Liberty Reference Manual

Version R-2020.09, September 2020

Copyright Notice

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 https://www.synopsys.com/company/legal/trademarks-brands.html.
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

Library Group Description and Syntax

This chapter describes the role of the `library` group in defining a CMOS logic 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.

This chapter provides information about the `library` group in the following sections:

Library-Level Attributes and Values

The `library` group is the superior group in a logic library. The `library` group contains all the groups and attributes that define the logic library.

Example 1 lists alphabetically a sampling of the attributes, groups, and values that you can define within a logic library. The example in General Syntax shows the general syntax and the functional order in which the attributes usually appear within a `library` group.

Example 1 Attributes, Groups, and Values in a Logic Library

```
library (name_string) {
   ... library description ...
}
```

```
  /* Library Description: Simple Attributes */

altitude_unit : value_enum ;
bus_naming_style : "string" ;
comment : "string" ;
current_unit : value_enum ;
date : "date" ;
delay_model : value_enum ;
em_temp_degradation_factor : float ;
input_threshold_pct_fall : trip_point value ;
input_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 ;
soft_error_rate_confidence : float ;
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 : name_string ;
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 : name_string ;

  /* 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_name_id, speed_grade_id) ;
define (name, object, type) ; /*user—defined attributes only */
define_cell_area (area_name, resource_type) ;
define_group (attribute_name_string, group_name_string,attribute_type_string  ;
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) { }
default_soft_error_rate (name) { }
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_name_id ) { }
rise_net_delay : name ;
rise_transition_degradation ()  { }
soft_error_rate_template (name) { }
timing (name | name_list) { }
type (name)  { }
voltage_state_range_list
wire_load (name)  { }
wire_load_selection (name)  { }
wire_load_table (name)  { }
```

General Syntax

The example in Library-Level Attributes and Values 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 2 does not contain every attribute or group listed in the example in Library-Level Attributes and Values.

Example 2 General Syntax of a Logic Library

```
library (name_string) {

  /* Library-Level Simple and Complex Attributes */

 technology (name_enum ) ;
 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-Level Group Statements */

 operating_conditions (name_string)  {
  ... operating conditions description ...
 }
 wire_load (name_string)  {
   ... wire load description ...
 }
 wire_load_selection (name_string)  {
   ... wire load selection criteria...
 }
 power_lut_template (name_string)  {
   ... power lookup table template information...
 }
    /* Cell definitions */
```

```
 cell (name_string2)  {
   ... cell description ...
 }
 ...
 type (name_string) {
   ... type description ...
 }
 input_voltage (name_string) {
   ... input voltage information ...
 }
 output_voltage (name_string)  {
   ... output voltage information ...
 }
}
```

library Group Name

The first line of the `library` group statement names the library. It is the first executable line in your library.

Syntax

```
library(name_string) {
   ... library description ...
}
```

Example

A library called example1 looks like this:

```
library (example1) {
   ... library description...
}
```

library Group Example

Example 3 shows a portion of a library group for a CMOS library. It contains buses and uses a nonlinear timing delay model.

Example 3 CMOS library Group

```
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.

altitude_unit Simple Attribute

The `altitude_unit` attribute specifies the unit of altitude in the library. Valid values are 1 m (default) or 1 km.

Syntax

```
altitude_unit : value;
```

Example

```
altitude_unit : 1km ;
```

bus_naming_style Simple Attribute

The `bus_naming_style` 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`.

If you do not define the `bus_naming_style` attribute, the default naming convention is applied, as shown.

Example

```
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.

Note:

If you do not define a `delay_model` attribute, `table_lookup`.

Syntax

`delay_model : value`$_{enum}$ `;`

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

```
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

```
device_layer(string) {} /* such as diffusion layer OD */
```

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");
}

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

```
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 ;
```

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_point_float ;
```

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_point_float ;
```

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

```
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*.

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

```
leakage_power_unit : value_enum ;
```

value

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

Example

```
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

```
nom_process : value_float ;
```

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_point_float ;
```

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_point_float ;
```

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 ;
```

power_model Simple 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

```
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" ;
```

revision Simple Attribute

The optional `revision` attribute defines a revision number for your library.

Syntax

```
revision : value  ;
```

value

The value can be either a floating-point number or a string.

Example

```
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

```
slew_derate_from_library : derate_float ;
```

derate

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

Example

```
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_point_value ;
```

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_point_value ;
```

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_point_value ;
```

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_point_value ;
```

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 ;
```

soft_error_rate_confidence Simple Attribute

The `soft_error_rate_confidence` attribute specifies the confidence level at which the cell soft error rate is sampled in the library. The value range is from 0 to 1.

Syntax

```
soft_error_rate_confidence : float ;
```

Example

```
soft_error_rate_confidence : 0.5 ; /* confidence level 50% */
```

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

```
voltage_unit : unit ;
```

unit

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

Example

```
voltage_unit : 100mV;
```

Defining Default Attribute Values in a CMOS Logic Library

Within the `library` group of a CMOS logic 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 lists the default attributes that you can define within the `library` group and the attributes that override them.

Table 1 CMOS Default Attributes for All Models

Default attribute	Description	Override with
`default_cell_leakage_power`	Default leakage power	`cell_leakage_power`
`default_connection_class`	Default connection class	`connection_class`
`default_fanout_load`	Fanout load of input pins	`fanout_load`
`default_inout_pin_cap`	Capacitance of inout pins	`capacitance`
`default_input_pin_cap`	Capacitance of input pins	`capacitance`
`default_max_capacitance`	Maximum capacitance of output pins	`max_capacitance`
`default_max_fanout`	Maximum fanout of all output pins	`max_fanout`
`default_max_transition`	Maximum transition of output pins	`max_transition`
`default_operating_conditions`	Default operating conditions for the library	`operating_conditions`
`default_output_pin_cap`	Capacitance of output pins	`capacitance`
`default_wire_load`	Wire load	No override available
`default_wire_load_area`	Wire load area	No override available
`default_wire_load_capacitance`	Wire load capacitance	No override available
`default_wire_load_mode`	Wire load mode	`set_wire_load -mode`
`default_wire_load_resistance`	Wire load resistance	No override available
`default_wire_load_selection`	Wire load selection	No override available

Complex Attributes

Following are the descriptions of the logic library complex attributes.

capacitive_load_unit Complex Attribute

The `capacitive_load_unit` attribute specifies the unit for all capacitance values within the logic library, including default, maximum fanout, pin, and wire capacitances.

Syntax

```
capacitive_load_unit (value_float,unit_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.

```
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

```
default_part (default_part_name_string, speed_grade_string) ;
```

default_part_name

The name of the default part.

speed_grade

The speed grade the design uses.

Example

```
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

```
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 2).

Syntax

```
technology (name_enum) ;
```

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

```
voltage_map (voltage_name_id, voltage_value_float) ;
```

voltage_name

Specifies a power supply.

voltage_value

Specifies a voltage value.

Example

```
voltage_map (VDD1, 3.0) ;
```

Group Statements

Following are the descriptions of the logic 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

```
library (my_compact_ccs_lib) {
   …
   base_curves (base_curves_name) {
   …
   }
}
```

Example

```
library(my_lib) {
   …
   base_curves (ctbct1) {
   …
   }
}
```

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_timing_half_curve` and `ccs_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

```
base_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 : ctbc1 ;
```

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 2 lists the valid characterization models for the `char_config` group attributes.

Table 2 Valid Characterization Models for the char_config Group

Model	Description
`all`	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.
`nldm`	Default nonlinear delay model (NLDM)
`nldm_delay` `nldm_transition`	Specific NLDMs that have higher priority over the default NLDM
`capacitance`	Capacitance model
`constraint`	Default constraint model

Model	Description

`constraint_setup`
`constraint_hold`
`constraint_recovery`
`constraint_removal`
`constraint_skew`
`constraint_min_pulse_width`
`constraint_no_change`
`constraint_non_seq_setup`
`constraint_non_seq_hold`
`constraint_minimum_period`
	Specific constraint models with higher priority over the default constraint model

`nlpm`	Default nonlinear power model (NLPM)

`nlpm_leakage`
`nlpm_input`
`nlpm_output`
	Specific NLPM with higher priority over the default NLPM

Selection Methods

Table 3 lists the valid selection methods used by the `char_config` group attributes.

Table 3 Valid Selection Methods for the char_config Group

Method	Description
`any`	Selects a random value from the state-dependent data.
`min`	Selects the minimum value from the state-dependent data at each index point.
`max`	Selects the maximum value from the state-dependent data at each index point.
`average`	Selects an average value from the state-dependent data at each index point.
`min_mid_table`	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.

Method	Description
max_mid_table	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.
follow_delay	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.

Example

```
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 ;
```

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_rise : float ;
```

Example

```
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_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 3.

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

```
default_value_selection_method_rise ( char_model, method ) ;
```

For valid values of the `method` argument, see Table 3.

Example

```
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

```
default_value_selection_method_fall ( char_model, method ) ;
```

For valid values of the `method` argument, see Table 3.

Example

```
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

```
merge_tolerance_abs ( char_model, float ) ;
```

Example

```
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

```
merge_tolerance_rel ( char_model, float ) ;
```

Example

```
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

```
merge_selection ( char_model, method ) ;
```

For valid values of the `method` argument, see Table 3.

Example

```
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

```
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") ;
 }
 ...
}
```

default_soft_error_rate Group

The `default_soft_error_rate` group is a lookup table that specifies the default cell soft error rate (SER) in the library. The table values must be greater than or equal to zero.

The default table applies to cells in libraries where the cell-level `soft_error_rate` group is not defined. The table can be one-dimensional or two-dimensional. It is recommended to specify the library-level `default_soft_error_rate` table even for libraries that contain a cell-level `soft_error_rate` group.

Syntax

```
library (library_name) {
```

```
    default_soft_error_rate (template_name) {
      index_1 ( … ) ;
      index_2 ( … ) ;
      values ( … ) ;
    }
  }
```

index_1 and index_2 Complex Attributes

The `index_1` attribute specifies the `altitude` index values at which the default cell SER is sampled in the library. The unit is defined by the library-level `altitude_unit` attribute.

The `index_2` attribute specifies the `frequency` index values at which the cell SER is sampled in the library. The values must be greater than or equal to zero. The unit is defined by the library-level `time_attribute` attribute, as the frequency unit is the reciprocal of the time unit.

values Complex Attribute

The `values` attribute specifies the default cell SER values with respect to the index values.

