1. Technology Library Group Description and Syntax

1.1 Library-Level Attributes and Values

1.2 General Syntax

1.3 Reducing Library File Size

1.4 library Group Name
   1.4.1 Syntax
   1.4.2 Example

1.5 library Group Example

1.6 Simple Attributes
   1.6.1 bus_naming_style Simple Attribute
   1.6.2 comment Simple Attribute
   1.6.3 current_unit Simple Attribute
   1.6.4 date Simple Attribute
   1.6.5 default_fpga_isd Simple Attribute
   1.6.6 default_threshold_voltage_group Simple Attribute
   1.6.7 delay_model Simple Attribute
   1.6.8 em_temp_degradation_factor Simple Attribute
   1.6.9 fpga_domain_style Simple Attribute
   1.6.10 fpga_technology Simple Attribute
   1.6.11 in_place_swap_mode Simple Attribute
   1.6.12 input_threshold_pct_fall Simple Attribute
   1.6.13 input_threshold_pct_rise Simple Attribute
   1.6.14 leakage_power_unit Simple Attribute
   1.6.15 nom_calc_mode Simple Attribute
   1.6.16 nom_process Simple Attribute
   1.6.17 nom_temperature Simple Attribute
   1.6.18 nom_voltage Simple Attribute
   1.6.19 output_threshold_pct_fall Simple Attribute
   1.6.20 output_threshold_pct_rise Simple Attribute
   1.6.21 piece_type Simple Attribute
   1.6.22 power_model Simple Attribute
   1.6.23 preferred_output_pad_slew_rate_control Simple Attribute
   1.6.24 preferred_input_pad_voltage Simple Attribute
   1.6.25 preferred_output_pad_voltage Simple Attribute
   1.6.26 pulling_resistance_unit Simple Attribute
   1.6.27 revision Simple Attribute
   1.6.28 simulation Simple Attribute
   1.6.29 slew_derate_from_library Simple Attribute
   1.6.30 slew_lower_threshold_pct_fall Simple Attribute
   1.6.31 slew_lower_threshold_pct_rise Simple Attribute
   1.6.32 slew_upper_threshold_pct_fall Simple Attribute
   1.6.33 slew_upper_threshold_pct_rise Simple Attribute
   1.6.34 time_unit Simple Attribute
   1.6.35 voltage_unit Simple Attribute

1.7 Defining Default Attribute Values in a CMOS Technology Library

1.8 Complex Attributes
   1.8.1 capacitive_load_unit Complex Attribute
   1.8.2 default_part Complex Attribute
   1.8.3 define Complex Attribute
   1.8.4 define_cell_area Complex Attribute
   1.8.5 define_group Complex Attribute
   1.8.6 library_features Complex Attribute
   1.8.7 piece_define Complex Attribute
   1.8.8 routing_layers Complex Attribute
   1.8.9 technology Complex Attribute
   1.8.10 voltage_map Complex Attribute

1.9 Group Statements
2. cell and model Group Description and Syntax

2.1 cell Group

2.1.1 Attributes and Values
2.1.2 Simple Attributes
2.1.3 Complex Attributes
2.1.4 Group Statements

2.2 model Group

2.2.1 Attributes and Values

3. pin Group Description and Syntax

3.1 Syntax of a pin Group in a cell or bus Group

3.1.1 pin Group Example
3.1.2 Simple Attributes
3.1.3 Complex Attributes

3.2 Group Statements

3.2.1 ccsn_first_stage Group
1. Technology Library Group Description and Syntax

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

The information in this chapter includes a description and syntax example 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 Chapter 3, "pin Group Description and Syntax"

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

- Library-Level Attributes and Values
  - General Syntax
  - Reducing Library File Size
  - library Group Name
  - library Group Example
  - Simple Attributes
  - Defining Default Attribute Values in a CMOS Technology Library
  - Complex Attributes
  - Group Statements

1.1 Library-Level Attributes and Values

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

Example 1-1 lists alphabetically a sampling of the attributes, groups, and values that you can define within a technology library. Example 1-2 shows the general syntax and the functional order in which the attributes usually appear within a library group.

Example 1-1 Attributes, Groups, and Values in a Technology Library

```plaintext
library (name string) {
  ... library description ... 
}

/* Library Description: Simple Attributes */

bus_naming_style : "string";
comment : "string";
current_unit : value enum;
date : "date";
delay_model : value enum;
```

Index
em_temp_degradation_factor : float;
fpga_technology : "fpga_technology_name" string;
in_place_swap_mode : match_footprint | no_swapping;
input_threshold_pct_fall : trip_point value;
input_threshold_pct_rise : trip_point value;
leakage_power_unit : value enum
nom_calc_mode : name id;
nom_process : float;
nom_temperature : float;
nom_voltage : float;
output_threshold_pct_fall : trip_point value;
output_threshold_pct_rise : trip_point value;
piece_type : value enum
power_model : tableLookup | polynomial;
preferred_output_pad_slew_rate_control : value enum
preferred_input_pad_voltage : string;
preferred_output_pad_voltage : string;
pulling_resistance_unit : 1ohm | 10ohm | 100ohm | 1kohm;
revision : float | string;
simulation : true | false;
slew_degrade_from_library : derate value;
slew_lower_threshold_pct_fall : trip_point value;
slew_lower_threshold_pct_rise : trip_point value;
slew_upper_threshold_pct_fall : trip_point value;
slew_upper_threshold_pct_rise : trip_point value;
time_unit : lps | 10ps | 100ps | lns;
voltage_unit : lmV | 10mV | 100mV | 1V;

InChildren Description: Default Attributes */
default_cell_leakage_power : float;
default_connection_class : name | name_list string;
default_fall_delay_intercept : float; /* piecewise model only*/
default_fall_pin_resistance : float; /* piecewise model only*/
default_fanout_load : float;
default_inout_pin_cap : float;
default_inout_pin_fall_res : float;
default_inout_pin_rise_res : float;
default_input_cap : float;
default_intrinsic_fall : float;
default_intrinsic_rise : float;
default_leakage_power_density : float;
default_max_capacitance : float;
default_max_fanout : float;
default_max_transition : float;
default_max_utilization : float;
default_min_porosity : float;
default_operating_conditions : name string;
default_output_pin_cap : float;
default_output_pin_fall_res : float;
default_output_pin_rise_res : float;
default_rise_delay_intercept : float; /* piecewise model only*/
default_rise_pin_resistance : float; /* piecewise model only*/
default_slope_fall : float;
default_slope_intercept : float;
default_slope_rise : 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;

