indicates a negative pulse on the constrained pin and a positive pulse on the related pin.

nochange_low_low (negative/negative)
Indicates a negative pulse on the constrained pin and a negative pulse on the related pin.

wave_rise_sampling_index and wave_fall_sampling_index Attributes

The wave_rise_sampling_index and wave_fall_sampling_index simple attributes override the default behavior of the wave_rise and wave_fall attributes (which select the first and the last vectors to define the sensitization patterns of the input to the output pin transition that are predefined inside the sensitization template specified at the library level).

Syntax

```plaintext
wave_rise_sampling_index : integer ;
wave_fall_sampling_index : integer ;
```

Example

```plaintext
wave_rise (2, 5, 7, 6); /* wave_rise ( wave_rise[0],
wave_rise[1], wave_rise[2], wave_rise[3] );*/
```

In the previous example, the wave rise vector delay is measured from the last transition (vector 7 changing to vector 6) to the output transition. The default wave_rise_sampling_index value is the last entry in the vector, which is 3 in this case (because the numbering begins at 0).

To override this default, set the wave_rise_sampling_index attribute, as shown:
```
wave_rise_sampling_index : 2 ;
```
When the attribute is set, the delay is measured from the second last transition of the sensitization vector to the final output transition, in other words from the transition of vector 5 to vector 7.

**when Simple Attribute**

The **when** attribute is used in state-dependent timing and conditional timing checks.

**Note:**

The **when** attribute also appears in the **min_pulse_width** group and the **minimum_period** group (described on page 3-84 and page 3-87, respectively). Both groups can be placed in **pin**, **bus**, and **bundle** groups. The **when** attribute also appears in the **power**, **fall_power**, and **rise_power** groups.

**Syntax**

```
when : "Boolean expression" ;
```

**Boolean expression**

A Boolean expression containing names of input, output, inout, and internal pins.

**Example**

```
when : "CD * SD" ;
```

**State-Dependent Timing**

In the **timing** group of a technology library, you can specify state-dependent delays that correspond to entries in OVI SDF 2.1 syntax. In state-dependent timing, the **when** attribute defines a conditional expression on which a timing arc is dependent to activate a path.

**Conditional Timing Check**

In a conditional timing check, the **when** attribute defines check-enabling conditions for timing checks such as setup, hold, and recovery.

**Conditional Timing Check in VITAL Models**

The **when** attribute is used in modeling timing check conditions for VITAL models, where, if you define **when**, you must also define **sdf_cond**.

**Syntax**

```
when : "Boolean expression" ;
```

**Boolean expression**

A valid logic expression as defined in Table 3-4 on page 3-77.
Example
when : "CLR & PRE" ;
sdf_cond : "CLR & PRE" ;

**when_end Simple Attribute**
The *when_end* attribute defines a timing-check condition specific to the end event in VHDL models.

**Syntax**
```
when_end : "Boolean expression" ;
```

*Boolean expression*
A Boolean expression containing names of input, output, inout, and internal pins.

**Example**
```
when_end : "CD * SD * Q'" ;
```

**when_start Simple Attribute**
The *when_start* attribute defines a timing-check condition specific to the start event in VHDL models.

**Syntax**
```
when_start : "Boolean expression" ;
```

*Boolean expression*
A Boolean expression containing the names of input, output, inout, and internal pins.

**Example**
```
when_start : "CD * SD" ;
```

**active_input_ccb Complex Attribute**
In referenced CCS noise modeling, the *active_input_ccb* attribute lists the active or switching *input_ccb* groups of the input pin that do not propagate the noise in the timing arc or the receiver capacitance load.

You can also specify this attribute in the *receiver_capacitance* group of the input pin.

**Syntax**
```
active_input_ccb(input_ccb_name1[, input_ccb_name2, ...]);
```

**Example**
```
active_input_ccb("A", "B");
```
active_output_ccb Complex Attribute

In referenced CCS noise modeling, the active_output_ccb attribute lists the output_ccb groups in the timing arc that drive the output pin, but do not propagate the noise. You must define both the output_ccb and timing groups in the same pin group.

Syntax
active_output_ccb(output_ccb_name);

Example
active_input_ccb("CCB_Q2");

function Complex Attribute

The function attribute can be defined in a pin or a bus group. It maps an output, inout, or an internal pin to a corresponding internal node or a variable1 or variable2 value in an ff, latch, ff_bank, or latch_bank group. The function attribute also accepts a Boolean equation containing variable1 or variable2, as well as other input, inout, or internal pins.

Example
pin (Q) {
    direction : output;
    function : "Q2";
    reference_input : "RET CK q1";
    ...
}

propagating_ccb Complex Attribute

The propagating_ccb attribute lists all the channel-connected block noise groups that propagate the noise to the output pin in a particular timing arc.

In the list, the first name is the input_ccb group of the input pin (specified by the related_pin attribute in the timing group). The second name, if present, is for the output_ccb group of the output pin.

Syntax
propagating_ccb(input_ccb_name, output_ccb_name);

Example
propagating_ccb("CCSN_CP2");
**reference_input Complex Attribute**

The `reference_input` attribute can be defined in a pin or a bus group. It specifies the input pins, which map directly to the reference pin names of the corresponding `ff`, `latch`, `ff_bank`, or `latch_bank` group. For each inout, output, or internal pin, the corresponding `ff`, `latch`, `ff_bank`, or `latch_bank` group is determined by the `variable1` or `variable2` value specified in its function statement.

**Example**

```plaintext
code
pin (Q) {
  direction : output;
  function : "Q2";
  reference_input : "RET CK q1";
  ...  
}
```

**mode Complex Attribute**

You define the `mode` attribute within a `timing` group. A `mode` attribute pertains to an individual timing arc. The timing arc is active when `mode` is instantiated with a name and a value. You can specify multiple instances of the `mode` attribute, but only one instance for each timing arc.

**Syntax**

```plaintext
code
mode (mode_name, mode_value);
```

**Example**

```plaintext
code
timing() {
  mode(rw, read);
}
```

Example 3-5 shows a `mode` description.

**Example 3-5  A mode Description**

```plaintext
code
pin(my_outpin) {
  direction : output;
  timing() {
    related_pin : b;
    timing_sense : non_unate;
    mode(rw, read);
    cell_rise(delay3x3) {
      values("1.1, 1.2, 1.3", "2.0, 3.0, 4.0", "2.5, 3.5, 4.5");
    }
    rise_transition(delay3x3) {
      values("1.0, 1.1, 1.2", "1.5, 1.8, 2.0", "2.5, 3.0, 3.5");
    }
    cell_fall(delay3x3) {
      values("1.1, 1.2, 1.3", "2.0, 3.0, 4.0", "2.5, 3.5, 4.5");
    }
    fall_transition(delay3x3) {
```
Example 3-6 shows multiple mode descriptions.