Example

```
default_soft_error_rate ( ser_temp ) {
  index_1 ( "1.6, 1.8" ) ; /* altitude 1.6 km, 1.8 km */
  index_2 ( "0.1, 0.2" ) ; /* frequency 100 MHz, 200 MHz */
  /* soft errors per 1 billion hours of operation */
  values ( "3.5, 3.6, 4.5, 4.6") ;
}
```

---

em_lut_template Group

The `em_lut_template` group is defined at the `library` group level.

Syntax

```
library (name_string) {
 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");
 }
}
```

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") ;
```

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

```
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 4 shows an example of the `fall_net_delay` group.

Example 4 fall_net_delay Group

```
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:

```
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 5 shows a `fall_transition_degradation` group.

Example 5 fall_transition_degradation Group

```
fall_transition_degradation(trans_deg) {
 index_1 ("1, 0, 2") ;
 index_2 ("0, 1, 2") ;
 values ("0.0, 0.8", "1.0, 1.8") ;
}
```

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 4 lists the predefined variables that can be used in an expression.

Table 4 Voltage-Level Variables for the input_voltage Group

CMOS or BiCMOS variable	Default value
VDD	5V
VSS	0V
VCC	5V

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 6 shows a collection of `input_voltage` groups.

Example 6 input_voltage Groups

```
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 (name_string) {
 fpga_isd (fpga_isd_name_string ) {
  ...  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 (name_string) {
 ...
 lu_table_template(name_string) {
  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
```

normalized_voltage Variable

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

```
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

```
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:

```
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 (name_string) {
 maxcap_lut_template (template_name_id) {
  ... 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

```
library (name_string) {
 maxtrans_lut_template (template_name_id) {
  ... 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) {
 ...
 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

```
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

```
 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 (name_string) {
 operating_conditions (name_string) {
  ... operating conditions description ...
 }
}
```

Simple Attributes

```
calc_mode : name_id ;
parameteri : float ;
process : float ;
process_label : "string" ;
temperature : float ;
tree_type : value_enum;
voltage : float ;
```

calc_mode Simple Attribute

An optional attribute, you can use `calc_mode` to specify an associated process mode.

Syntax

```
calc_mode : name_id ;
```

name

The name of the associated process mode.

parameter*i* Simple Attribute

Use this optional attribute to specify values for up to five user-defined variables.

Syntax

```
parameteri : value_float ; /* 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 : value_float ;
```

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 : value_float ;
```

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 : worse_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

```
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

```
library (name_string) {
 output_voltage(name_string) {
  vol : float | expression ;
  voh : float | expression ;
  vomin : float | expression ;
  vomax : float | expression ;
 }
  output_voltage (name_string) {
 ... output_voltage description ... ;
  }
}
```

The value for `vol`, `voh`, `vomin`, and `vomax` is a floating-point number or an expression. An expression allows you to define voltage levels as a percentage of VSS or VDD.

vol

The maximum output voltage generated to represent a logic 0.

voh

The minimum output voltage generated to represent a logic 1.

vomin

The minimum output voltage the pad can generate.

vomax

The maximum output voltage the pad can generate.

Table 5 lists the predefined variables you can use in an `output_voltage` expression attribute. Separate variables are defined for CMOS and BiCMOS.

Table 5 Voltage-Level Variables for the output_voltage Group

CMOS or BiCMOS variable	Default value
VDD	5V
VSS	0V
VCC	5V

The default values represent nominal operating conditions. These values fluctuate with the voltage range defined in the `operating_conditions` groups.

All voltage values are in the units you define with the `voltage_unit` attribute within the `library` group. Example 7 shows an example of an `output_voltage` group.

Example 7 output_voltage Group

```
output_voltage(GENERAL) {
 vol : 0.4 ;
 voh : 2.4 ;
 vomin : -0.3 ;
 vomax : VDD + 0.3 ;
}
```

part Group

You use a `part` group to describe a specific FPGA device. Use multiple `part` groups to describe multiple devices.

Syntax

```
library (name_string) {
```

```
 part(name_string) {
  ...device description...
 }
  part(name_string) {
  ...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_name_id ;
```

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_name_id, value_int ) ;
```

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_id", "name_2_id", ..."name_n_id" ) ;
```

“*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" ) ;
```

“*name_1*”, “*name_2*”, “*name_n*”

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(namestring) {
  ...
   speed_grade (namestring) {
   ...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

```
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 index_2 attribute is required in the power_lut_template group if the variable_2 attribute is present.

The index_3 complex attribute in the power_lut_template group specifies the breakpoints of the third dimension used to characterize cells for internal power within the library. You can overwrite the index_3 attribute in the internal_power group 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_3` attribute is required in the `power_lut_template` group if the `variable_3` attribute is present.

Example 8 shows four `power_lut_template` groups.

Example 8 Four power_lut_template Groups

```
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") ;
}
```

rise_net_delay Group

The `rise_net_delay` group is defined at the library level, as shown here:

```
library (name) {
 rise_net_delay(name) {
 ... rise net delay description ...
 }
 fall_net_delay (name){
 ... fall net delay description ...
 }
}
```

Complex Attributes

```
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 9 shows an example of the `rise_net_delay` group.

Example 9 rise_net_delay Group

```
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:

```
library (name) {
 rise_transition_degradation(name) {
 ... rise transition degradation description ...
 }
}
```

Complex Attributes

```
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 `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 10 shows an example of the `rise_transition_degradation` group.

Example 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") ;
}
```

See also fall_transition_degradation Group on page 61.

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 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 `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.

Syntax

```
library(library_name) {
   ...
   sensitization (sensitization_group_name) {
   ...
   }
}
```

Complex Attributes

```
pin_names
vector
```

pin_names Complex Attribute

The `pin_names` attribute specified at the library level defines a default 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.

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]…");
```

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" );
```

soft_error_rate_template Group

The `soft_error_rate_template` group defines the table template for the library-level `default_soft_error_rate` and the cell-level `soft_error_rate` groups. The template has two indexes, altitude and frequency. The template table can be one-dimensional or two-dimensional.

Syntax

```
library (library_name) {
  …
  soft_error_rate_template (temp_name) {
    variable_1 : [ altitude | frequency ] ;
    variable_2 : [ altitude | frequency ] ;
    index_1 ( … ) ;
    index_2 ( … ) ;
  }
}
```

variable_1 and variable_2 Complex Attributes

The template definition includes two one-dimensional variables, `variable_1` and `variable_2`, with values, `altitude` and `frequency`, respectively.

index_1 and index_2 Complex Attributes

The `index_1` attribute specifies the `altitude` values at which the cell soft error rate (SER) is sampled in the library. The unit is defined by the library-level `altitude_unit` attribute.

The `index_2` attribute specifies the `frequency` values at which the cell SER is sampled in the library. The values must be greater than or equal to zero. The unit is defined by the library-level `time_attribute` attribute, as the frequency unit is reciprocal of the time unit.

Example

```
soft_error_rate_template ( ser_temp ) {
  variable_1 : altitude ;
  variable_2 : frequency ;
  index_1 ( "1.6, 1.8" ) ; /* altitude 1.6 km, 1.8 km */
  index_2 ( "0.1, 0.2" ) ; /* frequency 100 MHz, 200 MHz */
}
```

---

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

```
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

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

name

Identifies the bus type.

Simple Attributes

```
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 11 illustrates a `type` group statement.

Example 11 type Group Description

```
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 12 Alternative type Group Descriptions

```
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

```
library (name) {
 user_parameters () {
 ... parameter descriptions...
```

```
  }
}
```

Simple Attributes

`parameter`*`i`*

parameter*i* 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

`parameter`*`i`* `:` *`value`*$_{float}$ `;`

value

A floating-point number representing a variable value.

Example

```
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 : value_float ;
```

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 : value_float ;
```

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 : value_float ;
```

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 : value_float ;
```

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 ( fanout_int,length_float,average_capacitance_float,\
   standard_deviation_float,number_of_nets_int ) ;
```

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

```
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) ;
 }
}
```

2

cell and model Group Description and Syntax

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.

cell Group

A `cell` group is defined within a `library` group, as shown here:

```
library (name_string) {
  cell (name_string) {
  ... cell description ...
  }
}
```

Attributes and Values

Example 13 lists alphabetically all the attributes and groups that you can define within a `cell` group.

Example 13 Attributes and Values for a cell Group

```
 /* Simple Attributes for cell group */

always_on : true | false ;
antenna_diode_type : power | ground | power_and_ground ;
area : float ;
auxiliary_pad_cell : true | false ;
base_name : cell_base_name_string ;builtin_clock_gating_integrate_cell:
 icg_type ;
bus_naming_style : "string" ;
cell_footprint : footprint_type_string ;
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 : name_id ;
em_temp_degradation_factor : value_float ;
interface_timing : true | false ;
io_type : name_id ;
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 ;
physical_variant_cells : "names-list" ;
power_cell_type : ;
preferred : true | false ;
retention_cell : retention_cell_style ;
single_bit_degenerate : string ;
/* black box, bus, and bundle cells only*/
slew_type : name_id ;
timing_model_type : "string" ;
use_for_size_only : true | false ;

/* Complex Attributes for cell Group */

pin_equal ("name_list_string") ;
```

```
pin_opposite ("name_list1string","name_list2string") ;
resource_usage (resource_nameid, number_of_resourcesid);

/* Group Statements for cell Group */

bundle (namestring)  { }
bus (namestring)  { }
clear_condition () {}
clock_condition () {}
dynamic_current () {}
ff (variable1string, variable2string)  { }
ff_bank (variable1string, variable2string, bitsinteger)  { }
generated_clock () {}
intrinsic_parasitic () {}
latch (variable1string, variable2string)  { }
latch_bank (variable1string, variable2string, bitsinteger) { }
leakage_current () {}
leakage_power ()  { }
lut (namestring)  { }
mode_definition () {}

pin (namestring | name_liststring)  { }
preset_condition () {}
retention_condition () {}
statetable ("input node names", "internal node names") { }
test_cell ()  { }
type (namestring)  { }
```

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 116.

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

```
base_name : "cell_base_name_id" ;
```

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 199.

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 6 lists some example values for the `clock_gating_integrated_cell` attribute.

Table 6 Some Values for the clock_gating_integrated_cell Attribute

Value of clock_gating_integrated_cell	Integrated cell must contain
latch_negedge	1. Latch-based gating logic 2. Logic appropriate for falling-edge-triggered registers
latch_posedge_postcontrol	1. Latch-based gating logic 2. Logic appropriate for rising-edge-triggered registers 3. Test control logic located after the latch
latch_negedge_precontrol	1. Latch-based gating logic 2. Logic appropriate for falling-edge-triggered registers 3. Test control logic located before the latch
none_posedge_control_obs	1. Latch-free gating logic 2. Logic appropriate for rising-edge-triggered registers 3. Test control logic (no latch) 4. Observability logic

For a complete listing of the values you can enter for the `clock_gating_integrated_cell` attribute, the corresponding circuitry that these values represent, and examples for each value, see Appendix A in the *Synopsys Liberty User Guide*.

For more details about the clock-gating integrated cells, see the “Modeling Power and Electromigration” chapter in the *Liberty User Guide, Vol. 1*.

Example

```
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 7. Setting some of the pin attributes, such as those for test and observability, is optional.