InChildren Description: Scaling Attributes */
k_process_cell_fall : float; /* nonlinear model only*/
k_process_cell_leakage_power : float;
k_process_cell_rise : float; /* nonlinear model only*/
k_process_drive_current : float;
k_process_drive_fall : float;
k_process_drive_intercept : float;
k_process_drive_rise : float;
k_process_fall_delay_intercept : float; /* piecewise model only*/
k_process_fall_pin_resistance : float; /* piecewise model only*/
k_process_fall_propagation : float; /* nonlinear model only*/
k_process_fall_transition : float; /* nonlinear model only*/
k_process_hold_fall : float;
k_process_hold_intercept : float;
k_process_hold_rise : float;
k__process__internal__power : float;
k__process__intrinsic__fall : float;
k__process__intrinsic__rise : float;
k__process__min__period : float;
k__process__min__pulse__width__high : float;
k__process__min__pulse__width__low : float;
k__process__nochange__fall : float ; /* nonnegative value */
k__process__nochange__rise : float ; /* nonnegative value */
k__process__pin__cap : float;
k__process__recovery__fall : float;
k__process__recovery__rise : float;
k__process__removal__fall : float;
k__process__removal__rise : float;
k__process__rise__delay__intercept : float ; /* piecewise model only */
k__process__rise__pin__resistance : float ; /* piecewise model only */
k__process__rise__propagation : float ; /* nonlinear model only */
k__process__rise__transition : float ; /* nonlinear model only */
k__process__setup__fall : float;
k__process__setup__rise : float;
k__process__skew__fall : float;
k__process__skew__rise : float;
k__process__slope__fall : float;
k__process__slope__rise : float;
k__process__wire__cap : float;
k__process__wire__res : float;
k_temp__cell__rise : float ; /* nonlinear model only */
k_temp__cell__fall : float ; /* nonlinear model only */
k_temp__cell__leakage__power : float;
k_temp__drive__current : float;
k_temp__drive__fall : float;
k_temp__drive__rise : float;
k_temp__fall__delay__intercept : float ; /* piecewise model only */
k_temp__fall__pin__resistance : float ; /* piecewise model only */
k_temp__fall__propagation : float ; /* nonlinear model only */
k_temp__fall__transition : float ; /* nonlinear model only */
k_temp__hold__fall : float;
k_temp__hold__rise : float;
k_temp__internal__power : float;
k_temp__intrinsic__fall : float;
k_temp__intrinsic__rise : float;
k_temp__min__period : float;
k_temp__min__pulse__width__high : float;
k_temp__min__pulse__width__low : float;
k_temp__nochange__fall : float;
k_temp__nochange__rise : float;
k_temp__pin__cap : float;
k_temp__recovery__fall : float;
k_temp__recovery__rise : float;
k_temp__removal__fall : float;
k_temp__removal__rise : float;
k_temp__rise__delay__intercept : float ; /* piecewise model only */
k_temp__rise__pin__resistance : float ; /* piecewise model only */
k_temp__rise__propagation : float ; /* nonlinear model only */
k_temp__rise__transition : float ; /* nonlinear model only */
k_temp__rise__wire__resistance : float;
k_temp__setup__fall : float;
k_temp__setup__rise : float;
k_temp__skew__fall : float;
k_temp__skew__rise : float;
k_temp__slope__fall : float;
k_temp__slope__rise : float;
k_temp__wire__cap : float;
k_temp__wire__res : float;
k_volt__cell__fall : float ; /* nonlinear model only */
k_volt__cell__leakage__power : float;
k_volt__drive__current : float;
k_volt__drive__fall : float;
k_volt__drive__rise : float;
k_volt__fall__delay__intercept : float ; /* piecewise model only */
k_volt__fall__pin__resistance : float ; /* piecewise model only */
k_volt__fall__propagation : float ; /* nonlinear model only */
k_volt__fall__transition : float ; /* nonlinear model only */
k_volt__hold__fall : float;
k_volt__hold__rise : float;
k_volt__internal__power : float;
k_volt__intrinsic__fall : float;
k_volt__intrinsic__rise : float;
k_volt__min__period : float;
k_volt_min_pulse_width_high : float;
k_volt_min_pulse_width_low : float;
k_volt_nochange_fail : float;
k_volt_nochange_rise : float;
k_volt_pin_cap : float;
k_volt_recovery_fail : float;
k_volt_recovery_rise : float;
k_volt_removal_fail : float;
k_volt_removal_rise : float;
k_volt_rise_delay_intercept : float; /*piecewise model only*/
k_volt_rise_pin_resistance : float; /*piecewise model only*/
k_volt_rise_propagation : float; /*nonlinear model only*/
k_volt_rise_transition : float; /*nonlinear model only*/
k_volt_setup_fail : float;
k_volt_setup_rise : float;
k_volt_skew_fail : float;
k_volt_skew_rise : float;
k_volt_slope_fail : float;
k_volt_slope_rise : float;
k_volt_wire_cap : float;
k_volt_wire_res : 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);
piece_define ("range0 [range1 range2 ...]");
routing_layers ("routing_layer_1_name", ..., "routing_layer_n_name");
technology ("name");

/* Library Description: Group Statements*/
cell (name) {}
dc_current_template (template_name id) {}
em_lut_template (name) {}
fall_net_delay : name;
fall_transition_degradation (name) {}
faults_lut_template (name) {}
input_voltage (name) {}
iv_lut_template (name string) {}
lu_table_template (name string) {}
noise_lut_template (name string) {}
operating_conditions (name) {}
output_voltage (name) {}
poly_template (template_name id) {}
power_lut_template (template_name id) {}
power_poly_template () {}
power_supply () {}
propagation_lut_template (name string) {}
rise_net_delay : name;
rise_transition_degradation () {}
scaled_cell (name, op_conds) {}
scaling_factors (name) {}
timing (name | name_list) {}
timing_range (name) {}
type (name) {}
wire_load (name) {}
wire_load_selection (name) {}
wire_load_table (name) {}

1.2 General Syntax

Example 1-2 shows the general syntax of the library group. The first line names the library. Subsequent lines show simple and complex attributes that apply to the library as a whole, such as technology, date, and revision.

The example indicates where you place the default and scaling factors in library syntax. Every cell in the library has a separate cell description.

Note:

Example 1-2 does not contain every attribute or group listed in Example 1-1.

Example 1-2  General Syntax of a Technology Library

library (name: string) {

    // Library-Level Simple and Complex Attributes */

technology (name: string);
delay_model: "model";
buss_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);
piece_type: type;
piece_define: "range0 range1 range2 ..."

define_cell_area (area_name, resource_type);
revision: float | string;
in_place_swap_mode: match_footprint | no_swapping;
simulation: true | false;

    /* 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 ...
    }

timing_range (name: string) {
    ... timing range description ...
}
wire_load (name: string) {
    ... wire load description ...
}
wire_load_selection (name: string) {
    ... wire load selection criteria...
}
poly_template (name: string) {
    ... polynomial template information...
}
power_lut_template (name: string) {
    ... power lookup table template information...
}
power_poly_template (name: string) {
    ... power polynomial template information...
}
power_supply () {
    ... power supply information...
}
/* Cell definitions */
cell (name: string2) {
    ... cell description ...
}
scaled_cell (name: string) {
    ... scaled cell description ...
}
...

type (name: string) {
    ... type description ...
}
input_voltage (name: string) {
    ... input voltage information ...
1.3 Reducing Library File Size

Large library files can compromise disk capacity and memory resources. To reduce file size and improve file management, the syntax allows you to combine multiple source files by referencing the files from within the source file containing the library group description.

Use the `include_file` statement to reference information in another file for inclusion during library compilation. Be sure the directory of the included file is defined in your search path—the `include` attribute takes only the file name as its value; a path is not allowed.

**Syntax**

```
include_file(file_name id);
```

**Example**

```plaintext
cell() {
  area : 0.1;
  ...
  include_file(opc_file);
  ...
}
```

where `opc_file` contains the `operating_conditions` group statements.

**Limitations**

The `include_file` attribute has these requirements:

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

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

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

  ```plaintext
cell(A) include(source_file id)
```

where `source_file id` is the following:

```plaintext
{
  attribute : value;
  attribute : value;
  ...
}
1.4 library Group Name

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

1.4.1 Syntax

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

1.4.2 Example

A library called example1 looks like this:

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

1.5 library Group Example

Example 1-3 shows a portion of a library group for a CMOS library. It contains buses and uses a nonlinear timing delay model.

Example 1-3  CMOS library Group

library (example1) {
  technology (cmos);
  delay_model : table_lookup;
  date : "December 12, 2003";
  comment : "Copyright 2003, General Silicon, Inc.";
  revision : 2003.12;
  bus_naming_style : "Bus%sPin%d";
  ...
}

1.6 Simple Attributes

Following are descriptions of the library group simple attributes. Similar sections describing the default, complex, and group statement attributes complete this chapter.

1.6.1 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

Can contain alphanumeric characters, braces, underscores, dashes, or parentheses. Must contain one %s symbol and one %d symbol. The %s and %d symbols can appear in any order, but at least one nonnumeric character must separate them.

The colon character is not allowed in a bus_naming_style attribute value because the colon is used to denote a range of bus members.

You construct a complete bused-pin name by using the name of the owning bus and the member number. The owning bus name is substituted for the %s, and the member number replaces the %d.

If you do not define the bus_naming_style attribute, Liberty applies the default naming convention, as shown in the following example.
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.

1.6.2 comment Simple Attribute

You use the comment attribute to include copyright or other product information in the library report that you generate using the report_lib command. 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.

Example

The following comment line:

`comment : "Copyright 2003, General Silicon, Inc." ;`

appears in the report like this:

Comment : Copyright 2003, General Silicon, Inc.

1.6.3 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 : valueenum ;`

`value`

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

Example

`current_unit : 100uA ;`

1.6.4 date Simple Attribute
The optional date attribute identifies the date your library was created. The date appears in the library report produced by the report_lib command.

Syntax

date : "date" ;

date

You can use any format within the quotation marks to report the date.

Example

date : "12 December 2003" ;

1.6.5 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_nameid ;

fpga_isd_name

The name of the default fpga_isd group.

Example

default_fpga_isd : lib_isd ;

1.6.6 default_threshold_voltage_group Simple Attribute

The default_threshold_voltage_group attribute specifies the name of a category to which a cell belongs, based on the voltage characteristics of the cell.

Syntax

default_threshold_voltage_group : group_nameid ;

group_name

A string value representing the name of the category.

Example

default_threshold_voltage_group : "high_vt_cell" ;

1.6.7 delay_model Simple Attribute

Use the delay_model attribute to specify which delay model to use in the delay calculations.

The delay_model attribute must be the first attribute in the library if a technology attribute is not present. Otherwise, it should follow the technology attribute.

Syntax

delay_model : valueenum ;

value

Valid values are generic_cmos, table_lookup (nonlinear delay model), piecewise_cmos, dcm (delay calculation module), and polynomial (scalable polynomial delay model).
Example
delay_model : table_lookup;

1.6.8 *em_temp_degradation_factor Simple Attribute*

The *em_temp_degradation_factor* attribute specifies the electromigration exponential degradation factor.

**Syntax**