Example 3-6  Multiple mode Descriptions

library (MODE EXAMPLE) {
  delay_model              : "table_lookup";
  time_unit                : "1ns";
  voltage_unit             : "1V";
  current_unit             : "1mA";
  pulling_resistance_unit  : "1kohm";
  leakage_power_unit       : "1nW";
  capacitive_load_unit     (1, pf);
  nom_process              : 1.0;
  nom_voltage              : 1.0;
  nom_temperature          : 125.0;
  slew_lower_threshold_pct_rise  : 10;
  slew_upper_threshold_pct_rise  : 90;
  input_threshold_pct_fall       : 50;
  output_threshold_pct_fall      : 50;
  input_threshold_pct_rise       : 50;
  output_threshold_pct_rise      : 50;
  slew_lower_threshold_pct_fall  : 10;
  slew_upper_threshold_pct_fall  : 90;
  slew_derate_from_library       : 1.0;
  cell (mode_example) {
    mode_definition(RAM_MODE) {
      mode_value(MODE_1) {
      }
      mode_value(MODE_2) {
      }
      mode_value(MODE_3) {
      }
      mode_value(MODE_4) {
      }
    }
    interface_timing : true;
    dont_use         : true;
    dont_touch       : true;
    pin(Q) {
      direction          : output;
      max_capacitance    : 2.0;
      three_state        : "!OE";
      timing() {         
        related_pin      : "CK";
        timing_sense     : non_unate
        timing_type      : rising_edge
        mode(RAM_MODE,"MODE_1 MODE_2")
        cell_rise(scalar) {
          values(" 0.0 ");
        }
      }
    }
  }
}

values("1.0, 1.1, 1.2", "1.5, 1.8, 2.0", "2.5, 3.0, 3.5");

Example 3-6 Multiple mode Descriptions
cell_fall(scalar) {
  values("0.0");
}

rise_transition(scalar) {
  values("0.0");
}

fall_transition(scalar) {
  values("0.0");
}

timing() {
  related_pin : "OE"
  timing_sense : positive_unate
  timing_type : three_state_enable
  mode(RAM_MODE, MODE_2 MODE_3);
  cell_rise(scalar) {
    values("0.0");
  }
  cell_fall(scalar) {
    values("0.0");
  }
  rise_transition(scalar) {
    values("0.0");
  }
  fall_transition(scalar) {
    values("0.0");
  }
}

timing() {
  related_pin : "OE"
  timing_sense : negative_unate
  timing_type : three_state_disable
  mode(RAM_MODE, MODE_3);
  cell_rise(scalar) {
    values("0.0");
  }
  cell_fall(scalar) {
    values("0.0");
  }
  rise_transition(scalar) {
    values("0.0");
  }
  fall_transition(scalar) {
    values("0.0");
  }
}

pin(A) {
  direction : input;
  capacitance : 1.0;
  max_transition : 2.0;
  timing() {
}


timing_type : setup_rising;
related_pin : "CK";
mode(RAM_MODE, MODE_2);
rise_constraint(scalar) {
values( " 0.0 ");
}
fall_constraint(scalar) {
values( " 0.0 ");
}
}
timing() {
  timing_type : hold_rising;
  related_pin : "CK";
  mode(RAM_MODE, MODE_1);
  rise_constraint(scalar) {
values( " 0.0 ");
  }
  fall_constraint(scalar) {
values( " 0.0 ");
  }
}

pin(OE) {
direction : input;
capacitance : 1.0;
max_transition : 2.0;
}

pin(CS) {
direction : input;
capacitance : 1.0;
max_transition : 2.0;
timing() {
  timing_type : setup_rising;
  related_pin : "CK";
  mode(RAM_MODE, MODE_1);
  rise_constraint(scalar) {
values( " 0.0 ");
  }
  fall_constraint(scalar) {
values( " 0.0 ");
  }
}

timing() {
  timing_type : hold_rising;
  related_pin : "CK";
  mode(RAM_MODE, MODE_1);
  rise_constraint(scalar) {
values( " 0.0 ");
  }
  fall_constraint(scalar) {
values( " 0.0 ");
  }
}

pin(CK) {
  timing() {
    timing_type : "min_pulse_width";
    related_pin : "CK";
    mode(RAM_MODE , MODE_4);
    fall_constraint(scalar) {
      values( " 0.0 ");
    }
    rise_constraint(scalar) {
      values( " 0.0 ");
    }
  }
  timing() {
    timing_type : "minimum_period";
    related_pin : "CK";
    mode(RAM_MODE , MODE_4);
    rise_constraint(scalar) {
      values( " 0.0 ");
    }
    fall_constraint(scalar) {
      values( " 0.0 ");
    }
  }
}
clock : true;
direction : input;
capacitance : 1.0;
max_transition : 1.0;
cell_leakage_power : 0.0;
}

**pin_name_map Complex Attribute**

Similar to the *pin_name_map* attribute defined in the cell level, the timing-arc *pin_name_map* attribute defines pin names used to generate stimulus for the current timing arc. The attribute is optional when *pin_name_map* pin names are the same as (listed in order of priority)

1. pin names in the *sensitization_master* of the current timing arc.
2. pin names in the *pin_name_map* attribute of the current cell group.
3. pin names in the *sensitization_master* of the current cell group.

The *pin_name_map* attribute is required when *pin_name_map* pin names are different from all of the pin names in the previous list.