Table 7 Pin Attributes for Integrated Clock Gating Cells

Integrated cell pin name	Data direction	Required attribute
clock	in	clock_gate_clock_pin
enable	in	clock_gate_enable_pin
test_mode or scan_enable	in	clock_gate_test_pin
enable_clock	out	clock_gate_out_pin

See Chapter 3, pin Group Description and Syntax," for details about these pin attributes.

For more details about the `clock_gating_integrated_cell` attribute and the corresponding pin attributes, see the *Synopsys Liberty User Guide*.

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 8 lists the edge values and the corresponding setup and hold arcs.

Table 8 Values of the clock_gating_integrated_cell Attribute

Value of clock_gating_integrated_cell attribute	Setup arc	Hold arc
latch_posedge	rising	rising
latch_negedge	falling	falling
none_posedge	falling	rising
none_negedge	rising	falling

For details about setting timing for an integrated cell, see the "Modeling Power and Electromigration" chapter in the *Synopsys Liberty User Guide*.

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 : value_Boolean ;
```

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 : value_Boolean ;
```

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

```
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;
     …
}
```

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 ;
```

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

`fpga_cell_type :` $value_{enum}$ `;`

value

Valid values are `rising_edge_clock_cell` and `falling_edge_clock_cell`.

Example

```
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

```
fpga_isd : fpga_isd_name_id ;
```

fpga_isd_name

The name of the library-level `fpga_isd` group.

Example

```
fpga_isd : part_cell_isd ;
```

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 : name_id ;
```

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) {
  ...
  pin(PAD) {
    direction : input;
    is_pad : true;
  }
}

cell (POWER_PAD_CELL) {
  …
  pad_cell : true ;
  pg_pin (my_pg_pin) {
    is_pad : true ;
    …
  }
  pin (my_pin) {
    …
  }
}
```

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

```
is_clock_gating_cell : true | false ;
```

Example

```
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.

See clock_gate_enable_pin Simple Attribute on page 234 for information about designating clock and enable ports on clock gates.

is_clock_isolation_cell Simple 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. For information about pin-level attributes of the clock-isolation cell, see clock_isolation_cell_clock_pin Simple Attribute on page 235 and isolation_cell_enable_pin Simple Attribute on page 259.

Syntax

```
is_clock_isolation_cell : true | false ;
```

Example

```
is_clock_isolation_cell : true ;
```

is_isolation_cell Simple Attribute

The cell-level `is_isolation_cell` attribute specifies that a cell is an isolation cell. The pin-level `isolation_cell_enable_pin` attribute specifies the enable input pin for the isolation cell.

Syntax

```
is_isolation_cell : true | false ;
```

Example

```
is_isolation_cell : true ;
```

is_level_shifter Simple Attribute

The cell-level `is_level_shifter` attribute specifies that a cell is a level shifter cell. The pin-level `level_shifter_enable_pin` attribute specifies the enable input pin for the level shifter cell.

Syntax

```
is_level_shifter : Boolean expression ;
```

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

```
level_shifter_type : level_shifter_type_value ;
```

Example

```
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

```
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

```
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

```
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

```
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.

The only type Synopsys supports is clock, which identifies a pad cell as a clock driver.

Syntax

```
pad_type : value ;
```

Example

```
pad_type : clock;
```

physical_variant_cells Simple Attribute

Use the `physical_variant_cells` attribute to specify a list of physical variant cells of a master cell.

Physical variant cells have different physical layouts but share the same logic Liberty models. For example, a design can have multiple cells that are variants of a master cell with only difference in the mask color of their pins. In low technology nodes, the characterization information of these cells does not vary and you can use the same Liberty model for the master cell and all its variants.

Syntax

```
physical_variant_cells : "names-list";
```

Example

```
physical_variant_cells : "ABC_1 ABC_2";
```

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 118.

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.

The Design Compiler tool uses this link to group narrower black boxes into wider black boxes or to group wider black boxes into narrower ones.

Syntax

```
single_bit_degenerate : "cell_name_id";
```

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" ;
  bus (D) {
    bus_type : bus4 ;
    direction : input ;
    ...
    timing ( ) {
```

```
      ...
      ...
    }
  }
}
```

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 : "name_id ";
```

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_name_id" ;
```

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 : "name_id ";
```

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 : value_Boolean ;
```

value

Valid values are true and false.

Example

```
library(lib1){
  cell(cell1){
    area : 14 ;
    use_for_size_only : true ;
    pin(A){
      ...
    }
    ...
```

```
    }
}
```

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 ($resource\_name_{id}$, $number\_of\_resources_{int}$) ;

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 14 shows cell definitions that include some of the CMOS cell attributes described so far.

Example 14 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 () {...}
    }
  }
} /* End of Library */
```

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

```
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

```
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

```
defect_type : enum (open, short, open_and_short);
```

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") ;
  }
   ...
...
```

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 (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") ;
  }
   ...
...
```

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

```
library (name_id) {
  cell (name_id) {
    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.

```
capacitance : float ;
direction : input | output | inout |internal ;
function : "Boolean" ;
```

Complex Attribute

```
members (name_id) ;
```

Group Statement

All `pin` group statements are valid in a `bundle` group.

```
pin (name_id | name_list_id) { }
```

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.

To review all the `pin` group attributes, see Chapter 3, pin Group Description and Syntax." The `capacitance`, `direction`, and `function` attributes are frequently used in `bundle` groups.

capacitance Simple Attribute

Use the `capacitance` attribute to define the load of an input, output, inout, or internal pin.

For a description of the `fall_capacitance` attribute, see fall_capacitance Simple Attribute. For a description of the `rise_capacitance` attribute, see rise_capacitance Simple Attribute.

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 15).

Syntax

```
direction : input | output | inout | internal ;
```

Example

In a `bundle` group, the direction of all pins must be the same. Example 15 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 15 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 9 lists the Boolean operators valid in a function statement.

Table 9 Valid Boolean Operators

Operator	Description
'	invert previous expression

Operator	Description
!	invert following expression
^	logical XOR
*	logical AND
&	logical AND
space	logical AND
+	logical OR
|	logical OR
1	signal tied to logic 1
0	signal tied to logic 0

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 OR B:

```
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.

```
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.

```
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_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 16 shows how to define a `bundle` group in a cell with a multibit latch.

Example 16 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(name_string) {
  pin (name_string) {
  ... 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

```
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 15 on page 132 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 17) as the value for the `bus_type` attribute in the `bus` group in the same library.

Example 17 shows a `bus` group in a `cell` group.

Example 17 Bused Pins

```
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.

```
bus_type : name ;
scan_start_pin : pin_name ;
scan_pin_inverted : true | false ;
```

Group Statement

```
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

```
bus_type : name ;
```

name

Define this name in the applicable `type` group in the library, as shown in the example in bus Group.

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.

For a description of the `fall_capacitance` attribute, see fall_capacitance Simple Attribute. For a description of the `rise_capacitance` attribute, see rise_capacitance Simple Attribute.

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 the example inExample Bus Description–Logic Library 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 the example inExample Bus Description–Logic Library.

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[O:2]) {}
```

The next example shows a `pin` group within a `bus` group that defines a new `capacitance` attribute value for a single pin number.

```
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–Logic Library

Example 18 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 18 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_capacitance
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 43.

---

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() {
        ... switching_group description...
        }
      }
    }
  }
}
```

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 41.

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 156.

```
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 */
      }
    }
  }
}
```

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", \ /* 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 */
}
```

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 `total_output_net_capacitance` variables is applied to the group. If a dense table is applied, the order of `total_output_net_capacitance` variables must map to the order of values in the `related_outputs` attribute.
- Data in this group is represented as a sparse cross table, if the `index_output` attribute is defined in the group.
- Data in this group is represented as a sparse diagonal table, if no `index_output` attribute is defined in the group and a template with exact one `total_output_net_capacitance` variable is applied to the group.

```
library (name) {
  cell (name) {
    dynamic_current () {
      switching_group() {
        pg_current () {}
          vector () {
          ...
          }
        }
      }
    }
  }
}
```

Simple Attributes

```
index_1 (float);
index_2 (float);
index_3 (float);
index_4 (float);
index_output : output_pin_name>;
reference_time: float;
```

```
values ("float, ...");
```

index_1, index_, index_3, and index_4 Simple Attributes

The index attributes specify values for variables specified in the `pg_current_template`. The index value for `input_net_transition` or `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 `related_outputs`, the table must be dense.
- If two outputs are specified in `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 218.

Syntax

```
library (name_string) {
   cell (name_string) {
      ff (variable1_string, variable2_string) {
         ... 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 161 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 10 shows the valid variable values for the clear_preset_var1 simple attribute.

Table 10 Valid Values for the clear_preset_var1 and clear_preset_var2 Attributes

Variable values	Equivalence
L	0

Variable values	Equivalence
H	1
N	No change[1]
T	Toggle the current value from 1 to 0, 0 to 1, or X to X[2]
X	Unknown[3]

Single-Stage Flip-Flop on page 161 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 161 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 161 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.

1. *Use these values to generate VHDL models.*
2. *Use these values to generate VHDL models.*
3. *Use these values to generate VHDL models.*

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'" ;
```

next_state Simple Attribute

Syntax

```
next_state : "Boolean expression" ;
```

The following example shows an `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` 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:

```
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

```
preset : "Boolean expression" ;
```

Example

```
preset : "PD'" ;
```

The next section, “Single-Stage Flip-Flop,” contains more information about the `preset` attribute.

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:

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

Table 11 shows the functions of the attributes in the `ff` group for a single-stage flip-flop.