```plaintext
em_temp_degradation_factor : value float;
```

`value`

A floating-point number in centigrade units consistent with other temperature specifications throughout the library.

**Example**

```plaintext
em_temp_degradation_factor : 40.0;
```

1.6.9 *fpga_domain_style Simple Attribute*

Use the *fpga_domain_style* attribute to specify a value that you reference from a calc_mode attribute in a domain group in a polynomial table.

**Syntax**

```plaintext
fpga_domain_style : "nameid";
```

`name`

The style value.

**Example**

```plaintext
fpga_domain_style : "speed";
```

1.6.10 *fpga_technology Simple Attribute*

Use this attribute to specify your FPGA technology. This attribute is required when your library technology is FPGA.

**Syntax**

```plaintext
fpga_technology : "fpga_technology_name string";
```

`fpga_technology_name`

The name of your FPGA technology.

**Example**

```plaintext
fpga_technology : "my_fpga_technology_1";
```

1.6.11 *in_place_swap_mode Simple Attribute*

In-place optimization occurs after placement and routing.

The *in_place_swap_mode* attribute specifies the criteria for cell swapping during in-place optimization. The basic criteria for cell swapping are:

- The cells must have the same function.
- The cells must have the same number of pins, and the pins must have the same
pin names.

Syntax

in_place_swap_mode : match_footprint | no_swapping ;

match_footprint

Cells are swapped if they meet the criteria and have the same footprint attribute.

no_swapping

In-place optimization is disabled. The cell_footprint attribute is ignored. The no_swapping value is the default for CMOS libraries.

Example

in_place_swap_mode : match_footprint ;

Use the report_lib command to print the type of in-place swap mode your technology library is using.

1.6.12 input_threshold_pct_fall Simple Attribute

Use the input_threshold_pct_fall attribute to set the default value of the 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 value.

Syntax

input_threshold_pct_fall : trip_pointfloat ;

trip_point

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

Example

input_threshold_pct_fall : 60.0 ;

1.6.13 input_threshold_pct_rise Simple Attribute

Use the input_threshold_pct_rise attribute to set the default value of the 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 value.

Syntax

input_threshold_pct_rise : trip_pointfloat ;

trip_point

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

Example

input_threshold_pct_rise : 40.0 ;

1.6.14 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, 100uW (for 100mW), 10uW (for 10mW), 1uW (for 1mW), 100nW, 10nW, 1nW, 100pW, 10pW, and 1pW.

Example

leakage_power_unit : "100uW";

1.6.15 nom_calc_mode Simple Attribute

This optional attribute defines a default process point, one of the nominal operating conditions for a library.

Syntax

nom_calc_mode : name id;

name

Represents the default process point.

Example

nom_calc_mode : nominal;

1.6.16 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;

1.6.17 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;
1.6.18  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;

1.6.19  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 value is 50.0.

Example

output_threshold_pct_fall : 40.0;

1.6.20  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 value is 50.0.

Example

output_threshold_pct_rise : 40.0;

1.6.21  piece_type Simple Attribute

The piece_type attribute allows you the option of using capacitance to define the piecewise linear model.

Syntax

piece_type : value_enum;

value

Valid values are piece_length, piece_wire_cap, piece_pin_cap,
and piece_total_cap.

Example

```plaintext
    piece_type : piece_length;
```

The `piece_wire_cap`, `piece_pin_cap`, and `piece_total_cap` values represent the piecewise linear model extensions that cause modeling to use capacitance instead of length. These values indicate wire capacitance alone, total pin capacitance, or the total wire and pin capacitance. If the `piece_type` attribute is not defined, modeling defaults to the `piece_length` model.

### 1.6.22 power_model Simple Attribute

The `power_model` attribute allows you to specify whether your library utilizes lookup tables or scalable polynomial equations to model power.

**Syntax**

```plaintext
    power_model : value;
```

**Value**

The valid values you can specify are `table_lookup` and `polynomial`. If no value is specified, `table_lookup` is assumed.

**Example**

```plaintext
    power_model : polynomial;
```

### 1.6.23 preferred_output_pad_slew_rate_control Simple Attribute

Use the `preferred_output_pad_slew_rate_control` attribute to embed directly in the library the desired preferred slew-rate control value.

**Syntax**

```plaintext
    preferred_output_pad_slew_rate_control : value enum;
```

**Value**

Valid values are `high`, `medium`, `low`, and `none`.

**Example**

```plaintext
    preferred_output_pad_slew_rate_control : high;
```

### 1.6.24 preferred_input_pad_voltage Simple Attribute

Use the `preferred_input_pad_voltage` and the `preferred_output_pad_voltage` attributes to embed in the library the preferred voltage values.

**Syntax**

```plaintext
    preferred_input_pad_voltage : name1 string; preferred_output_pad_voltage : name2 string;
```

**name1**

A string name for any of the input voltage groups defined in the library.

**name2**

A string name for any of the output voltage groups defined in the library.
For more information about output and input voltage groups, see the “Defining Core Cells” and “Defining I/O Pads” chapters in the Liberty User Guide, Volume 1.

### 1.6.25 preferred_output_pad_voltage Simple Attribute

For information about using the `preferred_output_pad_voltage` attribute, see the description of the “preferred_input_pad_voltage Simple Attribute”.

### 1.6.26 pulling_resistance_unit Simple Attribute

Use the `pulling_resistance_unit` attribute to define pulling resistance values for pull-up and pull-down devices.

**Syntax**

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

*unit*

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

**Example**

```plaintext
pulling_resistance_unit : "10ohm";
```

### 1.6.27 revision Simple Attribute

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

**Syntax**

```plaintext
revision : value;
```

*value*

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

**Example**

```plaintext
revision : V3.1a;
```

### 1.6.28 simulation Simple Attribute

Setting the `simulation` attribute to true lets a tool simulation library files.

**Syntax**

```plaintext
simulation : true | false;
```

The default for the `simulation` attribute is true.

**Example**

```plaintext
simulation : true;
```

### 1.6.29 slew_derate_from_library Simple Attribute

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

**Syntax**

```plaintext
slew_derate_from_library : deratefloat;
```

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

Example

slew_derate_from_library: 0.5;

1.6.30  **slew_lower_threshold_pct_fall Simple Attribute**

Use the `slew_lower_threshold_pct_fall` attribute to set the default value of the lower threshold point used for modeling the delay of a pin falling from 1 to 0. You can specify this attribute at the pin-level to override the default value.

**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 for modeling the delay of a pin falling from 1 to 0. The default value is 20.0.

**Example**

```
slew_lower_threshold_pct_fall : 30.0;
```

1.6.31  **slew_lower_threshold_pct_rise Simple Attribute**

Use the `slew_lower_threshold_pct_rise` attribute to set the default value of the lower threshold point used in modeling the delay of a pin rising from 0 to 1. You can specify this attribute at the pin-level to override the default value.

**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 for modeling the delay of a pin rising from 0 to 1. The default value is 20.0.

**Example**

```
slew_lower_threshold_pct_rise : 30.0;
```

1.6.32  **slew_upper_threshold_pct_fall Simple Attribute**

Use the `slew_upper_threshold_pct_fall` attribute to set the default value of the upper threshold point used for modeling the delay of a pin falling from 1 to 0. You can specify this attribute at the pin-level to override the default value.

**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 falling from 1 to 0. The default value is 80.0.

**Example**

```
slew_upper_threshold_pct_fall : 70.0;
```
1.6.33  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 for modeling the delay of a pin rising from 0 to 1. You can specify this attribute at the pin-level to override the default value.

**Syntax**

```plaintext
slew_upper_threshold_pct_rise : trip_pointvalue;
```

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

**Example**

```plaintext
slew_upper_threshold_pct_rise : 70.0;
```

1.6.34  time_unit Simple Attribute

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

**Syntax**

```plaintext
time_unit : unit;
```

- **unit**
  
  Valid values are 1ps, 10ps, 100ps, and 1ns. The `time_unit` attribute default value is 1ns.

**Example**

```plaintext
time_unit : 100ps;
```

1.6.35  voltage_unit Simple Attribute

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

**Syntax**

```plaintext
voltage_unit : unit;
```

- **unit**
  
  Valid values are 1mV, 10mV, 100mV, and 1V. The default value is 1V.

**Example**