**Syntax**

```
pin_name_map (string..., string);
```
Example

```
pin_name_map (CIN0, CIN1, CK, Z);
```

**wave_rise and wave_fall Complex Attributes**

The *wave_rise* and *wave_fall* attributes represent the two stimuli used in characterization. The value for both attributes is a list of integer values, and each value is a vector ID predefined in the library sensitization group. The following example describes the *wave_rise* and *wave_fall* attributes:

```plaintext
wave_rise (vector_id[m]..., vector_id[n]);
wave_fall  (vector_id[j]..., vector_id[k]);
```

**Syntax**

```plaintext
wave_rise (integer..., integer) ;
wave_fall (integer..., integer) ;
```

**Example**

```
library(my_library) {
  ...
  sensitization(sensi_2in_1out) {
    pin_names (IN1, IN2, OUT);
    vector (0, "0 0 0");
    vector (1, "0 0 1");
    vector (2, "0 1 0");
    vector (3, "0 1 1");
    vector (4, "1 0 0");
    vector (5, "1 0 1");
    vector (6, "1 1 0");
    vector (7, "1 1 1");
  }
  cell (my_nand2) {
    sensitization_master : sensi_2in_1out;
    pin_name_map (A, B, Z); /* these are pin names for the sensitization in this cell. */
    ...
    pin(A) {
    ...
    } Pin(B) {
    ...
    } pin(Z) {
    ...
    timing() {
      related_pin : "A";
      wave_rise (6, 3); /*6, 3 - vector id in sensi_2in_1out sensitization group. Waveform interpretation of the wave_rise is (for "A,B, Z" pins): 10 1 01 */
      wave_fall (3, 6);
    }
  }
  ...
}
```
...} /* end pin(Z)*/
} /* end cell(my_nand2) */

```c
wave_rise_time_interval and wave_fall_time_interval Complex Attributes

The `wave_rise_time_interval` and `wave_fall_time_interval` complex attributes control the time interval between transitions. By default, the stimuli (specified in `wave_rise` and `wave_fall`) are widely spaced apart during characterization (for example, 10 ns from one vector to the next) to allow all output transition to stabilize. The attributes allow you to specify the duration between one vector to the next to characterize special purpose cells.

The `wave_rise_time_interval` and `wave_fall_time_interval` attributes are optional when the default time interval is used for all transitions, and they are required when you need to define special time intervals between transitions. Usually, the special time interval is smaller than the default time interval.

The `wave_rise_time_interval` and `wave_fall_time_interval` attributes can have an argument count from 1 to \(n-1\), where \(n\) is the number of arguments in corresponding `wave_rise` or `wave_fall`. Use 0 to imply the default time interval used between vectors.

**Syntax**

```c
wave_rise_time_interval (float..., float);
wave_fall_time_interval (float..., float);
```

**Example**

```c
wave_rise (2, 5, 7, 6); /* wave_rise ( wave_rise[0],
wave_rise[1], wave_rise[2], wave_rise[3] );*/
wave_rise_time_interval (0.0, 0.3);
```

The previous example suggests the following:

- Use the default time interval between `wave_rise[0]` and `wave_rise[1]` (in other words, vector 2 and vector 5).
- Use 0.3 between `wave_rise[1]` and `wave_rise[2]` (in other words, vector 5 and vector 7).
• Use the default time interval between wave_rise[2] and wave_rise[3] (in other words, vector 7 and vector 6).

ccs_retain_rise and ccs_retain_fall Groups

The ccs_retain_rise and ccs_retain_fall groups are provided in the timing group for expanded CCS retain arcs.

Syntax

cell(namestring) {
  pin (namestring) {
    timing() {
      ccs_retain_rise() {
        vector(template_namestring) {
          reference_time : float;
          index_1("float");
          index_2("float");
          index_3("float, ..., float");
          values("float, ..., float");
        }
      }
    }
  }
}

cell_degradation Group

The cell_degradation group describes a cell performance degradation design rule for compiling a design. A cell degradation design rule specifies the maximum capacitive load a cell can drive without causing cell performance degradation during the fall transition.

Syntax

pin (output pin name) {
  timing() {
    cell_degradation (template name) {
      ...cell_degradation description...
    }
  }
}

Complex Attributes

index_1 /* lookup table */
values /* lookup table */

For information about the syntax and usage of these attributes, see “cell_degradation Group” on page 3-120.

Example 3-7 Specifying cell_degradation in a Lookup Table

pin (Z) {
  timing() {
    cell_degradation (Z) {
      ...cell_degradation description...
    }
  }
}
cell_degradation (constraint) {
    index_1 ("1.0, 1.5, 2.0") ;
    values ("1.0, 1.5, 2.0") ;
}
...
...


cell_fall Group

The cell_fall group defines cell delay lookup tables (independently of transition delay) in CMOS nonlinear timing models. Define the cell_fall group in the timing group.

Note:

The same k-factors that scale the cell_fall and cell_rise values also scale the retaining_fall and retaining_rise values. There are no separate k-factors for the retaining_fall and retaining_rise values.

Syntax

library (namestring) {
    cell (namestring) {
        pin (namestring) {
            timing () {
                cell_fall (namestring) {
                    ... cell fall description...
                }
            }
        }
    }
}

Complex Attributes

index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
values ("float, ..., float", ..., "float, ...,float");

Examples from a CMOS library:

cell_fall (cell_template) {
    values ("0.00, 0.24", "0.15, 0.26") ;
}

cell_fall (cell_template) {
    values ("0.00, 0.33", "0.11, 0.38") ;
}

Each lookup table has an associated string name to indicate which lu_table_template in the library group it is to use. The name must be the same as the string name you
previously defined in the library lu_table_template. For information about the
lu_table_template syntax, see the description in “lu_table_template Group” on
page 1-50.

You can overwrite index_1, index_2, or index_3 in a lookup table, but the overwrite must
occur before the actual definition of values. The number of floating-point numbers for
index_1, index_2, or index_3 must be the same as the number you used in the
lu_table_template.

The delay value of the table is stored in the values complex attribute. It is a list of nindex_1
floating-point numbers for a one-dimensional table, nindex_1 x nindex_2 floating-point
numbers for a two-dimensional table, or nindex_1 x nindex_2 x nindex_3 floating-point
numbers for a three-dimensional table.

In a two-dimensional table, nindex_1 and nindex_2 are the size of index_1 and index_2
of the lu_table_template group. Group together nindex_1 and nindex_2 by using
quotation marks (“ ”).

In a three-dimensional table, nindex_1 x nindex_2 x nindex_3 are the sizes of index_1,
index_2, and index_3 of the lu_table_template group. Group together nindex_1,
nindex_2, and nindex_3 by using quotation marks (“ ”).

Transition and cell table delay values must be 0.0 or greater. Propagation tables can contain
negative delay values.

cell_rise Group

The cell_rise group defines cell delay lookup tables (independently of transition delay) in
CMOS nonlinear timing models.

Note:
The same k-factors that scale the cell_fall and cell_rise values also scale the
retaining_fall and retaining_rise values. There are no separate k-factors for the
retaining_fall and retaining_rise values.

Syntax
library (namestring) {
cell (namestring) {
  pin (namestring) {
    timing () {
      cell_rise (namestring){
        ... cell rise description ...
      }
    }
  }
}
}
Complex Attributes
index_1 ("float, ..., float")
index_2 ("float, ..., float")
index_3 ("float, ..., float")
values ("float, ..., float", ..., "float, ..., float")

Examples from a CMOS library