Table 11 Function Table for a Single-Stage Flip-Flop

clocked_on	clear	preset	variable1	variable2
active edge	inactive	inactive	next_state	!next_state
--	active	inactive	0	1
--	inactive	active	1	0
--	active	active	clear_preset_var1	clear_preset_var2

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. See Table 11 for the valid variable values.

If the `clear` and `preset` attributes are both included in the group, either `clear_preset_var1`, `clear_preset_var2`, or both must be defined. Conversely, if either `clear_preset_var1`, or `clear_preset_var2`, or both are included, both `clear` and `preset` must be defined.

The flip-flop cell is activated whenever the value of `clear`, `preset`, `clocked_on`, or `clocked_on_also` changes.

Example 19 is an `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 19 Single-Stage D Flip-Flop

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

Example 20 is an `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 20 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 21 is an ff group for a D flip-flop with synchronous negative clear.

Example 21 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 12 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 12 Function Table for a Master-Slave Flip-Flop

Variable	Functions				
clear	active	active	inactive	inactive	inactive
preset	active	inactive	active	inactive	inactive
internal1	clear_preset_var1	0	1	next_state	
internal2	clear_preset_var2	1	0	!next_state	
variable1	clear_preset_var1	0	1		internal1
variable2	clear_preset_var2	1	0		internal2
active edge				clocked_on	clocked_on_also

Example 22 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 22 Master-Slave D Flip-Flop

```
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 218.

The syntax for the `ff_bank` group is similar to that of the `ff` group.

Syntax

```
library (name_string) {
  cell (name_string) {
    ff_bank (variable1_string, variable2_string, bits_integer) {
      ... 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 23 on page 168 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 161 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 161 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 161 for more information about the `clear_preset_var1` attribute, including its function and values.

Table 13 shows the valid variable values for the `clear_preset_var1` attribute.

Table 13 Valid Values for the clear_preset_var1 and clear_preset_var2 Attributes

Variable values	Equivalence
L	0
H	1
N	No change[4]
T	Toggle the current value from 1 to 0, 0 to 1, or X to X[5]
X	Unknown[6]

4. *Use these values to generate VHDL models.*
5. *Use these values to generate VHDL models.*
6. *Use these values to generate VHDL models.*

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. Table 13 shows the valid variable values for the `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 161 for more information about the `clear_preset_var1` attribute, including its function and values.

Multibit Flip-Flop

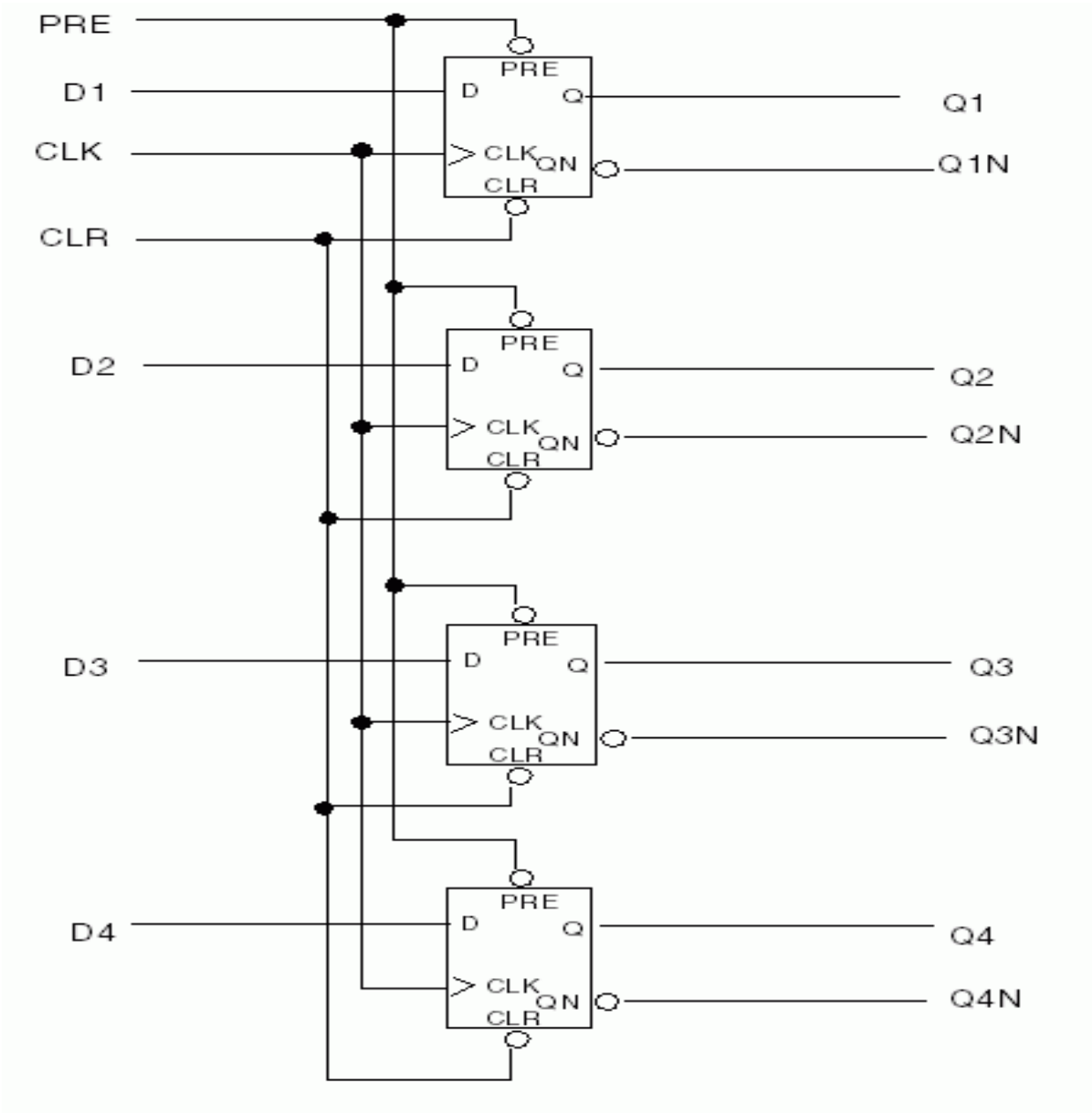
The *bits* value in the `ff_bank` definition is the number of bits in this multibit cell.

Syntax

```
library (name_string) {
  cell (name_string) {
    ff_bank (variable1_string, variable2_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 1 and Example 23.

Figure 1 Multibit Register



Example 23 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 ") ;
      }
    }
  }
  pin (CLR) {
    direction : input ;
    capacitance : 0 ;
    timing() {
      related_pin     : "CLK" ;
      timing_type     : recovery_rising ;
      cell_rise(scalar) {
        values (" 1.0 ") ;
      }
      cell_fall(scalar) {
        values (" 1.0 ") ;
      }
    }
  }
  pin (PRE) {
    direction : input ;
    capacitance : 0 ;
    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 ") ;
    }
  }

  timing() {
    related_pin     : "PRE" ;
    timing_type     : preset ;
    timing_sense    : negative_unate ;
    cell_rise(scalar) {
      values (" 1.0 ") ;
    }
  }
  timing() {
    related_pin     : "CLR" ;
    timing_type     : clear ;
    timing_sense    : positive_unate ;
    cell_fall(scalar) {
      values (" 1.0 ") ;
    }
  }
}
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 (name_id) {
```

```
  fpga_condition (condition_group_name_id) {
    fpga_condition_value (condition_name_id){
    ...
    }
  }
}
```

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 (name_id) {
    fpga_condition_value (condition_name_id){
      fpga_arc_condition : condition_Boolean ;
    }
  }
}
```

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 24 shows a generated clock description.

Example 24 Description of a Generated Clock

```
cell(acell) {
  ...
  generated_clock(genclk1) {
    clock_pin : clk1;
    master_pin : clk;
    divided_by : 2;
    invert : true;
  }
  generated_clock(genclk2) {
    clock_pin : clk1;
    master_pin : clk;
```

```
    multiplied_by : 2;
    duty_cycle : 50.0;
  }
  generated_clock(genclk3) {
    clock_pin : clk1;
    master_pin : clk;
    edges(1, 3, 5);
    shifts(5.0, -5.0, 0.0);
  }
  ....
  pin(clk) {
    direction : input;
    clock : true;
    capacitance : 0.1;
  }
  pin(clk1) {
    direction : input;
    clock : true;
    capacitance : 0.1;
  }
}
```

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

```
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" );
}
```

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

```
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 218.

```
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 14 shows the valid values for the `clear_preset_var1` and `clear_preset_var2` attributes.

Table 14 Valid Values for the clear_preset_var1 and clear_preset_var2 Attributes

Variable values	Equivalence
L	0
H	1
N	No change[7]
T	Toggle the current value from 1 to 0, 0 to 1, or X to X[8]
X	Unknown[9]

See Single-Stage Flip-Flop on page 161 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(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 `data_in` and `enable` attributes are optional, but if you use one of them, you must also use the other.

7. *Use these values to generate VHDL models.*
8. *Use these values to generate VHDL models.*
9. *Use these values to generate VHDL models.*

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 : "value_Boolean " ;
```

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 15 shows the functions of the attributes in the `latch` group.

Table 15 Function Table for a latch Group

enable	clear	preset	variable1	variable2
active	inactive	inactive	data_in	!data_in
--	active	inactive	0	1
--	inactive	active	1	0
--	active	active	clear_preset_var1	clear_preset_var2

Example 25 shows a `latch` group for a D latch with active-high enable and negative clear.

Example 25 D Latch With Active-High Enable and Negative Clear

```
latch(IQ, IQN) {
  enable : "G" ;
  data_in : "D" ;
  clear : "CD'" ;
}
```

Example 26 shows a `latch` group for an SR latch. The `enable` and `data_in` attributes are not required for an SR latch.

Example 26 SR Latch

```
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 218.

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 183).

Syntax

```
library (name_string) {
  cell (name_string) {
     latch_bank(variable1_string,variable2_string,
bits_integer){
       ... multibit latch register description ...
     }
  }
}
```

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

Simple Attributes

```
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 27 shows a `latch_bank` group for a multibit register containing four rising-edge-triggered D latches.

Example 27 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 the table in latch Group for the valid values for the `clear_preset_var1` and `clear_preset_var2` attributes.

See Single-Stage Flip-Flop on page 161 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.