```plaintext
voltage_unit : 100mV;
```

1.7  Defining Default Attribute Values in a CMOS Technology Library

Within the `library` group of a CMOS technology library, you can define default values for the `pin` and `timing` group attributes. Then, as needed, you can override the default settings by defining corresponding attributes at the individual `pin` or `timing` group levels.

The following tables list the default attributes that you can define within the `library` group and the attributes that override them.

- Table 1-1 lists the default attributes you can use in all the CMOS models.
Table 1-2 lists the default attributes you can use in Piecewise linear delay models.

Table 1-3 lists the default attributes you can use in CMOS linear delay models.

Table 1-1  CMOS Default Attributes for All Models

<table>
<thead>
<tr>
<th>Default attribute</th>
<th>Description</th>
<th>Override with</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_cell_leakage_power</td>
<td>Default leakage power</td>
<td>cell_leakage_power</td>
</tr>
<tr>
<td>default_connection_class</td>
<td>Default connection class</td>
<td>connection_class</td>
</tr>
<tr>
<td>default_fanout_load</td>
<td>Fanout load of input pins</td>
<td>fanout_load</td>
</tr>
<tr>
<td>default_inout_pin_cap</td>
<td>Capacitance of inout pins</td>
<td>capacitance</td>
</tr>
<tr>
<td>default_input_pin_cap</td>
<td>Capacitance of input pins</td>
<td>capacitance</td>
</tr>
<tr>
<td>default_intrinsic_fall</td>
<td>Intrinsic fall delay of a timing arc</td>
<td>intrinsic_fall</td>
</tr>
<tr>
<td>default_intrinsic_rise</td>
<td>Intrinsic rise delay of a timing arc</td>
<td>intrinsic_rise</td>
</tr>
<tr>
<td>default_leakage_power_density</td>
<td>Default leakage power density</td>
<td>cell_leakage_power</td>
</tr>
<tr>
<td>default_max capacitance</td>
<td>Maximum capacitance of output pins</td>
<td>max_capacitance</td>
</tr>
<tr>
<td>default_max_fanout</td>
<td>Maximum fanout of all output pins</td>
<td>max_fanout</td>
</tr>
<tr>
<td>default_max_transition</td>
<td>Maximum transition of output pins</td>
<td>max_transition</td>
</tr>
<tr>
<td>default_max_utilization</td>
<td>Maximum limit of utilization</td>
<td>No override available</td>
</tr>
<tr>
<td>default_min_porosity</td>
<td>Minimum porosity constraint</td>
<td>set_min_porosity</td>
</tr>
<tr>
<td>default_operating_conditions</td>
<td>Default operating conditions for the library</td>
<td>operating_conditions</td>
</tr>
<tr>
<td>default_output_pin_cap</td>
<td>Capacitance of output pins</td>
<td>capacitance</td>
</tr>
<tr>
<td>default_slope_fall</td>
<td>Fall sensitivity factor of a timing arc</td>
<td>slope_fall</td>
</tr>
<tr>
<td>default_slope_rise</td>
<td>Rise sensitivity factor of a timing arc</td>
<td>slope_rise</td>
</tr>
<tr>
<td>default_wire_load</td>
<td>Wire load</td>
<td>No override available</td>
</tr>
<tr>
<td>default_wire_load_area</td>
<td>Wire load area</td>
<td>No override available</td>
</tr>
<tr>
<td>default_wire_load capacitance</td>
<td>Wire load capacitance</td>
<td>No override available</td>
</tr>
<tr>
<td>default_wire_load_mode</td>
<td>Wire load mode</td>
<td>set_wire_load ~mode</td>
</tr>
<tr>
<td>default_wire_load_resistance</td>
<td>Wire load resistance</td>
<td>No override available</td>
</tr>
<tr>
<td>default_wire_load_selection</td>
<td>Wire load selection</td>
<td>No override available</td>
</tr>
</tbody>
</table>

Table 1-2  CMOS Default Attributes for Piecewise Linear Delay Models

<table>
<thead>
<tr>
<th>Default attribute</th>
<th>Description</th>
<th>Override with</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_fall_delay_intercept</td>
<td>Falling-edge intercept of a timing arc</td>
<td>fall_delay_intercept</td>
</tr>
<tr>
<td>default_fall_pin_resistance</td>
<td>Fall resistance of output pins</td>
<td>fall_pin_resistance</td>
</tr>
</tbody>
</table>
### 1.8 Complex Attributes

Following are the descriptions of the technology library complex attributes.

#### 1.8.1 capacitive_load_unit Complex Attribute

The **capacitive_load_unit** attribute specifies the unit for all capacitance values within the technology library, including default capacitances, max fanout capacitances, pin capacitances, and wire capacitances.

**Syntax**

capacitive_load_unit (value\text{float}, unit\text{enum})

- **value**
  - A floating-point number.
- **unit**
  - Valid values are ff and pf.

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.

**Example**

```plaintext
capacitive_load_unit (1, pf);
capacitive_load_unit (0.059, pf);
```

The **capacitive_load_unit** attribute has no default value when the attribute is omitted.

#### 1.8.2 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\text{name}\text{string}, speed\text{grade}\text{string})

- **default_part_name**
  - The name of the default part.
- **speed_grade**
  - The speed grade the design uses.
Example
default_part ("AUTO", "+5");

1.8.3 define Complex Attribute

Use this special 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.

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";
```

1.8.4 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` 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) ;
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;
    ...
}
```

### 1.8.5 define_group Complex Attribute

Use this special attribute to define new, temporary, or user-defined groups for use in technology libraries.

**Syntax**

```
define_group (groupid, parent_nameid);
```

- `group`
  - The name of the user-defined group.
- `parent_name`
  - The name of the group statement in which the attribute is to be used.

The following example shows how you define a new group called `myGroup`:

**Example**

```
define_group (myGroup, timing);
```

### 1.8.6 library_features Complex Attribute

The `library_features` attribute lets other tools use the command features that you specify as attribute values.

**Syntax**

```
library_features (valueid);
```

- `value`
  - Valid values are `report_delay_calculation`, `report_noise_calculation`, `report_user_data` and `allow_update_attribute`. The default value is `none`.

  - `report_delay_calculation`
    - Displays timing information.
  
  - `report_noise_calculation`
    - Displays noise information for tied-off cells.
  
  - `report_user_data`
    - Allows the `report_lib` command to display information about user-defined attributes and groups.
  
  - `allow_update_attribute`
    - Allows the `update_lib` and `set_lib_attribute` commands to overwrite user-defined attribute values.

  **Example**
1.8.7  **piece_define Complex Attribute**

The `piece_define` complex attribute statement defines the pieces used in the piecewise linear delay model. With this attribute, you can define the ranges of length or capacitance for indexed variables, such as `rise_pin_resistance`, used in the delay equations.

**Syntax**

```
piece_define ("range0 range1 range2 ...") ;
```

Each range is a floating-point number defining the lower limit of the respective range. If the piecewise linear model is in the `piece_length` mode, as described in the `piece_type` attribute section, a wire whose length is between `range0` and `range1` is named as piece 0, a wire whose length is between `range1` and `range2` is piece 1, and so on.

For example, in the following `piece_define` specification, a wire of length 5 is referred to as piece 0, a wire of length 12 is piece 1, and a wire of length 20 or more is piece 2.

**Example**

```
piece_define ("0 10 20") ;
```

Each capacitance is a positive floating-point number defining the lower limit of the respective range. A piece of wire whose capacitance is between `range0` and `range1` is identified as `piece0`, a capacitance between `range1` and `range2` is piece 1, and so on.

You must include in the `piece_define` statement all ranges of wire length or capacitance for which you want to enter a unique attribute value.

1.8.8  **routing_layers Complex Attribute**

The `routing_layers` attribute declares the routing layers available for place and route for the library. The `routing_layers` attribute is a string that represents the symbolic name used later in a library to describe routability information associated with each layer. The `routing_layers` attribute must be defined in the library before other routability information in a cell. Otherwise, cell routability information in the library is considered an error. Each different library can have only one `routing_layers` complex attribute.

**Syntax**

```
routing_layers("routing_layer_1_name",..., "routing_layer_n_name") ;
```

The `report_lib` command displays library routing layer information. The report looks similar to the following example.