```plaintext
cell_rise(cell_template) {
  values("0.00, 0.23", "0.11, 0.28") ;
}
cell_rise(cell_template) {
  values("0.00, 0.25", "0.11, 0.28") ;
}
```

Each lookup table has an associated string name to indicate where in the library group it is to be used. The name must be the same as the string name you previously defined in the library lu_table_template. For information about the lu_table_template syntax, see the description in “lu_table_template Group” on page 1-50.”

You can overwrite index_1, index_2, or index_3 in a lookup table, but the overwrite must occur before the actual definition of values. The number of floating-point numbers for index_1, index_2, or index_3 must be the same as the number you used in the lu_table_template.

The delay value of the table is stored in a values complex attribute. It is a list of nindex_1 floating-point numbers for a one-dimensional table, nindex_1 x nindex_2 floating-point numbers for a two-dimensional table, or nindex_1 x nindex_2 x nindex_3 floating-point numbers for a three-dimensional table.

In a two-dimensional table, nindex_1 and nindex_2 are the sizes of index_1 and index_2 of the lu_table_template group. Group together nindex_1 and nindex_2 by using quotation marks (" ").

In a three-dimensional table, nindex_1 x nindex_2 x nindex_3 are the sizes of index_1, index_2, and index_3 of the lu_table_template group. Group together nindex_1, nindex_2, and nindex_3 by using by quotation marks (" ").

Each group represents a row in the table. The number of floating-point numbers in a group must equal nindex_2, and the number of groups in the values complex attribute must equal nindex_1. The floating-point nindex_2 for a one-dimensional table is "1".

Transition and cell table delay values must be 0.0 or greater. Propagation tables can contain negative delay values.

The index_3 attribute is part of the functionality that supports three-dimensional tables.
**char_config Group**

Define the `char_config` group in the `timing` group to specify the characterization settings for timing-arc constraints.

**Syntax**

```plaintext
timing() {
    char_config() {
        /* characterization configuration attributes */
    }
}
```

**Simple Attributes**

- `three_state_disable_measurement_method`
- `three_state_disable_current_threshold_abs`
- `three_state_disable_current_threshold_rel`
- `three_state_disable_monitor_node`
- `three_state_cap_add_to_load_index`
- `ccs_timing_segment_voltage_tolerance_rel`
- `ccs_timing_delay_tolerance_rel`
- `ccs_timing_voltage_margin_tolerance_rel`
- `receiver_capacitance1_voltage_lower_threshold_pct_rise`
- `receiver_capacitance1_voltage_upper_threshold_pct_rise`
- `receiver_capacitance1_voltage_lower_threshold_pct_fall`
- `receiver_capacitance1_voltage_upper_threshold_pct_fall`
- `receiver_capacitance2_voltage_lower_threshold_pct_rise`
- `receiver_capacitance2_voltage_upper_threshold_pct_rise`
- `receiver_capacitance2_voltage_lower_threshold_pct_fall`
- `receiver_capacitance2_voltage_upper_threshold_pct_fall`
- `capacitance_voltage_lower_threshold_pct_rise`
- `capacitance_voltage_lower_threshold_pct_fall`
- `capacitance_voltage_upper_threshold_pct_rise`
- `capacitance_voltage_upper_threshold_pct_fall`

**Complex Attributes**

- `driver_waveform`
- `driver_waveform_rise`
- `driver_waveform_fall`
- `input_stimulus_transition`
- `input_stimulus_interval`
- `unrelated_output_net_capacitance`
- `default_value_selection_method`
- `default_value_selection_method_rise`
- `default_value_selection_method_fall`
- `merge_tolerance_abs`
- `merge_tolerance_rel`
- `merge_selection`
Example

timing() {
    char_config() {
        driver_waveform_rise(constraint, input_driver_rise);
        driver_waveform_fall(constraint, input_driver_fall);
        ccs_timing_segment_voltage_tolerance_rel: 2.0;
    } 
}

For more information about the char_config group and the group attributes, see “char_config Group” on page 1-28.

compact_ccs_retain_rise and compact_ccs_retain_fall Groups

The compact_ccs_retain_rise and compact_ccs_retain_fall groups are provided in the timing group for compact CCS retain arcs.

Syntax

pin(pin_name) {
    direction : string;
    capacitance : float;
    timing() {
        compact_ccs_retain_rise (template_name) {
            base_curves_group : "base_curves_name";
            index_1 ("float…, float");
            index_2 ("float…, float");
            index_3 ("string…, string");
            values ("…”…)
        }
    }
}

compact_ccs_rise and compact_ccs_fall Groups

The compact_ccs_rise and compact_ccs_fall groups define the compact CCS timing data in the timing arc.

Syntax

compact_ccs_rise (template_name) {
compact_ccs_fall (template_name) {

Example

timing() {
    compact_ccs_rise (LTT3) {
        base_curves_group : "ctbct1";
        values ("0.1, 0.5, 0.6, 0.8, 1, 3", 
                  "0.15, 0.55, 0.65, 0.85, 2, 4", 
                  "0.2, 0.6, 0.7, 0.9, 3, 2", 
                  "0.25, 0.65, 0.75, 0.95, 4, 1");
    }
}
compact_ccs_fall (LTT3) {
  values ("-0.12, -0.51, 0.61, 0.82, 1, 2", \\
  "-0.15, -0.55, 0.65, 0.85, 1, 4", \\
  "-0.24, -0.67, 0.76, 0.95, 3, 4", \\
  "-0.25, -0.65, 0.75, 0.95, 3, 1");
}

Simple Attribute
base_curves_group

Complex Attribute
values

base_curves_group Simple Attribute

The base_curves_group attribute is optional at this level when base_curves_name is the same as that defined in the compact_lut_template that is being referenced by the compact_ccs_rise or compact_ccs_fall group.

Syntax
base_curves_group : "base_curves_name" ;

Example
base_curves_group : "ctbct1" ;

values Complex Attribute

The values attribute defines the compact CCS timing data values. The values are determined by the index_3 values.

Syntax
values ("float, float, ...", "float, float, ...");

Example
  values ("0.1, 0.5, 0.6, 0.8, 1, 3", \\
  "0.15, 0.55, 0.65, 0.85, 2, 4", \\
  "0.2, 0.6, 0.7, 0.9, 3, 2", \\
  "0.25, 0.65, 0.75, 0.95, 4, 1");

fall_constraint Group

Together with the rise_constraint group, the fall_constraint group defines timing constraints (cell delay lookup tables) sensitive to clock or data input transition times.

The fall_constraint group is defined in a timing group, as shown here:
  library (name_string) {

cell (\texttt{namestring}) {
  pin (\texttt{namestring}) {
    timing () {
      \texttt{fall\_constraint (namestring)}{
      \hspace{1em} ... \hspace{1em} \texttt{fall\_constraint described} ... 
    }
  }
}
}

\textbf{Complex Attributes}

\begin{itemize}
  \item \textbf{index\_1 ("float, \ldots, float")};
  \item \textbf{index\_2 ("float, \ldots, float")};
  \item \textbf{index\_3 ("float, \ldots, float")};
  \item \textbf{values ("float, \ldots, float", \ldots, "float, \ldots, float")};
\end{itemize}

\textbf{Example}

\begin{verbatim}
fall\_constraint(constraint\_template) {
  values ("0.0, 0.14, 0.20", \ "0.22, 0.24, 0.42", \ "0.34, 0.38, 0.51");
}
...

rise\_constraint(constraint\_template) {
  values ("0.0, 0.13, 0.19", \ "0.21, 0.23, 0.41", \ "0.33, 0.37, 0.50");
}
\end{verbatim}

\textbf{Example 3-8 on page 3-143} shows constraints in a timing model.

\textbf{fall\_propagation Group}

Together with the \texttt{rise\_propagation} group, the \texttt{fall\_propagation} group specifies transition delay as a term in the total cell delay.

The \texttt{fall\_propagation} group is defined in the \texttt{timing} group, as shown here.

\begin{verbatim}
library (\texttt{namestring}) {
  cell (\texttt{namestring}) {
    pin (\texttt{namestring}) {
      timing () {
        fall\_propagation (\texttt{namestring}){
        \hspace{1em} ... \hspace{1em} \texttt{fall\_propagation described} ... 
      }
    }
  }
}
\end{verbatim}
Complex Attributes

index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
values ("float, ..., float", ..., "float, ..., float");

Example

fall_propagation (prop_template) {
  values ("0.02, 0.15", "0.12, 0.30") ;
}
rise_propagation (prop_template) {
  values ("0.04, 0.20", "0.17, 0.35") ;
}

fall_transition Group

The fall_transition group is defined in the timing group, as shown here:

library (namestring) {
  cell (namestring) {
    pin (namestring) {
      timing () {
        fall_transition (namestring){
          ...
        }
      }
    }
  }
}

Complex Attributes

index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
values ("float, ..., float", ..., "float, ..., float");

intermediate_values ("float, ..., float", ..., "float, ...
... , float");

Note:
As an option, you can use the intermediate_values table attribute to specify the transition from the first slew point to the output delay threshold. The intermediate_values table attribute has to use the same format as the table attribute.

Example

fall_transition(tran_template) {
  values ("0.01, 0.11, 0.18, 0.40");
ocv_sigma_cell_fall Group

Use this optional group to specify a lookup table for the fall delay on-chip variation (OCV) values. In the lookup table, each absolute fall-delay variation offset from the corresponding nominal fall delay is specified at one sigma (σ), where sigma is the standard deviation of the fall delay distribution.

Note:
The ocv_sigma_cell_fall group is part of the Liberty variation format (LVF) syntax. The Liberty variation format (LVF) is used to represent the parametric OCV library data, such as slew-load table per delay timing arc. The Liberty variation format (LVF) syntax consists of groups for sigma variation cell delay, output transition or slew, and constraint tables that are a function of load and input-slew. The variation values are used during parametric OCV analysis.

In a parametric OCV model, the random variation is specific to a cell or a timing arc, unlike an advanced OCV model where a specific derating factor applies to multiple cells or an entire library.

You define the ocv_sigma_cell_fall group in the timing group of an output pin, as shown here:

```plaintext
library (namestring) {
  cell (namestring) {
    pin (namestring) {
      timing () {
        ocv_sigma_cell_fall (template_namestring){
          ... values description...
        }
      }
    }
  }
}
```

**template_name**

The name of a lu_table_template group.

For more information about the ocv_sigma_cell_fall group syntax and attributes, see *Liberty User Guide, Vol. 1*.

**Simple Attribute**

sigma_type
sigma_type Simple Attribute

Specify the optional sigma_type attribute to define the type of arrival time listed in the ocv_sigma_cell_rise, ocv_sigma_cell_fall, ocv_sigma_rise_transition, and ocv_sigma_fall_transition group lookup tables. The values are early, late, and early_and_late. The default is early_and_late.

You can specify the sigma_type attribute in the ocv_sigma_cell_rise and ocv_sigma_cell_fall groups.

Syntax
sigma_type: early | late | early_and_late;

Example
sigma_type: early;

ocv_sigma_cell_rise Group

Use this optional group to specify a lookup table of the rise delay on-chip variation (OCV) values. In the lookup table, each absolute rise-delay variation offset from the corresponding nominal rise delay is specified at one sigma (σ), where sigma is the standard deviation of the rise delay distribution.

Note:
The ocv_sigma_cell_rise group is part of the parametric OCV Liberty variation format (LVF) syntax.

For more information about the ocv_sigma_cell_rise group syntax, see "ocv_sigma_cell_fall Group" on page 3-129.

ocv_sigma_fall_constraint Group

Use this optional group to specify a lookup table for the fall constraint on-chip variation (OCV) values. In the lookup table, each absolute fall-constraint variation offset from the corresponding nominal fall constraint is specified at one sigma (σ), where sigma is the standard deviation of the fall constraint distribution.

Note:
The ocv_sigma_fall_constraint group is part of the Liberty variation format (LVF) syntax. The Liberty variation format (LVF) is used to represent the parametric OCV library data, such as slew-load table per delay timing arc. The Liberty variation format (LVF) syntax consists of groups for sigma variation cell delay, output transition or slew, and constraint tables that are a function of load and input-slew. The variation values are used during parametric OCV analysis.
In a parametric OCV model, the random variation is specific to a cell or a timing arc, unlike an advanced OCV model where a specific derating factor applies to multiple cells or an entire library.

You define the `ocv_sigma_fall_constraint` group in a `timing` group of an input or inout pin, as shown here:

```plaintext
library (namestring) {
    cell (namestring) {
        pin (namestring) {
            timing () {
                ocv_sigma_fall_constraint (template_namestring){
                    ... values description...
                }
            }
        }
    }
}
```

`template_name`

The name of a `lu_table_template` group.

For more information about the `ocv_sigma_fall_constraint` group syntax and attributes, see Liberty User Guide, Vol. 1.

**ocv_sigma_fall_transition Group**

Use this optional group to specify a lookup table for the fall transition on-chip variation (OCV) values. In the lookup table, each absolute fall-transition variation offset from the nominal fall transition is specified at one sigma (σ), where sigma is the standard deviation of the fall transition distribution.

**Note:**

The `ocv_sigma_fall_transition` group is part of the Liberty variation format (LVF) syntax. The Liberty variation format (LVF) is used to represent the parametric OCV library data, such as slew-load table per delay timing arc. The Liberty variation format (LVF) syntax consists of groups for sigma variation cell delay, output transition or slew, and constraint tables that are a function of load and input-slew. The variation values are used during parametric OCV analysis.

In a parametric OCV model, the random variation is specific to a cell or a timing arc unlike an advanced OCV model where a specific derating factor applies to multiple cells or an entire library.
You define the `ocv_sigma_fall_transition` group in the `timing` group of an output pin, as shown here:

```
library (namestring) {
  cell (namestring) {
    pin (namestring) {
      timing () {
        ocv_sigma_fall_transition (template_namestring){
          ... values description...
        }
      }
    }
  }
}
```

`template_name`

The name of a `lu_table_template` group.

For more information about the `ocv_sigma_fall_transition` group syntax and attributes, see `Liberty User Guide, Vol. 1`.

**Simple Attribute**

`sigma_type`

**sigma_type Simple Attribute**

Specify the optional `sigma_type` attribute to define the type of arrival time specified in the `ocv_sigma_cell_rise`, `ocv_sigma_cell_fall`, `ocv_sigma_rise_transition`, and `ocv_sigma_fall_transition` group lookup tables. The values are `early`, `late`, and `early_and_late`. The default is `early_and_late`.

**Syntax**

```
sigma_type: early | late | early_and_late;
```

**Example**

```
sigma_type: early;
```

**ocv_sigma_rise_constraint Group**

Use this optional group to specify a lookup table of the rise constraint on-chip variation (OCV) values. In the lookup table, each absolute rise-constraint variation offset from the nominal rise constraint is specified at one sigma ($\sigma$), where sigma is the standard deviation of the rise constraint distribution.

**Note:**

The `ocv_sigma_rise_constraint` group is part of the parametric OCV Liberty variation format (LVF) syntax.
Do not specify the `sigma_type` attribute in the `ocv_sigma_rise_constraint` and group.

For more information about the `ocv_sigma_rise_constraint` group syntax, see “ocv_sigma_fall_constraint Group” on page 3-130.

**ocv_sigma_rise_transition Group**

Use this optional group to specify a lookup table of the rise transition on-chip variation (OCV) values. In the lookup table, each absolute rise-transition variation offset from the nominal rise transition is specified at one sigma (\(\sigma\)), where sigma is the standard deviation of the rise transition distribution.

Note:
The `ocv_sigma_rise_transition` group is part of the parametric OCV Liberty variation format (LVF) syntax.

For more information about the `ocv_sigma_rise_transition` group syntax, see “ocv_sigma_fall_transition Group” on page 3-131.

**ocv_sigma_retaining_fall Group**

Use the `ocv_sigma_retaining_fall` group to specify the absolute variation offset from the nominal retain arc table values at one sigma (\(\sigma\)), where sigma is the standard deviation of the delay distribution.

Note:
The `ocv_sigma_retaining_rise` and `ocv_sigma_retaining_fall` groups are part of the Liberty variation format (LVF) retain arc model syntax. The variation values are used during parametric OCV analysis.

You define the `ocv_sigma_retaining_rise` and `ocv_sigma_retaining_fall` groups in the `timing` group of an output pin, as shown here:

```plaintext
library (name) {
  cell (name) {
    pin (name) {
      timing () {
        ocv_sigma_retaining_rise (template_name){
          ... values description...
        }...
        ocv_sigma_retaining_fall (template_name){
          ... values description...
        }
      }
    }
  }
}
```
template_name

The name of a lu_table_template group.

For more information about the ocv_sigma_retaining_fall group syntax and attributes, see Liberty User Guide, Vol. 1.

Simple Attribute

sigma_type

sigma_type Simple Attribute

Specify the optional sigma_type attribute to define the type of arrival time in the ocv_sigma_retaining_rise and ocv_sigma_retaining_fall group lookup tables. The values are early, late, and early_and_late. The default is early_and_late.

Syntax

sigma_type: early | late | early_and_late;

Example

sigma_type: early;

ocv_sigma_retaining_rise Group

Use the ocv_sigma_retaining_rise group to specify the absolute variation offset from the nominal retain arc table values at one sigma(σ), where sigma is the standard deviation of the delay distribution.

For more information, see “ocv_sigma_retaining_fall Group” on page 3-133.

ocv_sigma_retain_fall_slew Group

Use the ocv_sigma_retain_fall_slew group to specify the absolute variation offset from the nominal retain arc table values at one sigma(σ), where sigma is the standard deviation of the delay distribution.

Note:

The ocv_sigma_retain_rise_slew and ocv_sigma_retain_fall_slew groups are part of the Liberty variation format (LVF) retain arc model syntax. The variation values are used during parametric OCV analysis.

You define the ocv_sigma_retain_rise_slew and ocv_sigma_retain_fall_slew groups in the timing group of an output pin, as shown here:

```
library (name) {
  cell (name) {
    pin (name) {
      timing () {
```
ocv_sigma_retain_rise_slew (template_name){
    ... values description...
}

ocv_sigma_retain_fall_slew (template_name){
    ... values description...
}

*template_name*

The name of a *lu_table_template* group.

For more information about the *ocv_sigma_retain_fall_slew* group syntax and attributes, see *Liberty User Guide, Vol. 1*.

**Simple Attribute**

*sigma_type*

**sigma_type Simple Attribute**

Specify the optional *sigma_type* attribute to define the type of arrival time in the *ocv_sigma_retain_rise_slew* and *ocv_sigma_retain_fall_slew* group lookup tables. The values are *early*, *late*, and *early_and_late*. The default is *early_and_late*.

**Syntax**

*sigma_type*: early | late | early_and_late;

**Example**

*sigma_type*: early;

**ocv_sigma_retain_rise_slew Group**

Use the *ocv_sigma_retain_rise_slew* group to specify the absolute variation offset from the nominal retain arc table values at one sigma(σ), where sigma is the standard deviation of the delay distribution.

For more information, see "*ocv_sigma_retain_fall_slew Group*" on page 3-134.

**ocv_mean_shift_* Groups**

Use the *ocv_mean_shift_cell_rise*, *ocv_mean_shift_cell_fall*, *ocv_mean_shift_rise_transition*, *ocv_mean_shift_fall_transition*, *ocv_mean_shift_retaining_rise*, *ocv_mean_shift_retaining_fall*, *ocv_mean_shift_retain_rise_slew*, and *ocv_mean_shift_retain_fall_slew*,
ocv_mean_shift_rise_constraint, and ocv_mean_shift_fall_constraint lookup tables to specify the offset value from the mean of an asymmetric or a biased timing variation distribution. These groups are part of the moment-based Liberty variation format (LVF) syntax.

**ocv_skewness_* Groups**

Use the ocv_skewness_cell_rise, ocv_skewness_cell_fall, ocv_skewness_rise_transition, ocv_skewness_fall_transition, ocv_skewness_retaining_rise, ocv_skewness_retaining_fall, ocv_skewness_retain_rise_slew, ocv_skewness_retain_fall_slew, ocv_skewness_rise_constraint, ocv_skewness_fall_constraint lookup tables to specify the skewness values of an asymmetric or a biased timing variation distribution. These groups are part of the moment-based Liberty variation format (LVF) syntax.

**ocv_std_dev_* Groups**

Use the ocv_std_dev_cell_rise, ocv_std_dev_cell_fall, ocv_std_dev_rise_transition, ocv_std_dev_fall_transition, ocv_std_dev_retaining_rise, ocv_std_dev_retaining_fall, ocv_std_dev_retain_rise_slew, ocv_std_dev_retain_fall_slew, ocv_std_dev_rise_constraint, and ocv_std_dev_fall_constraint lookup tables to specify the values of standard deviation of an asymmetric or a biased timing variation distribution. These groups are part of the moment-based Liberty variation format (LVF) syntax.

**output_current_fall Group**

Use the output_current_fall and the output_current_rise groups to specify the output current for a nonlinear lookup table model.

The output_current_fall group is defined in the timing group, as shown here.

```plaintext
library 'name_string' { 
cell 'name_string' { 
  pin 'name_string' { 
    timing () { 
      output_current_fall 'name_string'{
        ... description ...
      }
    }
  }
}
```
Groups
vector

vector Group

Use the vector group to store information about the input slew and output load.

The vector group is defined in the output_current_fall group, as shown:

```plaintext
library (namestring) {
    cell (namestring) {
        pin (namestring) {
            timing () {
                output_current_fall (namestring){
                    vector () {
                        ... description ...
                    }
                }
            }
        }
    }
}
```

Simple Attribute
reference_time

reference_time Simple Attribute

Use the reference_time attribute to specify the time at which the input waveform crosses the rising or falling input delay threshold.

The reference_time attribute is defined in the vector group.

```plaintext
library (namestring) {
    cell (namestring) {
        pin (namestring) {
            timing () {
                output_current_fall (namestring){
                    vector () {
                        reference_time : ;
                    }
                }
            }
        }
    }
}
```

Example

```plaintext
timing () {
    output_current_rise () {
        vector (CCT){
```
output_current_rise Group

For information about using the `output_current_rise` group, see the definition of the "output_current_fall Group" on page 3-136.

receiver_capacitance_fall Group

In the multisegment receiver capacitance model, define the `receiver_capacitance_fall` group to reference a lookup table template. For information about this group, see "receiver_capacitance_fall Group" on page 3-90.

receiver_capacitance_rise Group

For information about using the `receiver_capacitance_rise` group, see "receiver_capacitance_fall Group" on page 3-90.

receiver_capacitance1_fall Group

You can define the `receiver_capacitance1_fall` group at the pin level and at the timing level. Define the `receiver_capacitance1_fall` group at the timing level to specify receiver capacitance for a timing arc. For information about using the group at the pin level, see "receiver_capacitance1_fall Group" on page 3-89.

Syntax

```java
receiver_capacitance1_fall (value) {

Complex Attribute

description

Example

timing() {

```
receiver_capacitance1_rise Group

For information about using the receiver_capacitance1_rise group, see the description of the receiver_capacitance1_fall Group.

receiver_capacitance2_fall Group

For information about using the receiver_capacitance2_fall group, see the description of receiver_capacitance1_fall Group.

receiver_capacitance2_rise Group

For information about using the receiver_capacitance2_rise group, see the description of the receiver_capacitance1_fall Group.

retaining_fall Group

The retaining_fall group specifies the length of time the output port retains its current logical value of 1 after the output port’s corresponding input port’s value has changed.

This attribute is used only with nonlinear delay models.

Note:

The same k-factors that scale the cell_fall and cell_rise values also scale the retaining_fall and retaining_rise values. There are no separate k-factors for the retaining_fall and retaining_rise values.

Syntax

library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        retaining_fall (name_string) {
          ... retaining fall description ...
        }
      }
    }
  }
}

Complex Attributes

index_1 ("float, ... , float");
index_2 ("float, ... , float");
index_3 ("float, ... , float");
values ("float, ... , float", "float, ... , float", "float, ... , float");
Example

```plaintext
retaining_rise (retaining_table_template) {
    values ("0.00, 0.23", "0.11, 0.28") ;
}
retaining_fall (retaining_table_template) {
    values ("0.01, 0.30", "0.12, 0.18") ;
}
```

**retaining_rise Group**

The `retaining_rise` group specifies the length of time an output port retains its current logical value of 0 after the output port’s corresponding input port’s value has changed.

This attribute is used only with nonlinear delay models.

**Note:**

The same k-factors that scale the `cell_fall` and `cell_rise` values also scale the `retaining_fall` and `retaining_rise` values. There are no separate k-factors for the `retaining_fall` and `retaining_rise` values.

**Syntax**

```plaintext
library (namestring) {
    cell (namesstring) {
        pin (namestring) {
            timing () {
                retaining_rise (namestring){
                    ... retaining rise description ...
                }
            }
        }
    }
}
```

**Complex Attributes**

```plaintext
index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
values ("float, ..., float", "float, ...,float", "float, ...
,...,float");
```

Example