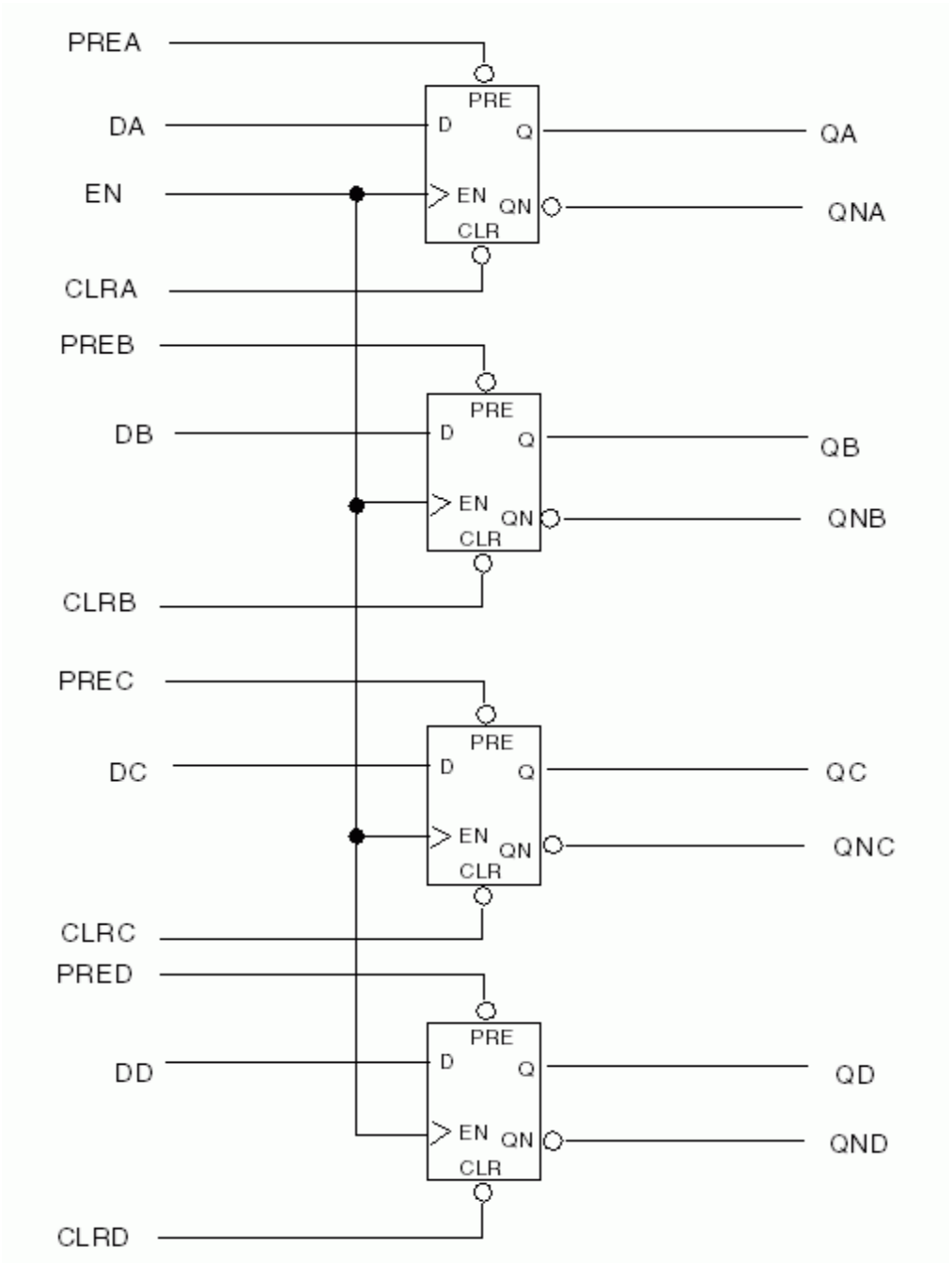
```
preset : "PD'" ;
```

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 and Example 28 show a multibit register containing four high-enable D latches with the `clear` attribute.

Figure 2 Multibit Register With Latches



Example 28 Multibit Register With Four D Latches

```
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 `pg_current` group and specify the leakage current value. Otherwise, specify the value in the `pg_current` group. Current

conservation is applied for each `leakage_current` group. The `value` attribute specifies the absolute value of leakage current on a single power and ground pin.

Syntax

```
value : value_float ;
```

value

A floating-point number representing the leakage current.

mode Complex Attribute

The `mode` attribute specifies the current mode of operation of the cell. Use this attribute in the `leakage_current` group to define the leakage current in the specified mode.

Syntax

```
mode (mode_name, mode_value) ;
```

Example

```
mode (rw, read) ;
```

pg_current Group

Use this group to specify a power or ground pin where leakage current is to be measured.

Syntax

```
cell(cell_name) {
  ...
  leakage_current() {
    when : boolean expression;
    pg_current(pg_pin_name) {
      value : float;
    }
```

pg_pin_name

Specifies the power or ground pin where the leakage current is to be measured.

Simple Attribute

```
value
```

Use this attribute in the `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 : value_float ;
```

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 103.

Syntax

```
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

```
power_level : "name" ;
```

name

Name of the power rail defined in the power supply group.

Example

```
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_pin_id;
```

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 16 lists the Boolean operators valid in a `when` statement.

Table 16 Valid Boolean Operators

Operator	Description
'	invert previous expression
!	invert following expression
^	logical XOR
*	logical AND

Operator	Description
&	logical AND
space	logical AND
+	logical OR
|	logical OR
1	signal tied to logic 1
0	signal tied to logic 0

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.

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_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

```
pg_setting (psd_name, psv_name);
```

Example

```
pg_setting (psd1, psv1);
```

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.

Syntax

```
cell (cell_name)
  …
  pg_setting_definition(psd_name) {
    …
  }
}

psd_name
```

Name of the pg_setting_definition group. The system power states must be mutually exclusive.

Example

```
pg_setting_definition(psd1) {
  …
}
```

Simple Attributes

```
default_pg_setting
illegal_transition_if_undefined
```

Groups

```
pg_setting_value
pg_setting_transition
mode_definition
```

default_pg_setting Simple 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.

Syntax

```
default_pg_setting : psv_name ;
```

psv_name is the name of a pg_setting_value group specified in the same pg_setting_definition group.

Example

```
default_pg_setting : "psv1" ;
```

illegal_transition_if_undefined Simple 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

```
illegal_transition_if_undefined : true | false ;
```

Example

```
illegal_transition_if_undefined : true ;
```

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 ;
```

permit_power_down Simple Attribute

The `permit_power_down` attribute identifies the power pin of an isolation cell, that can be powered down.

Syntax

```
permit_power_down : boolean_expression ;
```

Boolean expression

Valid values are `true` or `false`. Setting the attribute value to `true` allows the power pin to be powered down in the isolation mode. The default is `false`.

Example

```
permit_power_down : true ;
```

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 `deeppwell`. The `pwell` and `nwell` values specify regular wells, and the `deeppwell` 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" ;
    related_ bias_pin : "vpw vnw";
    power_down_function : "vdd' + vss + vnw' + vpw ";
    …
  }
 } /* end of cell group */
} /* end of library group*/
```

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.

```
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";
  …
}
   } /* End of cell group */
}/* End of library group */
```

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:

```
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.

```
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.

```
pg_pin(VDDS){
   voltage_name : VDDS;
   direction : input;
   pg_type : pwell | nwell | deepnwell | deeppwell;
   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.

```
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.

```
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

```
is_insulated : Boolean expression ;
```

Boolean expression

Valid values are true and false.

Example

```
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.

Syntax

```
required_condition : "Boolean_expression" ;
```

---

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.

```
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

```
power_down_function : "Boolean_expression" ;
```

Example

```
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

```
required_condition : "Boolean_expression" ;
```

Example

```
required_condition : "!CK * !RET" ;
```

soft_error_rate Group

The `soft_error_rate` group is a lookup table that specifies the soft error rate (SER) for the cell. The table values must be greater than or equal to zero. You can specify this table for all cell types. The cell-level values override the library-level `default_soft_error_rate` values.The table can be one-dimensional or two-dimensional.

Syntax

```
cell (cell_name) {
  soft_error_rate (template_name) {
    index_1 ( … ) ;
    index_2 ( … ) ;
    values ( … ) ;
  }
}
```

index_1 and index_2 Complex Attributes

The `index_1` attribute specifies the `altitude` index values at which the default cell SER is sampled. The unit is defined by the library-level `altitude_unit` attribute.

The `index_2` attribute specifies the `frequency` index values at which the cell SER is sampled. The values must be greater than or equal to zero. The unit is defined by the

library-level `time_attribute` attribute, as the frequency unit is the reciprocal of the time unit.

values Complex Attribute

The `values` attribute specifies the cell SER values with respect to the index values.

Example

```
default_soft_error_rate ( ser_temp ) {
  index_1 ( "1.6, 1.8" ) ; /* altitude 1.6 km, 1.8 km */
  index_2 ( "0.1, 0.2" ) ; /* frequency 100 MHz, 200 MHz */
  /* soft errors per 1 billion hours of operation */
  values ( "3.55, 3.65, 4.55, 4.65") ;
}
```

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.

A sequential library cell can have only one state table. For a complete description of a state table and the `statetable` group format, see the “Describing Sequential Cells With the Statetable Format” section of the “Defining Sequential Cells” chapter in the *Synopsys Liberty User Guide*.

Note:

In the `statetable` group, `table` is a keyword.

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";
}
```

The following example shows a state table for a JK flip-flop with an active-low, direct-clear, and negative-edge clock:

Example

```
statetable ("J  K  CN  CD",  "IQ" ) {
  table:    "-  -  -   L    : - :  L,\
```

```
                -   -   ~F   H     : - :   N,\
                L   L   F    H    :L/H:   L/H,\
                H   L   F    H     : - :   H,\
                L   H   F    H     : - :   L,\
                H   H   F    H    :L/H:   H/L" ;
}
```

test_cell Group

The `test_cell` group is in a `cell` group or `model` group. It models only the nontest behavior of a scan cell, which is described by an `ff`, `ff_bank`, `latch`, `latch_bank` or `statetable` statement and `pin` function attributes.