```
Porosity information:
Routing_layers:
"metal2" "metal3"
default_min_porosity: 15.0
```

If there is no porosity information in the library, `report_lib` displays the following line:

```
No porosity information specified.
```

1.8.9  **technology Complex Attribute**

Use this attribute to specify which technology family is used in the library. When you define the `technology` attribute, it must be the first attribute you use and it must be placed at the top of the listing (see Example 1-2).

**Syntax**
technology (name\textsubscript{enum}) :

   name

   Valid values are CMOS or FPGA. If you specify FPGA, you must also specify the fpga\_technology attribute at the library level. The default is CMOS.

Example

   technology (cmos);

1.8.10 voltage\_map Complex Attribute

Use this attribute to specify the cell-level pg\_pin groups.

Syntax

   voltage\_map (voltage\_name\textsubscript{id}, voltage\_value\textsubscript{float}) :

      voltage\_name

      Specifies a power supply.

      voltage\_value

      Specifies a voltage value.

Example

   voltage\_map (VDD1, 3.0);

1.9 Group Statements

Following are the descriptions of the technology library group statement attribute.

1.9.1 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

1.9.2 base\_curve\_type Complex Attribute

The base\_curve\_type attribute specifies the type of base curve. Currently, the only valid value for base\_curve\_type is ccs\_timing\_half\_curve.
### Syntax

```
base_curve_type: enum (ccs_timing_half_curve);
```

### Example

```
base_curve_type: ccs_timing_half_curve;
```

#### 1.9.3 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 better readability and easier 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. The `curve_x` value must be between 0 and 1 and increase monotonically for `ccs_timing_half_curve` type base curves.

### Syntax

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

### Example

```
curve_x ("0.2, 0.5, 0.8");
```

#### 1.9.4 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 better readability and easier implementation.

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");
```

#### 1.9.5 compact_lut_template Group

The `compact_lut_template` group is a lookup table template used for compact CCS timing 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
```
1.9.6 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 is used.

Syntax

_base_curves_group : base_curves_name ;

Example

base_curves_group : ctbc1 ;

1.9.7 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 ;

1.9.8 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 ;

1.9.9 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") ;

1.9.10 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 in order 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");

1.9.11 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 (namestring) {
  dc_current_template (template_nameid) {
    ...
    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");
  }
1.9.12  *em_lut_template* Group

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

**Syntax**

```plaintext
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:

```plaintext
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**

```plaintext
index_1 ("float, ..., float");
index_2 ("float, ..., float");
```

For `index_1`, the floating-point numbers that specify the breakpoints of the first dimension of the electromigration table used to characterize cells for electromigration within the library. For `index_2`, the floating-point numbers that specify the breakpoints for the second dimension of the electromigration table used to characterize cells for electromigration within
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 are the rules for the relationship between variables and indexes:

- If you have `variable_1`, you can have only `index_1`.
- If you have `variable_1` and `variable_2`, you can have `index_1` and `index_2`.
- The value you enter for `variable_1` (used for one-dimensional tables) is determined by how `index_1` is measured. The value you enter for `variable_2` (used for two-dimensional tables) is determined by how `index_2` is measured.

**Examples**

```plaintext
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");
}
```

### 1.9.13 fall_net_delay Group

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

```plaintext
library (name) {
    fall_net_delay (name) {
        ... fall_net_delay description ...
    }
}
```

**Complex Attributes**

```plaintext
    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, as shown in the following examples.

**Example 1-4** shows an example of the `fall_net_delay` group.

**Example 1-4 fall_net_delay Group**

```plaintext
fall_net_delay (net_delay_table_template) {
    index_1 ("0, 1, 2");
    index_2 ("1, 0, 2");
    values ("0.00, 0.57", "0.10, 0.48");
}
```

### 1.9.14 fall_transition_degradation Group
The `fall_transition_degradation` group is defined at the library level, as shown here:

```liberty
library (name) {
    fall_transition_degradation(name) {
        ... fall transition degradation description ...
    }
}
```

**Complex Attributes**

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

The `fall_transition_degradation` group and the `rise_transition_degradation` group describe, in the form of lookup tables, the transition degradation functions for rise and fall transitions. The lookup tables are indexed by the transition time at the net driver and the connect delay between the driver and a particular load. This indexing allows the library developer to model degraded transitions as any function of output-pin transition and connect delay between the driver and the load.

Transition degradation tables are used for indexing into any delay table in a library that has the `input_net_transition`, `constrained_pin_transition`, or `related_pin_transition` table parameters in the `lu_table_template` group.

The transition degradation tables in one library have no effect on computations related to cells from other libraries. **Example 1-5** shows a `fall_transition_degradation` group.

**Example 1-5 fall_transition_degradation Group**

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

1.9.15 `faults_lut_template`

To model yield information, use the `faults_lut_template` group to specify fab names and time ranges applicable for all fault tables in the .lib file. You define fault tables in the cell-level `functional_yield_metric` group using the `average_number_of_faults` attribute. See "[functional_yield_metric Group](#)". Define the `faults_lut_template` group at the library level, as shown here:

**Syntax**

```liberty
library (name_string) {
    ...
    faults_lut_template(name_string) {
        variable_1: value_enum;
        variable_2: value_enum;
        index_1 ("float, ..., float");
        index_2 ("float, ..., float");
    }
}
```

**Simple Attributes**

- `variable_1`
- `variable_2`

**Complex Attributes**
The following are the values you can assign to the variables:

```plaintext
faults_lut_template {namestring} {
  variable_1 : fab_name;
  variable_2 : time_range;
}
```

**Example**

```plaintext
library (my_library_name) {
  ...
  faults_lut_template (my_faults_temp) {
    variable_1 : fab_name;
    variable_2 : time_range;
    index_1 (" fab1, fab2, fab3 ");
    index_2 (" 2005.01, 2005.07, 2006.01, 2006.07 ");
  }
  ...
  cell (and2) {
    functional_yield_metric () {
      average_number_of_faults (my_faults_temp) {
        values (" 73.5, 78.8, 85.0, 92 ";
               " 74.3, 78.7, 84.8, 92.2 ";
               " 72.2, 78.1, 84.3, 91.0 ");
      }
    }
    ...}
  } /* end of cell */
  } /* end of library */
```

This template example represents fault data for three fab lines (fab1, fab2, fab3) and holds fault data collected for four time ranges:

- 2005.01 represents a time range from January 2005 to June 2005
- 2005.07 represents a time range from July 2005 to December 2005
- 2006.01 represents a time range from January 2006 to June 2006
- 2006.07 represents July 2006 or later

For details on the `functional_yield_metric` group, see "[functional_yield_metric Group](#)".

### 1.9.16 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**

```plaintext
library (namestring) {
  input_voltage (namestring) {
    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.

\[ \text{vimin} \]

The minimum acceptable input voltage.

\[ \text{vimax} \]

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**

```plaintext
pin() {
  ...
  input_voltage : my_input_volages;
  ...
}
```

The value of each attribute is expressed as a floating-point number, an expression, or both. Table 1-4 lists the predefined variables that can be used in an expression.

<table>
<thead>
<tr>
<th>CMOS/BiCMOS variable</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD</td>
<td>5V</td>
</tr>
<tr>
<td>VSS</td>
<td>0V</td>
</tr>
<tr>
<td>VCC</td>
<td>5V</td>
</tr>
</tbody>
</table>

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

All voltage values are in the units you define with the `library` group `voltage_unit` attribute.

**Example 1-6** shows a collection of `input_voltage` groups.

**Example 1-6  input_voltage Groups**

```plaintext
input_voltage(CMOS) {
  vil : 0.3 * VDD;
  vih : 0.7 * VDD;
  vimin : -0.5;
  vimax : VDD + 0.5;
}

input_voltage(TTL_5V) {
  vil : 0.8;
  vih : 2.0;
  vimin : -0.5;
  vimax : VDD + 0.5;
}
```

### 1.9.17 fpga_isd Group

You can define one or more `fpga_isd` groups at the library-level to specify the drive current, IO 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 to use as the default.
Syntax

library (namestring) {
    fpga_isd (fpga_isd_namestring) {
        ... description ...
    }
}

Example

fpga_isd (part_cell_isd) {
    ... description ...
}

Simple Attributes

- drive
- io_type
- slew

**drive Simple Attribute**

The `drive` attribute is optional and specifies the output current of the FPGA part or the FPGA cell.

**Syntax**

```
 drive : value_id
   value
   A string
```

**Example**

```
 drive : 24;
```

**io_type Simple Attribute**

The `io_type` attribute is required and specifies the input or output voltage of the FPGA part or the FPGA cell.

**Syntax**

```
io_type : value_id
   value
   A string
```

**Example**

```
io_type : LVTLL;
```

**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
```

Valid values are FAST and SLOW.

**Example**

```plaintext
slew : FAST ;
```

### 1.9.18 *iv_lut_template* Group

The *iv_lut_template* group describes a template for specifying a current-voltage curve. The template specifies I-V output voltage of the breakpoints.