```plaintext
retaining_rise (retaining_table_template) {
    values ("0.00, 0.23", "0.11, 0.28") ;
}
retaining_fall (retaining_table_template) {
    values ("0.01, 0.30", "0.12, 0.18") ;
}
```
retain_fall_slew Group

Use this group in the timing group to define a slew table associated with the retaining_fall delay. The slew table describes the rate of decay of the output logic value.

Syntax
retain_fall_slew (retaining_time_templatestring) {
   values (index1float, index2float, index3float) ;
}

retaining_time_template
Name of the table template to use for the lookup table.

index1, index2, index3
Values to use for indexing the lookup table.

Examples

cell (cell_name) {
   ...
   pin (pin_name) {
      direction : output :
      ...
      timing ( ) {
         related_pin : "related_pin" ;
         ...
         retaining_fall (retaining_table_template) {
            values ( "0.00, 0.23", "0.11, 0.28") ;
         }
         ...
         retain_fall_slew (retaining_time_template) {
            values ( "0.01, 0.02" ) ;
         }
         ...
      }
   }
}

retain_rise_slew Group

Use this group in the timing group to define a slew table associated with the retaining_rise delay. The slew table describes the rate of decay of the output logic value.

Syntax
retain_rise_slew (retaining_time_templatestring) {
   values (index1float, index2float, index3float) ;
}
retaining_time_template

Name of the table template to use for the lookup table.

index1float index2float index3float

Values to use for indexing the lookup table.

Examples

cell (cell_name) {

... pin (pin_name) {

direction : output :
...

timing ( ) {

related_pin : "related_pin"
...

timing (retaining_table_template) {

values ( "0.00, 0.23", "0.11, 0.28") ;
}
...

timing (retain_rise_slew (retaining_time_template) {   
values ( "0.01, 0.02") ;
}
...

}

}

rise_constraint Group

Together with the fall_constraint group, the rise_constraint group defines timing constraints (cell delay lookup tables) sensitive to clock or data input transition times.

Syntax

library (namestring) {

cell (namestring) {

pin (namestring) {

timing () {

rise_constraint (namestring) {

... values description...

}
}
}

}
Complex Attributes

index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
values ("float, ..., float", ..., "float, ..., float");

Example

rise_constraint(constraint_template) {
  values ("0.0, 0.13, 0.19", \
       "0.21, 0.23, 0.41", \n       "0.33, 0.37, 0.50");
}

Example 3-8 shows constraints in a timing model.

Example 3-8  CMOS Nonlinear Timing Model Using Constraints

library( vendor_b ) {
  /* 1. Use delay lookup table */
  delay_model : table_lookup;
  /* 2. Define table template of size 3 x 3*/
  lu_table_template(constraint_template) {
    variable_1 : constrained_pin_transition;
    variable_2 : related_pin_transition;
    index_1 ("0.0, 0.5, 1.5");
    index_2 ("0.0, 2.0, 4.0");
  }
  ...
  cell(dff) {
    pin(d) {
      direction: input;
      timing( "t1" | "t1", "t2", "t3" ) {
        related_pin : "clk";
        timing_type : setup_rising;
        ...
        /* Inherit the ‘constraint_template’ template */
        rise_constraint(constraint_template) {
          /* Specify all the values */
          values ("0.0, 0.13, 0.19", \
                 "0.21, 0.23, 0.41", \
                 "0.33, 0.37, 0.50");
        }
        fall_constraint(constraint_template) {
          values ("0.0, 0.14, 0.20", \
                 "0.22, 0.24, 0.42", \
                 "0.34, 0.38, 0.51");
        }
      }
    }
  }
}
rise_propagation Group

With the fall_propagation group, the rise_propagation group specifies transition delay as a term in the total cell delay.

Syntax

```plaintext
library (name_string) {  
cell (name_string) {  
  pin (name_string) {  
    timing () {  
      rise_propagation (name_string){  
        ... rise propagation description ...  
      }  
    }  
  }  
}  
}
```

Complex Attributes

```plaintext
index_1 ("float, ..., float");  
index_2 ("float, ..., float");  
index_3 ("float, ..., float");  
values ("float, ..., float", ..., "float, ..., float");
```

Example

```plaintext
fall_propagation (prop_template) {  
  values("0.00, 0.21","0.14, 0.38") ;  
}  
rise_propagation (prop_template) {  
  values("0.05, 0.25","0.15, 0.48") ;  
}
```

rise_transition Group

The rise_transition group is defined in the timing group, as shown here:

```plaintext
library (name_string) {  
cell (name_string) {  
  pin (name_string) {  
    timing () {  
      rise_transition (name_string){  
        ... rise transition description ...  
      }  
    }  
  }  
}  
}
```
Complex Attributes

index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
values ("float, ..., float", ..., "float, ..., float");

intermediate_values ("float, ..., float", ..., "float, ..., float");

Note:
Optionally, you can use the intermediate_values table attribute to specify the transition from the first slew point to the output delay threshold. The intermediate_values table attribute has to use the same format as the table attribute.

Examples

rise_transition(tran_template) {
  values ("0.01, 0.08, 0.15, 0.40");
}

fall_transition(tran_template) {
  values ("0.01, 0.11, 0.18, 0.40");
}

---

tlatch Group

In timing analysis, use a tlatch group to describe the relationship between the data pin and the enable pin on a transparent level-sensitive latch.

You define the tlatch group in a pin group, but it is only effective if you also define the timing_model_type attribute in the cell that the pin belongs to. For more information about the timing_model_type attribute, see “timing_model_type Simple Attribute” on page 2-24.

Syntax

library (namestring) {
  cell (namestring) {
    ...
    timing_model_type : valueenum ;
    ....
  pin (data_pin_namestring) {
    tlatch (enable_pin_namestring) {
      ... tlatch description ...
    }
  }
}
}
Simple Attributes

edge_type

tdisable

disable Simple Attribute

Use the edge_type attribute to specify whether the latch is positive (high) transparent or negative (low) transparent.

Syntax

disable : valueBoolean

value

Valid values are rising and falling.

Example

disable : TRUE ;

tdisable Simple Attribute

The tdisable attribute disables transparency in a latch. During path propagation, timing analysis tools disable and ignore all data pin output pin arcs that reference a t latch group whose tdisable attribute is set to true on an edge triggered flip flop.

Syntax

tdisable : valueBoolean

value

The valid values are TRUE and FALSE. When set to FALSE, the latch is ignored.

Example

tdisable : FALSE ;