Syntax

```
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 (variable1_id, variable2_id) { }
ff_bank (variable1_id, variable2_string, bits_int) { }
latch (variable1_id, variable2_id) { }
latch_bank (variable1_id, variable2_id, bits_int){}
pin (name_id)  { }
statetable ("input node names", "internal node names") { }
```

For more information about these groups, see the “Defining Sequential Cells” chapter in the *Synopsys Liberty User Guide*.

ff Group

For a discussion of the `ff` group syntax, see ff Group on page 157.

ff_bank Group

For a discussion of the `ff_bank` group syntax, see ff_bank Group on page 164.

latch Group

For a discussion of the `latch` group syntax, see latch Group on page 183.

latch_bank Group

For a discussion of the `latch_bank` group syntax, see latch_bank Group on page 187.

pin Group in a test_cell Group

Both test `pin` and nontest `pin` groups appear in `pin` groups within a `test_cell` group, as shown:

```
library (name_string) {
   cell (name_string) {
      test_cell (name_string) {
         pin (name_string | name_list_string) {
         ... 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_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_output_only : true | false ;
```

Group

```
statetable() { }
```

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 132.

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

```
test_output_only : true | false ;
```

Example

```
test_output_only : true ;
```

statetable Group

For a discussion of the `statetable` group syntax, see statetable Group on page 217.

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

```
cell (name_string) {
  type (name_string) {
     ... 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

```
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

```
bit_from : value_int ;
```

value

Indicates the member number assigned to the MSB of successive array members. The default is 0.

Example

```
bit_from : 0 ;
```

bit_to Simple Attribute

The `bit_to` attribute specifies the member number assigned to the least significant bit (LSB) of successive array members.

Syntax

```
bit_to : value_int ;
```

value

Indicates the member number assigned to the LSB of successive array members. The default is 0.

Example

```
bit_to : 3 ;
```

bit_width Simple Attribute

The `bit_width` attribute specifies the integer that designates the number of bus members.

Syntax

```
bit_width : value_int ;
```

value

Designates the number of bus members. The default is 1.

Example

```
bit_width : 4 ;
```

data_type Simple Attribute

Only the `bit` data type is supported.

Example

```
data_type : 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 29 illustrates a `type` group statement in a cell.

Example 29 type Group Within a Cell

```
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 (name_string) {
  model (name_string) {
  ... 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 99.

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 and there is no active logic between these two ports.

Syntax

```
short ("name_list_string") ;
```

Example

```
short(b, y);
```

Example 30 shows how to use a `short` attribute in a `model` group.

Example 30 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

Syntax of a pin Group in a cell or bus 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 31 lists alphabetically a sampling of the attributes and groups that you can define within a `pin` group.

Example 31 Attributes and Values in a pin Group

```
  /* Simple Attributes in a pin Group */
alive_during_partial_power_down : true | false ;
```

```
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 ;
clamp_0_function : "Boolean expression" ;
clamp_1_function : "Boolean expression" ;
clamp_latch_function : "Boolean expression" ;
clamp_z_function : "Boolean expression" ;
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 | sao1 ;
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 : "name_string | name_list" ;
input_signal_level : string ;
input_voltage : string ;
internal_node : name_string ; /* Required in statetable cells */
inverted_output : true | false ;/* Required in statetable cells */
is_pad : true | false ;
is_unconnected : true | false ;
max_capacitance : float ;
max_fanout : float ;
max_input_delta_overdrive_high : float ;
max_input_delta_underdrive_high : 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" ;

  /* Complex Attributes in a pin Group */

fall_capacitance_range ( float, float) ;
rise_capacitance_range ( float, float) ;

  /* Group Statements in a pin Group */

electromigration () { }
input_ccb (string) { }
internal_power ()  { }
max_trans () { }
min_pulse_width ()  { }
minimum_period ()  { }
output_ccb (string) { }
timing ()  { }
tlatch () {}
```

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.

Syntax

```
alive_during_partial_power_down : boolean_expression ;
```

Boolean expression

Valid values are `true` or `false`. The default is `false`.

Example

```
alive_during_partial_power_down : true ;
```

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 `antenna_diode_related_ground_pins` attribute specifies the related ground pin of the cell. Apply the `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 `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.

Syntax

```
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 `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.

For a cell with a built-in antenna-diode pin or port, 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.

Syntax

```
antenna_diode_related_power_pins : "power_pin1 power_pin2" ;
```

Example

```
antenna_diode_related_power_pins : "VDD1 VDD2" ;
```

antenna_diode_type Simple Attribute

The `antenna_diode_type` attribute specifies the type of pin in a macro cell. Valid values are `power`, `ground`, and `power_and_ground`.

Note:

You can specify the pin-level `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 : 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 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

```
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

```
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

```
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

```
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

```
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

```
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 236.

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 247.

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

```
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

```
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 241.

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.

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 17 lists the driver types, their signal mappings, and the applicable pin types.

Table 17 Pin Driver Types

Driver type	Signal mapping	Pin
pull_up	01Z->01H	in, out
pull_down	01Z->01L	in, out
open_drain	01Z->0ZZ	out
open_source	01Z->0Z1Z	out
bus_hold	01Z->01S	inout
resistive	01Z->LHZ	out
resistive_0	01Z->0HZ	out
resistive_1	01Z->L1Z	out

Keep the following concepts in mind when interpreting Table 17.

- 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 17, the `pull_up`, `pull_down`, and `bus_hold` driver types define a resolution scheme. The remaining driver types define transformations.

Example 32 describes an output pin with a pull-up resistor and the bidirectional pin on a `bus_hold` cell.

Example 32 Pin Driver Type Specifications

```
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 ;
  }
}
```

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:

```
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.

```
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 18 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 18 Programmable Driver Type Functions

Programmable driver type	Applicable on pin types
`pull_up_function`	Input, output and inout
`pull_down_function`	Input, output and inout
`bus_hold_function`	Inout
`open_drain_function`	Output and inout
`open_source_function`	Output and inout
`resistive_function`	Output and inout
`resistive_0_function`	Output and inout
`resistive_1_function`	Output and inout

Programmable Driver Type 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 107.

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 108.

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

```
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.

```
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

```
fall_current_slope_after_threshold : value_float ;
```

value

A floating-point number that represents the change in current.

Example

```
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

```
fall_current_slope_before_threshold : value_float ;
```

value

A floating-point number that represents the change in current.

Example

```
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

```
fall_time_after_threshold : value_float ;
```

value

A floating-point number that represents the time interval.

Example

```
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

```
fall_time_before_threshold : value_float ;
```

value

A floating-point number that represents the time interval.

Example

```
fall_time_before_threshold : 0.55 ;
```

fanout_load Simple Attribute

The `fanout_load` attribute gives the internal fanout load for an input pin.

Syntax

```
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

```
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.

The `fault_model` attribute, which is used only by DFT Compiler, is optional. If you enter a `fault_model` attribute, use the `complementary_pin` attribute to designate the inverted input pin associated with the noninverting pin.

For details on the `complementary_pin` attribute, see complementary_pin Simple Attribute on page 236.

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 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

```
cell (diff_buffer) {
  ...
  pin (A) {  /* noninverting pin /
    direction : input ;
    complementary_pin : ("DiffA")
    fault_model : "1x" ;
  }
}
```

Pin A (noninverting pin)	DiffA (complementary_pin)	Resulting signal pin value
1	0	1
0	1	0
0	0	1
1	1	x

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 19 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 19 Valid Boolean Operators

Operator	Description
'	invert previous expression
!	invert following expression
^	logical XOR
*	logical AND
&	logical AND
space	logical AND
+	logical OR
|	logical OR
1	signal tied to logic 1
0	signal tied to logic 0

The following example describes pin Q with the function A OR B.

Example

```
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:

```
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.

```
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 129 and bus Group on page 136. 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

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

the function for Z[0] is interpreted as

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

and the function for Z[1] is interpreted as

```
function : "(A[1] & B[1])" ;
```

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

```
function : "(A[1] & B)" ;
```

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 vil and vol voltage ratings are actual transition points. When the `hysteresis` attribute is omitted, the value is assumed to be false and no hysteresis occurs.

Example

```
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

```
illegal_clamp_condition : "Boolean expression" ;
```

Example

```
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

```
input_map : name_id ;
```

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

```
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

```
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

```
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

```
input_threshold_pct_fall : trip_point_float ;
```

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. You can specify this attribute at the library level to set a default for all the pins.

Syntax

```
input_threshold_pct_rise : trip_point_float ;
```

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 : name_id ; ;
output_voltage : name_id ; ;
```

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_name_id ;
```

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

```
inverted_output : Boolean expression ;
```

Example

```
inverted_output : true
```

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:

```
cell (cell_name) {
  ...
  pin (pin_name) {
    is _analog: true | false ;
    ...
  }
}
```

Example

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

```
pin(Analog) {
  direction : input;
  capacitance : 1.0 ;
  is_analog : 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.

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 ;
```

is_pad Simple Attribute

The `is_pad` attribute indicates which pin represents the pad. The valid values are `true` and `false`. You can also specify the `is_pad` attribute on PG pins.

Syntax

```
is_pad : Boolean expression ;
```

This attribute must be used on at least one pin with a `pad_cell` attribute.

Example

```
cell(INBUF) {
  ...
  pad_cell : true ;
  ...
  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 (feedback_pin_name) {
    is_pll_feedback_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;
  }
  …
}
```

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.

Syntax

```
cell (cell_name) {
  is_pll_cell : true;
  pin (ref_pin_name) {
    is_pll_reference_pin : true;
    direction : output;
    …
```

```
  }
  …
  pin (output_pin_name) {
    is_pll_output_pin : true;
    direction : output;
    …
  }
}
```

Example

```
cell(my_pll) {
  is_pll_cell : true;

  pin( REFCLK ) {
direction : input;
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

```
is_unbuffered : Boolean expression ;
```

Boolean expression

Valid values are `true` and `false`.

is_unconnected Simple Attribute

The `is_unconnected` attribute specifies a pin as internally not connected, which means that the pin is not part of a feedthrough path and it does not have the `related_power_pin` or the `related_ground_pin` attribute. You can also define this attribute under the `bus` or `bundle` group.

Syntax

```
is_unconnected : true | false ;
```

Example

```
is_unconnected : true ;
```

isolation_cell_data_pin Simple Attribute

The `isolation_cell_data_pin` attribute identifies the data pin of any isolation cell.

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 ;
```

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. For more information about isolation cells, see is_isolation_cell Simple Attribute on page 113.

Syntax

```
isolation_cell_enable_pin : boolean_expression ;
```

Boolean expression

Valid values are `true` and `false`.