**Syntax**

```plaintext
library (name_string) {
  iv_lut_template (template_name_string) {
    ... template description ...
  }
}
```

**Simple Attribute**

```plaintext
variable_1
```

**Complex Attribute**

```plaintext
index_1
```

**variable_1 Simple Attribute**

You can assign the following value to the template for one-dimensional current-voltage tables.

```plaintext
variable_1 : iv_output_voltage ;
```

**index_1 Complex Attribute**

```plaintext
index_1 ("float, ..., float");
```

**Example**

```plaintext
library (my_library) {
  ...
  iv_lut_template (my_template) {
    variable_1 : iv_output_voltage ;
    index_1 ("-1, -0.1, 0.8, 1.6, 2");
  }
  ...
}
```

### 1.9.19 *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 here:

**Syntax**

```plaintext
library (name_string) {
  ...
  lu_table_template (name_string) {
    variable_1 : values_enum;
  }
  ...
}
```
Simple Attributes

variable_1
variable_2
variable_3

Complex Attributes

index_1
index_2
index_3

Group

domain

normalized_voltage Variable

The normalized_voltage variable is specified under the lu_table_template table in order 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, Volume 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 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:

\[
\text{lu_table_template(}<\text{dc_template_name}>)( \{ \\
\text{variable}_1 : \text{input}_\text{voltage}; \\
\text{variable}_2 : \text{output}_\text{voltage}; \\
\} )
\]

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.

\[
\text{lu_table_template(}<\text{output_voltage_template_name}>)( \{ \\
\text{variable}_1 : \text{input}_\text{net}_\text{transition}; \\
\text{variable}_2 : \text{total}_\text{output}_\text{net}_\text{capacitance}; \\
\text{variable}_3 : \text{time}; \\
\} )
\]

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.

\[
\text{lu_table_template(}<\text{propagated_noise_template_name}>)( \{ \\
\text{variable}_1 : \text{input}_\text{noise}_\text{height}; \\
\text{variable}_2 : \text{input}_\text{noise}_\text{width}; \\
\text{variable}_3 : \text{total}_\text{output}_\text{net}_\text{capacitance}; \\
\text{variable}_4 : \text{time}; \\
\} )
\]

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:

\[
\text{input}_\text{net}_\text{transition} | \text{total}_\text{output}_\text{net}_\text{capacitance} | \\
\text{output}_\text{net}_\text{length} | \text{output}_\text{net}_\text{wire}_\text{cap} | \\
\text{output}_\text{net}_\text{pin}_\text{cap} | \text{related}_\text{out}_\text{total}_\text{output}_\text{net}_\text{capacitance} | \\
\text{related}_\text{out}_\text{output}_\text{net}_\text{length} | \\
\text{related}_\text{out}_\text{output}_\text{net}_\text{wire}_\text{cap} | \\
\text{related}_\text{out}_\text{output}_\text{net}_\text{pin}_\text{cap};
\]

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:

Following are the values (divided into sets) that you can assign for variable_1, variable_2, and variable_3 to the templates for constraint tables:

\[
\text{constrained}_\text{pin}_\text{transition} | \text{related}_\text{pin}_\text{transition} | \\
\text{related}_\text{out}_\text{total}_\text{output}_\text{net}_\text{capacitance} | \\
\text{related}_\text{out}_\text{output}_\text{net}_\text{length} | \\
\text{related}_\text{out}_\text{output}_\text{net}_\text{wire}_\text{cap} | \\
\text{related}_\text{out}_\text{output}_\text{net}_\text{pin}_\text{cap};
\]

In Wire Delay Tables:
The following is the value set that you can assign for \texttt{variable}_1, \texttt{variable}_2, and \texttt{variable}_3 to the templates for wire delay tables:

\begin{verbatim}
fanout_number | fanout_pin_capacitance | driver_slew
\end{verbatim}

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 \texttt{variable}_1 and \texttt{variable}_2 to the templates for net delay tables:

\begin{verbatim}
output_transition | rc_product
\end{verbatim}

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:

\begin{verbatim}
variable_1: output_pin_transition | connect_delay;
variable_2: output_pin_transition | connect_delay;
\end{verbatim}

The cell degradation table template allows only one-dimensional tables:

\begin{verbatim}
variable_1: input_net_transition
\end{verbatim}

Following are the rules for the relationship between variables and indexes:

- If you have \texttt{variable}_1, you must have \texttt{index}_1.
- If you have \texttt{variable}_1 and \texttt{variable}_2, you must have \texttt{index}_1 and \texttt{index}_2.
- If you have \texttt{variable}_1, \texttt{variable}_2, and \texttt{variable}_3, you must have \texttt{index}_1, \texttt{index}_2, and \texttt{index}_3.

**CMOS Nonlinear Timing Model Examples**

\begin{verbatim}
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*);
}
\end{verbatim}

**Syntax**

```plaintext
calc_mode : namestring:

name

The name of the associated process mode.

domain Group

In the case of a piecewise lookup table, use one or more domain groups in the lu_table_template group to specify subsets of the lookup table template. Variables in a domain group can be the variables in the lu_table_template group.
Syntax

library (namestring) {
  lu_table_template (template_namestring) {
    domain(domain_1_nameid) {
      ... domain description ...
    }
  }
}

domain_1_name

A string representing the name of the domain.

Simple Attributes

calc_mode
variable_1
variable_2
variable_3

Complex Attributes

index_1
index_2
index_3

calc_mode Simple Attribute

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

Example

calc_mode : OC1;

variable_1, variable_2, and variable_3 Simple Attributes

The variables in a domain group are a subset of the variables in the
lu_table_template group.

1.9.20 maxcap_lut_template Group

The maxcap_lut_template group defines a template for specifying the maximum
acceptable capacitance of an input or an output pin.

Syntax

library (namestring) {
  maxcap_lut_template (template_nameid) {
    ... template description ...
  }
}

Simple Attributes

variable_1
variable_2

Complex Attributes

index_1
index_2

variable_1 and variable_2 Simple Attributes
The value you can assign to variable_1 is frequency. The value you can assign to variable_2 is input_transition_time.

index_1 and index_2 Complex Attributes

Along with variable_1 and variable_2, you must specify the index values.

index_1 ("float, ..., float");
index_2 ("float, ..., float");

Example

library (my_library) {
  ...
  maxcap_lut_template (my_template) {
    variable_1: frequency;
    variable_2: input_transition_time;
    index_1 ("100.0000, 200.0000");
    index_2 ("0.0, 0.0");
  }
  ...
}

1.9.21 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 (namestring) {
  maxtrans_lut_template (template_nameid) {
    ... 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

The value you can assign to variable_1 is frequency. The value you can assign to variable_2 is input_transition_time. The value you can assign to variable_3 is total_output_net_capacitance.

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) {
  ...
  maxtrans_lut_template (my_template) {
    variable_1: frequency;
    variable_2: input_transition_time;
    variable_3: total_output_net_capacitance;
    index_1 ("100.0000, 200.0000");
    index_2 ("0.0, 0.0");
    index_3 ("0.0, 0.0");
  }
  ...
}

1.9.22 noise_lut_template Group

The noise_lut_template group defines a template for specifying a noise immunity curve. Use the template to specify the input noise width output load and breakpoints that represent the input height table.

Syntax

library (namestring) {
  noise_lut_template (template_nameid) {
    ...
    template description ...
  }
}

Simple Attributes

variable_1
variable_2

Complex Attributes

index_1
index_2

variable_1 and variable_2 Simple Attributes

The values you can assign to variable_1 and variable_2 are input_noise_width and total_output_net_capacitance.

index_1 and index_2 Complex Attributes

Along with variable_1 and variable_2, you must define both index_1 and index_2.

index_1 ("float, ..., float");
index_2 ("float, ..., float");

Example

library (my_library) {
  ...
  noise_lut_template (my_template) {
    variable_1: input_noise_width;
    variable_2: total_output_capacitance;
    index_1 ("0, 0.1, 2");
    index_2 ("0, 2");
  }
  ...
}

1.9.23 normalized_driver_waveform Group

The library-level normalized_driver_waveform group represents a collection of driver
waveforms under various input slew values. The \texttt{index\_1} specifies the input slew and \texttt{index\_2} specifies the normalized voltage. Note that the slew index in the \texttt{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.

\textbf{Simple Attributes}

\begin{itemize}
  \item \texttt{driver\_waveform\_name}
\end{itemize}

\textbf{Complex Attributes}

\begin{itemize}
  \item \texttt{index\_1}
  \item \texttt{index\_2}
  \item \texttt{values}
\end{itemize}

\textbf{Syntax}

\begin{verbatim}
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");
}
\end{verbatim}

\textbf{Example}

\begin{verbatim}
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*/
}
\end{verbatim}

\textbf{driver\_waveform\_name Simple Attribute}

The \texttt{driver\_waveform\_name} string attribute differentiates the driver waveform table from other driver waveform tables when multiple tables are defined. Cell-specific and rise- and fall-specific driver waveform usage modeling depend on this attribute.

The \texttt{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 \texttt{.db} file.

\textbf{Syntax}

\begin{verbatim}
driver\_waveform\_name : string;
\end{verbatim}

\textbf{Example}

\begin{verbatim}
normalized\_driver\_waveform (waveform\_template) {
  driver\_waveform\_name : clock\_driver; \
  index\_1 ("0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75");
  index\_2 ("0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
  values ("0.012, 0.03, 0.045, 0.06, 0.075, 0.090, 0.105, 0.13, 0.145", \ 
  ... \ 
  "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9");
}
\end{verbatim}
Use this group to define operating conditions; that is, process, voltage, and temperature. You define an operating_conditions group at the library-level, as shown here:

**Syntax**

```
library (namestring) {
    operating_conditions (namestring) {
        ... operating conditions description ...
    }
}
```

**Simple Attributes**

- `calc_mode`: name_id
- `parameter`: float; `i`: i = 1..5
- `process`: float
- `temperature`: float
- `tree_type`: valueenum
- `voltage`: float

**Complex Attribute**

```
power_rail (string, float); /* one or more */
```

**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 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.

**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.

power_rail Complex Attribute

Use the power_rail attribute in the operating_conditions group to specify a voltage value for each power supply.

Syntax

    power_rail (power_supply_name_string, voltage_value_float) :

    power_supply_name

    Specifies a power supply name that can be used later for reference. You can refer to it by assigning a string to the rail connection input_signal_level or the output_signal_level attribute.

    voltage_value

    Identifies the voltage value associated with the power_supply_name. The value is specified by the units you define in the library group voltage_unit attribute.

Example

    operating_conditions (MPSS) {
        calc_mode : worst;
        process : 1.5;
        temperature : 70;
        voltage : 4.75;
        tree_type : worse_case_tree;
        power_rail (VDD1, 4.8);
        power_rail (VDD2, 2.9);
    }

1.9.25 output_current_template Group

Use the output_current_template group to describe a table template for composite
current source (CCS) modeling.

**Syntax**

```liberty
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 index_3 Complex Attributes**

Along with `variable_1` and `variable_2`, you must specify the index values.

- `index_1`: ("float, ...", float);
- `index_2`: ("float, ...", float);

**Example**

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

**1.9.26 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 1-5 lists the predefined variables you can use in an output_voltage expression attribute. Separate variables are defined for CMOS and BiCMOS.

<table>
<thead>
<tr>
<th>CMOS/BiCMOS variable</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD</td>
<td>5V</td>
</tr>
<tr>
<td>VSS</td>
<td>0V</td>
</tr>
<tr>
<td>VCC</td>
<td>5V</td>
</tr>
</tbody>
</table>

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

All voltage values are in the units you define with the voltage_unit attribute within the library group.

Example 1-7 shows an example of an output_voltage group.

Example 1-7 output_voltage Group

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

1.9.27 part Group

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

Syntax

```liberty
library (name_string) {
  part(name_string) {
    ...device description...
  }
  part(name_string) {
```
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**

```plaintext
default_step_level : "nameid" ;

"name"
```

An alphanumeric string identifier, enclosed within double quotation marks, representing the default step level for the device.

**Example**

```plaintext
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**

```plaintext
fpga_isd : fpga_isd_nameid ;

fpga_isd_nameid
```

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

**Example**

```plaintext
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.

**Syntax**

```
num_luts : value_int;
```

**value**

An integer representing the number of lookup tables on the FPGA device.

**Example**

```
num_luts : 1280;
```
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;

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;

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, value);

resource_name

The name of the resource being constrained.

value

An integer representing the maximum constraint of the resource.

Example

max_count (BUGFGTS, 4);

valid_speed_grade Complex Attribute

Use the valid_speed_grade attribute to specify the various speed grades for the FPGA device.

Syntax

valid_speed_grade ("name_1", "name_2", ..., "name_n");
A list of alphanumeric string identifiers, each enclosed within double quotation marks, represents the various speed grades for the device. Each identifier corresponds to an operating condition under which the library is characterized and under which the device is used.

Example

```plaintext
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**

```plaintext
valid_step_levels ("name_1id", "name_2id", ..."name_nid");
```

A list of alphanumeric string identifiers, each enclosed within double quotation marks, representing various step levels for the device. Each identifier corresponds to an operating condition under which the library is characterized and under which the device is used.

Example

```plaintext
valid_step_levels ("STEP0", "STEP1", "STEP2");
```

**speed_grade Group**

The `speed_grade` group associates a valid speed grade with a valid step level.

**Syntax**

```plaintext
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

```plaintext
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**

\[
\text{fpga\_isd : fpga\_isd\_name}_{id};
\]

\[
\text{fpga\_isd\_name}
\]

The name of a library-level fpga_isd group.

**Example**

\[
fpga\_isd : \text{part\_cell\_isd};
\]

**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**

\[
\text{step\_level (name}_{string});
\]

\[
\text{name}
\]

The alphanumeric identifier for a valid step level.

**Example**

\[
\text{step\_level () ;}
\]

1.9.28 **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**

\[
\text{library \{name}_{string}\{}
\]

\[
\text{pg\_current\_template \{template\_name}_{id}\{}
\]

\[
\text{... template description ...}
\]

\[
\}
\]

**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.

```plaintext
index_1 ("float, ...")
index_2 ("float, ...")
index_3 ("float, ...")
index_4 ("float, ...")
```

Example

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

1.9.29 poly_template Group

Use the `poly_template` group to define a template of polynomials to be used by the `timing` group. For reference purposes, the `poly_template` group requires a name.

Syntax

```plaintext
library (library_nameid) {
...
poly_template(poly_template_nameid) {
  variables(variable_i Enum, ..., variable_n Enum);
  variable_1_range(min_valuefloat, max_valuefloat);
  ...
  variable_n_range(min_valuefloat, max_valuefloat);
  mapping(value Enum, power_rail_nameid);
  orders();
  domain(domain_nameid) {
    calc_mode : nameid;
    variables(variable_i Enum, ..., variable_n Enum);
    variable_1_range(min_valuefloat, max_valuefloat);
    ...
    variable_n_range(min_valuefloat, max_valuefloat);
    mapping(value Enum, power_rail_nameid);
    orders();
  }
}
poly_template_name
```

Complex Attributes

- `variables`
- `variable_n_range`
- `mapping`
variables Complex Attribute

Use the variables attribute to specify the name of a variable or a list of the variables that characterizes library cells for timing, noise immunity, and noise propagation. The variable or variables are used in the polynomial equations.

Note:

At least one variable name is required.

Syntax

variables(variable_1enum,...,variable_nenum);

variable_1,...,variable_n

The values depend on the group as shown in the following.

The valid variables for a noise immunity template referenced by a noise immunity polynomial, such as noise_immunity_high, are:

input_noise_width | total_output_net_capacitance | voltage
| voltage_i | temperature | parameter

The valid variables for a noise propagation template referenced by a noise propagation group, such as propagated_noise_height_high, are:

input_noise_height | input_noise_width |
input_noise_time_to_peak | total_output_net_capacitance |
voltage, voltage_i, temperature | parameter

The valid variables in a steady state group, such as steady_state_current_high, are:

iv_output_voltage | voltage | voltage_i | temperature |
parameter

The valid variables in a timing group are generally divided into four sets:

- Set 1
  - input_net_transistion | constrained_pin_transition
- Set 2
  - total_output_net_capacitance | output_net_length,
output_net_wire_cap | output_net_pin_cap |
related_pin_transition
- Set 3
  - related_out_total_output_net_capacitance |
related_out_output_net_length |
related_out_output_net_wire_cap |
related_out_output_net_pin_cap
- Set 4
  - temperature, voltage | voltage_i

In Set 4, voltage is the default power supply voltage for the design and voltage_i is normally used only when your design requires dual power supplies for the cell; for example, for a level shifter.
For a one-dimensional polynomial, substitute the values in Sets 1 and 2 for \textit{variable}_1. For a two-dimensional polynomial, substitute the values in Sets 1, 2, and 4 for \textit{variable}_1 and \textit{variable}_2. For polynomials with three or more dimensions, substitute the values in Sets 1, 2, 3, and 4 for \textit{variable}_1, \textit{variable}_2, ..., \textit{variable}_n.

\textbf{Example}

\begin{verbatim}
variables(temperature, voltage,
          total_output_net_capacitance);
\end{verbatim}

\textbf{variable}_n\_range Complex Attribute

Use the \textit{variable}_n\_range attribute to specify the range of the value for the \textit{n}th variable in the \textit{variables} attribute.

\textbf{Note:}

A \textit{variable}_n\_range attribute is required for each variable listed in the \textit{variables} attribute.

\textbf{Syntax}

\begin{verbatim}
variable_n_range(float, float);
\end{verbatim}

\begin{itemize}
  \item \textit{float, float}
\end{itemize}

Floating-point number pairs that specify the value range.

\textbf{Example}

\begin{verbatim}
variable_2_range(1.5, 2.0);
\end{verbatim}

\textbf{mapping Complex Attribute}

The \textit{mapping} attribute specifies the relationship between \textit{voltage} attribute (as used in the polynomial) and the corresponding \textit{power_rail} attribute defined in the \textit{power_supply} attribute.

\textbf{Note:}

You can have no more than two \textit{mapping} attributes in a \textit{poly_template} group.

\textbf{Syntax}

\begin{verbatim}
mapping(valueenum, power_rail_nameid);
\end{verbatim}

\begin{itemize}
  \item \textit{valueenum, power_rail_nameid}
\end{itemize}

Valid values are \textit{voltage} and \textit{voltage1}.

\begin{itemize}
  \item \textit{power_rail_nameid}
\end{itemize}

Identifies the corresponding power rail.

\textbf{Example}

\begin{verbatim}
mapping(voltage, VDD2);
\end{verbatim}

\textbf{orders Complex Attribute}

Use the \textit{orders} attribute to specify the order for the variables for the polynomial.

\textbf{Syntax}

\begin{verbatim}
variable_n_range(float, float);
\end{verbatim}

\begin{itemize}
  \item \textit{float, float}
\end{itemize}
Example