Example

```
isolation_cell_enable_pin : 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.

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.

For more information about level-shifter cells, see is_level_shifter Simple Attribute on page 114.

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.

For more information about level-shifter cells, see is_level_shifter Simple Attribute on page 114.

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 117.

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_input_delta_overdrive_high Simple Attribute

The `max_input_delta_overdrive_high` and `max_input_delta_underdrive_high` attributes specify the allowed overdrive and underdrive delta voltage values of the signal pin with respect to the `related_power_pin` voltage. The attribute value units are taken from the library-level `voltage_unit` attribute.

You can specify this attribute only for input and inout signal pins of macro cells, memory cell, switch cells, level-shifter cells, and pad cells.

Syntax

```
max_input_delta_overdrive_high : value ;
max_input_delta_underdrive_high : value ;
```

Example

```
max_input_delta_overdrive_high : 0.3 ;
max_input_delta_underdrive_high : 0.2 ;
```

max_input_delta_underdrive_high Simple Attribute

See max_input_delta_overdrive_high Simple Attribute.

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.

Syntax

```
min_fanout : value_float ;
```

value

A floating-point number that represents the minimum number of fanouts the pin can drive. There are no fixed units for `min_fanout`. Typical units are standard loads or pin count.

Example

In the following example, pin X can drive a fanout load of no less than 3.0.

```
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

```
min_period : value_float ;
```

value

A floating-point number indicating a time unit.

Example

```
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

```
min_pulse_width_high : value_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.

```
pin(CLK) {
   direction : input ;
   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

```
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

```
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

```
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.

The example in the Multibit Flip-Flop shows a `nextstate_type` attribute in a `bundle` group.

output_signal_level Simple Attribute

See input_signal_level Simple Attribute on page 252.

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 325.

output_signal_level_low Simple Attribute

See output_signal_level_high Simple Attribute on page 266.

output_voltage Simple Attribute

See input_voltage Simple Attribute on page 253.

pg_function Simple Attribute

The `pg_function` attribute models the logical function of a virtual or derived power and ground (PG) pin as a Boolean expression involving the cells input signal pins, internal signal pins, PG pins. The attribute Boolean expression is checked during library compile to ensure that only one `pg_pin` is always active at this virtual or derived PG pin. If more than one `pg_pin` is found to be active at the virtual or the derived `pg_pin` output, the `read_lib` command generates an error.

Syntax

```
pg_function : "VDD1 * !(en1 & en2 & sleep) + VDD2 * (en1 & en2 & sleep)";
```

Example

```
pg_function : "((VDD1 * EN) + (VDD2 * !EN))";
```

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).

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.

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.

```
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

```
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

```
pulling_resistance : resistance value ;
```

resistance value

The resistive strength of the pull-up or pull-down device.

Example

```
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

```
pulse_clock :  pulse_type_enum ;
```

pulse_type

The valid values are `rise_triggered_high_pulse`, `rise_triggered_low_pulse`, `fall_triggered_high_pulse`, and `fall_triggered_low_pulse`.

Example

```
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 269.

Syntax

```
related_power_pin :  pg_pin_name_id ;
```

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.

```
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

`rise_current_slope_after_threshold : `*`value`*`float` ;

value

A negative floating-point number that represents the change in current.

Example

```
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

`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 : value_float ;
```

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 : value_float ;
```

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_point_value ;
```

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_point_value ;
```

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_point_value ;
```

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

```
slew_upper_threshold_pct_rise : trip_point_value ;
```

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

```
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

```
state_function : "Boolean expression" ;
```

Example

```
state_function : "EN*X" ;
```

For a list of Boolean operators, see the table in function Simple Attribute.

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

```
std_cell_main_rail : true | false ;
```

Example

```
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 : value_Boolean ;
```

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 248. 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 the table in function Simple Attribute.

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 282.

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 117.

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) ;
```

value_1, *value_2*

Positive floating-point numbers that specify the range of values.

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 () { }
min_pulse_width () { }
minimum_period () { }
output_ccb (string) {}
output_signal_swing () {}
pin_capacitance() { }
timing ()  { }
tlatch () { }
```

ccsn_first_stage Group

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`, and optional `related_ccb_node` 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

```
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;
        related_ccb_node : spice_node_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;
        }
      }
    }
}
```

Simple Attributes

```
is_inverting
is_needed
is_pass_gate
miller_cap_fall
miller_cap_rise
related_ccb_node
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 : value_Boolean ;
```

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 : value_Boolean ;
```

value

Valid values are *true* and *false*. The default is true. Set the value to false for cells such as diodes, antennas, and cload 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 : value_float ;
```

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 : value_float ;
```

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 ;
```

related_ccb_node Simple Attribute

The optional `related_ccb_node` attribute specifies the SPICE node in the subcircuit netlist that is used for the `dc_current` table characterization and waveform measurements.

Syntax

```
related_ccb_node : spice_node_name ;
```

Example

```
related_ccb_node : "XYZ" ;
```

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 : value_enum ;
```

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 : value_boolean ;
```

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

```
mode (mode_definition_name, mode_value) ;
```

Example

```
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

```
dc_current( dc_current_template_id ) { }
  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.

```
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 289.

---

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 283.

---

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 43.

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

```
index_1 ("float, ..., float") ; /* optional */
index_2 ("float, ..., float") ; /* optional */
values ("float, ..., float") ;
```

Group Statement

```
em_max_toggle_rate (em_template_name) {}
```

lifetime_profile Simple Attribute

The optional `lifetime_profile` attribute specifies a label that describes the lifetime of a chip. To support electromigration analysis at different lifetimes in downstream tools, define multiple `electromigration` groups with different values of the `lifetime_profile` attribute. This attribute does not have a default value.

An `electromigration` group where the `lifetime_profile` attribute is not specified, is considered to be the default cell electromigration model.

Syntax

```
lifetime_profile : profile_name ;
```

Example

```
lifetime_profile : lpf_alpf_a ;
```

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

```
related_pin : "name | name_list"
```

name | *name_list*

Name of input pin or pins.

Example

```
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

```
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 the table in when Simple Attribute.

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 `nindex_1` positive floating-point numbers if the table is one-dimensional and can 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`, specified for these two indexes in the `em_max_toggle_rate` group or in the `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") ;
```

em_max_toggle_rate Group

The `em_max_toggle_rate` group is a pin-level group that is defined within the `electromigration` group.

```
library (name) {
  cell (name) {
    pin (name) {
      electromigration () {
        em_max_toggle_rate(em_template_name) {
        ... em_max_toggle_rate description...
        }
      }
    }
  }
}
```

Simple Attribute

```
current_type
```

Complex Attributes

```
index_1 : ("float, ..., float") ; /*this attribute is optional*/
index_2 : ("float, ..., float") ; /*this attribute is optional*/
values : ("float, ..., float") ;
```

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 283. 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 296.

The `output_ccb` group includes all the attributes and subgroups of the ccsn_last_stage Group on page 290.

internal_power Group

An `internal_power` group is defined in a `pin` group, as shown here:

```
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) {}
power (template name) {}
rise_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:

```
internal_power() {
  power (template name) {
    values ("float, ..., float") ;
  }
}
```

This is the syntax for a one-dimensional table using `fall_power`, and `rise_power`:

```
internal_power() {
  fall_power (template name) {
    values ("float, ..., float");
  }
  rise_power (template name) {
    values ("float, ..., float");
  }
}
```

This is the syntax for a two-dimensional table using the `power` group:

```
internal_power() {
  power (template name) {
    values ("float, ..., float");
  }
}
```

This is the syntax for a two-dimensional table using the `related_pin` attribute and the `fall_power` and `rise_power` groups:

```
internal_power() {
  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 303 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 303 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 300, rising_together_group Simple Attribute on page 302, and switching_together_group Simple Attribute on page 303.

Syntax

```
switching_interval : value_float ;
```

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 303 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 20 lists the Boolean operators valid in a `when` statement.

Table 20 Valid Boolean Operators

Operator	Description
'	invert previous expression
!	invert following expression
^	logical XOR
*	logical AND
&	logical AND
space	logical AND
+	logical OR
|	logical OR
1	signal tied to logic 1

Operator	Description
0	signal tied to logic 0

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` 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 fall transition. If you have a `fall_power` group, you must have a `rise_power` group.

The example in rise_power Group 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

```
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.

The example in rise_power Group 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 33 shows cells that contain internal power information in the `pin` group.

Example 33 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_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 ;
```

```
      ...
    }
      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) {
      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 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) {
      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_name_id ) {
      ... transition description ...
      }
    }
  }
}
```

template_name

A value representing the name of a `maxtrans_lut_template` group.

Example

```
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

```
pin() {
  ...
  min_pulse_width() {
    constraint_high : value ;
    constraint_low : value ;
    when : "Boolean expression" ;
    /* enabling condition          */
    sdf_cond : "Boolean expression" ;
    /* in SDF syntax */
  }
}
```

Example

```
pin(A) {
   . . .
   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 the example in Sequential Timing Arcs.

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 : value_float ;
```

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 logic expression format.

Syntax

```
when : "Boolean expression" ;
```

For a list of Boolean operators, see the table in switching_together_group Simple Attribute.

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.

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 the table in when Simple Attribute.

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 (name_string) {
  cell (name_string) {
    pin (name_string) {
      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 365.

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 365.

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`

Applicable only to two-segment receiver capacitance model

`char_when`
`char_when_fall`
`char_when_rise`

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_capacitance` group that represents a default condition timing arc, that is, does not have the conditional `when` attribute.

char_when_fall and char_when_rise Attributes

The `char_when_fall` and `char_when_rise` attributes specify the falling and rising input states at which the receiver capacitance timing arc was characterized. Correlation tools can use this input state for SPICE validation.

Syntax

```
char_when_fall : Boolean expression ;
char_when_rise : Boolean expression ;
```

Example

```
char_when_fall : "A1 * A3" ;
char_when_rise : "A1 * !A3" ;
```

---

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

```
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 |
  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 ;
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_rise_transition(){}
ocv_mean_shift_fall_transition(){}
ocv_mean_shift_rise_constraint(){}
ocv_mean_shift_fall_constraint(){}

ocv_mean_shift_retaining_rise(){}
ocv_mean_shift_retaining_fall(){}
ocv_mean_shift_retain_rise_slew(){}
ocv_mean_shift_retain_fall_slew(){}

ocv_skewness_cell_rise(){}
ocv_skewness_cell_fall(){}
ocv_skewness_rise_transition(){}
ocv_skewness_fall_transition(){}
ocv_skewness_rise_constraint(){}
ocv_skewness_fall_constraint(){}

ocv_skewness_retaining_rise(){}
ocv_skewness_retaining_fall(){}
ocv_skewness_retain_rise_slew(){}
ocv_skewness_retain_fall_slew(){}

ocv_std_dev_cell_rise(){}
ocv_std_dev_cell_fall(){}
ocv_std_dev_rise_transition(){}
ocv_std_dev_fall_transition(){}
ocv_std_dev_rise_constraint(){}
ocv_std_dev_fall_constraint(){}

ocv_std_dev_retaining_rise(){}
```

```
ocv_std_dev_retaining_fall(){}
ocv_std_dev_retain_rise_slew(){}
ocv_std_dev_retain_fall_slew (){}

output_current_fall () { }
output_current_rise () { }
receiver_capacitance_fall
receiver_capacitance_rise
receiver_capacitance1_fall () { }
receiver_capacitance1_rise () { }
receiver_capacitance2_fall () { }
receiver_capacitance2_rise () { }
retaining_fall ()  { }
retaining_rise ()  { }
retain_fall_slew ()  { }
retain_rise_slew ()  { }
rise_constraint ()  { }
rise_propagation ()  { }
rise_transition ()  { }
```

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 : condition_Boolean ;
```

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 ;
  interdependence_id : 2 ;


...
pin (D_IN) {
            ...
/* original nonconditional setup/hold constraints */
```

```
setup/hold constraints
/* new interdependence data for nonconditional constraint
checking */
setup/hold, interdependent_id = 1
setup/hold, interdependent_id = 2
setup/hold, interdependent_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>, interdependent_id = 1
setup/hold when <condition_a>, interdependent_id = 2
setup/hold when <condition_a>, interdependent_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>, interdependent_id = 1
setup/hold when <condition_b>, interdependent_id = 2
setup/hold when <condition_b>, interdependent_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: `setup_rising` with `hold_rising` and `setup_falling` with `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

```
output_signal_level_high : float ;
output_signal_level_low : float ;
```

Example

```
output_signal_level_high : 0.75;
output_signal_level_low : 0.15;
```

output_signal_level_low Simple Attribute

See output_signal_level_high Simple Attribute on page 325.

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

```
related_output_pin : name ;
```

Example

```
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

```
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

```
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

```
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

```
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

```
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

```
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

```
sdf_cond_end : "SDF expression" ;
```

SDF expression

An SDF expression containing names of input, output, inout, and internal pins.

Example

```
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 ;
```

You must distinguish between combinational and sequential timing types, because each type serves a different purpose.

The Design Compiler tool uses the combinational timing arcs information to calculate the physical delays in timing propagation and to trace paths. The timing analyzer uses path-tracing arcs for circuit timing analysis.

The Design Compiler tool uses the sequential timing arcs information to determine rule-based design optimization constraints. More information on optimization constraints is available in the Design Compiler documentation.

The following sections show the `timing_type` attribute values for the following timing arcs. For information about when to use the different types, see the *Synopsys Liberty User Guide*.

- Combinational
- Sequential

- Nonsequential
- No-change

Combinational Timing Arcs

The timing type and timing sense define the signal propagation pattern. The default timing type is combinational. Table 21 shows the timing type and timing sense values for combinational timing arcs.

Table 21 timing_type and timing_sense Values for Combinational Timing Arcs

Timing type	Timing sense		
	Positive_Unate	**Negative_Unate**	**Non_Unate**
`combinational`	R->R,F->F	R->F,F->R	{R,F}->{R,F}
`combinational_rise`	R->R	F->R	{R,F}->R
`combinational_fall`	F->F	R->F	{R,F}->F
`three_state_disable`	R->{0Z,1Z}	F->{0Z,1Z}	{R,F}->{0Z,1Z}
`three_state_enable`	R->{Z0,Z1}	F->{Z0,Z1}	{R,F}->{Z0,Z1}
`three_state_disable_rise`	R->0Z	F->0Z	{R,F}->0Z
`three_state_disable_fall`	R->1Z	F->1Z	{R,F}->1Z
`three_state_enable_rise`	R->Z1	F->Z1	{R,F}->Z1
`three_state_enable_fall`	R->Z0	F->Z0	{R,F}->Z0

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 rising 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 `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 34 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 34 Example Library With timing_type Statements

```
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) {
            index_1("0.2, 0.4, 0.6");
            index_2("0.2, 0.4");
            values( "0.20 0.22", \
              "0.24 0.26" \
              "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", \
              "0.24 0.26" \
              "0.28 0.30");
         }
      }
   }
   ...
   } /* end of arc */
  } /* end of cell */
} /* end of library */
```

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

```
wave_rise_sampling_index : integer ;
wave_fall_sampling_index : integer ;
```

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 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 min_pulse_width Group and minimum_period Group, 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.

For more details, see the “Modeling Power and Electromigration” and “Timing Arcs” chapters in the *Synopsys Liberty User Guide*.

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 logic 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 the table in when Simple Attribute.

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

```
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

```
mode (mode_name, mode_value);
```

Example

```
timing() {
  mode(rw, read);
}
```

Example 35 shows a `mode` description.

Example 35 A mode 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 36 shows multiple `mode` descriptions.

Example 36 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 ");
        }
      }
      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:

```
  wave_rise (vector_id[m]…, vector_id[n]);
  wave_fall (vector_id[j]…, vector_id[k]);
```

Syntax

```
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);
    …
 }
 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_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

```
wave_rise_time_interval (float..., float) ;
```

```
wave_fall_time_interval (float..., float) ;
```

Example

```
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 */
```

Example 37 Specifying cell_degradation in a Lookup Table

```
pin (Z) {
  timing () {
    cell_degradation (constraint) {
      index_1 ("1.0, 1.5, 2.0") ;
      values ("1.0, 1.5, 2.0") ;
    }
    ...
  }
  ...
}
```

For information about the syntax and usage of these attributes, see cell_degradation Group on page 347.

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 (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        cell_fall (name_string){
          ... 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 65.

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 (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        cell_rise (name_string){
          ... 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

```
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 65."

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

```
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 43.

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 (name_string) {
    pin (name_string) {
      timing () {
        fall_constraint (name_string){
          ... fall constraint description...
        }
      }
    }
  }
}
```

Complex Attributes

```
index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
values  ("float, ..., float", ..., "float, ...,float");
```

Example

```
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");
}
```

The example in rise_constraint Group shows constraints in a timing model.

fall_propagation Group

Together with the `rise_propagation` group, the `fall_propagation` group specifies transition delay as a term in the total cell delay.

The `fall_propagation` group is defined in the `timing` group, as shown here.

```
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        fall_propagation (name_string){
          ... fall propagation description...
        }
      }
    }
  }
}
```

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 (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        fall_transition (name_string){
          ... values 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:

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:

```
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        ocv_sigma_cell_fall (template_name_string){
```

```
            ... 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 356.

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:

```
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        ocv_sigma_fall_constraint (template_name_string){
          ... 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 (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        ocv_sigma_fall_transition (template_name_string){
          ... 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 (σ), 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 358.

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 (σ), 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 358.

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(σ), 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:

```
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 360.

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 362.

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`, `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` look up 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.

```
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:

```
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        output_current_fall (name_string){
          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.

```
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
```

```
        output_current_fall (name_string){
          vector () {
            reference_time : ;
          }
        }
      }
    }
  }
}
```

Example

```
timing () {
  output_current_rise () {
    vector (CCT) {
      reference_time : 0.05 ;
      index_1 (0.1) ;
      index_1 (1.1) ;
      index_1 (1, 3, 3, 4, 5) ;
      values ( 1.1, 1.3, 1.5, 1.2, 1.4) ;
    }
  }
}
```

output_current_rise Group

For information about using the `output_current_rise` group, see the definition of the output_current_fall Group on page 363.

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 317.

receiver_capacitance_rise Group

For information about using the `receiver_capacitance_rise` group, see receiver_capacitance_fall Group on page 317.

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 316.

Syntax

```
receiver_capacitance1_fall (value) {
```

Complex Attribute

`values`

Example

```
timing() {
  ...
  receiver_capacitance1_fall () {
    values ("2.0, 4.0, 1.0, 3.0") ;
  }
}
```

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

```
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

```
library (name_string) {
  cell (names_tring) {
    pin (name_string) {
      timing () {
        retaining_rise (name_string){
          ... retaining rise description ...
        }
      }
    }
  }
}
```

Complex Attributes

```
index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
values  ("float, ..., float", "float, ...,float", "float,
...,float");
```

Example

```
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_template_string) {
  values (index1_float, index2_float, index3_float) ;
}
```

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_template_string) {
  values(index1_float, index2_float, index3_float) ;
}
```

retaining_time_template

Name of the table template to use for the lookup table.

index1$_{float}$, index2$_{float}$, index3$_{float}$

Values to use for indexing the lookup table.

Examples

```
cell (cell_name) {
  ...
  pin (pin_name) {
    direction : output :
    ...
    timing ( ) {
      related_pin : "related_pin"
      ...
      retaining_rise (retaining_table_template) {
        values ( "0.00, 0.23", "0.11, 0.28") ;
      }
      ...
      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 (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
```

```
        rise_constraint (name_string){
            ... 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", \
            "0.33, 0.37, 0.50");
}
```

Example 38 shows constraints in a timing model.

Example 38 CMOS Nonlinear Timing Model Using Constraints

```
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( "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

```
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        rise_propagation (name_string){
          ... rise propagation description ...
        }
      }
    }
  }
}
```

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("0.00, 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:

```
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 121.

Syntax

```
library (name_string) {
  cell (name_string) {
    ...
    timing_model_type : value_enum ;
```

```
    ....
    pin (data_pin_name_string) {
      tlatch (enable_pin_name_string){
      ... tlatch description ...
      }
    }
  }
}
```

Simple Attributes

```
edge_type
tdisable
```

edge_type Simple Attribute

Use the `edge_type` attribute to specify whether the latch is positive (high) transparent or negative (low) transparent.

Syntax

```
edge_type : name_id ;
```

name

Valid values are `rising` and `falling`.

Example

```
edge_type : rising ;
```

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 `tlatch` group whose `tdisable` attribute is set to true on an edge triggered flip flop.

Syntax

```
tdisable : value_Boolean
```

value

The valid values are TRUE and FALSE. When set to FALSE, the latch is ignored.

Example

```
tdisable : FALSE ;
```