```plaintext
variable_2_range(1.5, 2.0);
```

**domain Group**

In the case of a piecewise polynomial, use one or more `domain` groups in the `poly_template` group. The variables in a `domain` group are the same as the variables in the `poly_template` group.

**Note:**

A domain name is required and the name must be unique.

**Syntax**

```plaintext
library (library_nameid) {
  poly_template (poly_template_nameid) {
    ...
    domain (domain_nameid) {
      ...
    }
  }
}
```

**Simple Attribute**

`calc_mode`

**Complex Attributes**

- `variables`
- `variable_n_range`
- `mapping`
- `orders`

**calc_mode Simple Attribute**

Use the `calc_mode` attribute to specify the associated process mode.

**Syntax**

```plaintext
calc_mode : nameid;
```

`name`

The name of the associated process mode.

**Example**

```plaintext
calc_mode : best;
```

**variables, variable_n_range, mapping, and orders Complex Attributes**

For the description of how each of these attributes is used, see the "`poly_template Group`".

**Example**

```plaintext
domain (D1) {
  calc_mode : best;
  variables (temperature, voltage,
           total_output_net_capacitance);
  variable_1_range (1.5, 2.0);
  variable_2_range (1.0, 2.0);
}
1.9.30  power_lut_template Group

The `power_lut_template` group is defined within the `library` group, as shown here:

**Syntax**

```plaintext
library (name) {
    power_lut_template (template_name) {
        variable_1 : input_transition_time |
        total_output_net_capacitance |
        equal_or_opposite_output_net_capacitance ;
        variable_2 : input_transition_time |
        total_output_net_capacitance |
        equal_or_opposite_output_net_capacitance ;
        variable_3 : input_transition_time |
        total_output_net_capacitance |
        equal_or_opposite_output_net_capacitance ;
        index_1 ("float, ..., float") ;
        index_2 ("float, ..., float") ;
        index_3 ("float, ..., float") ;
    }
}
```

**Simple Attributes**

- `variable_1`
- `variable_2`
- `variable_3`

**Complex Attributes**

- `index_1`
- `index_2`
- `index_3`

**Group**

- `domain`

The `power_lut_template` group creates a template of the index used by the `internal_power` group (defined in a `pin` group within a cell).

The name of the template (`template_name`) is a name you choose that can be used later for reference.

**Note:**

A `power_lut_template` with the name `scalar` is predefined; its size is 1. You can refer to it by entering `scalar` as the name of a `fall_power` group, `power` group, or `rise_power` group within the `internal_power` group (defined in the `pin` group).

**variable_1, variable_2, and variable_3 Simple Attributes**

The `variable_1` attribute in the `power_lut_template` group specifies the first dimensional variable used by the library developer to characterize cells in the library for internal power.

The `variable_2` attribute in the `power_lut_template` group specifies the second dimensional variable the library developer uses to characterize cells in the library for internal power.

The `variable_3` attribute in the `power_lut_template` group specifies the third dimensional variable the library developer uses to characterize cells in the library for internal power.
internal power.

If the \texttt{index}_1 attribute is measured with the loading of the output net capacitance of the pin specified in the \texttt{pin} group that contains the \texttt{internal_power} group (defined in a \texttt{cell} group), the value for \texttt{variable}_1 is \texttt{equal_or_opposite_output_net_capacitance}.

If the \texttt{index}_1 attribute is measured with the input transition time of the pin specified in the \texttt{pin} group or the \texttt{related_pin} attribute of the \texttt{internal_power} group, the value for \texttt{variable}_1 is \texttt{input_transition_time}.

If the \texttt{index}_2 attribute is measured with the loading of the output net capacitance of the pin specified in the \texttt{pin} group that contains the \texttt{internal_power} group (defined in a \texttt{cell} group), the value for \texttt{variable}_2 is \texttt{equal_or_opposite_output_net_capacitance}.

If the \texttt{index}_2 attribute is measured with the input transition time of the pin specified in the \texttt{pin} group or the \texttt{related_pin} attribute of the \texttt{internal_power} group, the value for \texttt{variable}_2 is \texttt{input_transition_time}.

If the \texttt{index}_3 attribute is measured with the loading of the output net capacitance of the pin specified in the \texttt{pin} group that contains the \texttt{internal_power} group (defined in a \texttt{cell} group), the value for \texttt{variable}_3 is \texttt{equal_or_opposite_output_net_capacitance}.

If the \texttt{index}_3 attribute is measured with the input transition time of the pin specified in the \texttt{pin} group or the \texttt{related_pin} attribute of the \texttt{internal_power} group, the value for \texttt{variable}_3 is \texttt{input_transition_time}.

\texttt{index}_1, \texttt{index}_2, \texttt{index}_3 Complex Attributes

The \texttt{index}_1 complex attribute in the \texttt{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 monotonically increasing order. You can overwrite the \texttt{index}_1 attribute by providing the same attribute in the \texttt{fall_power} group, \texttt{power} group, or \texttt{rise_power} group within the \texttt{internal_power} group (defined in the \texttt{pin} group). The \texttt{index}_1 attribute is required in the \texttt{power_lut_template} group.

The \texttt{index}_2 complex attribute in the \texttt{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 \texttt{index}_2 attribute by providing the same attribute in the \texttt{fall_power} group, \texttt{power} group, or \texttt{rise_power} group within the \texttt{internal_power} group (defined in the \texttt{pin} group). The \texttt{index}_2 attribute is required in the \texttt{power_lut_template} group if the \texttt{variable}_2 attribute is present.

The \texttt{index}_3 complex attribute in the \texttt{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 \texttt{index}_3 attribute in the \texttt{internal_power} group by providing the same attribute in the \texttt{fall_power} group, \texttt{power} group, or \texttt{rise_power} group within the \texttt{internal_power} group (defined in the \texttt{pin} group). The \texttt{index}_3 attribute is required in the \texttt{power_lut_template} group if the \texttt{variable}_3 attribute is present.

\texttt{Example 1-8} shows four \texttt{power_lut_template} groups.

\texttt{Example 1-8 Four power_lut_template Groups}

\begin{verbatim}
power_lut_template (output_by_cap) { 
variable_1 : total_output_net_capacitance;
index_1 (*0.0, 5.0, 20.0*) ;
}
power_lut_template (output_by_cap_and_trans) { 
variable_1 : total_output_net_capacitance;
variable_2 : input_transition_time;
index_1 (*0.0, 5.0, 20.0*) ;
index_2 (*0.1, 1.0, 5.0*) ;
}
power_lut_template (input_by_trans) { 
variable_1 : input_transition_time;
index_1 (*0.0, 1.0, 5.0*) ;
}
power_lut_template (output_by_cap2_and_trans) { 
variable_1 : total_output_net_capacitance;
variable_2 : input_transition_time;
}
\end{verbatim}
Use the `report_lib -timing` command to report `power_lut_template` information in the library. You can do this only while reading in the library source file in the same session. The report will resemble the one shown in Example 1-9.

### Example 1-9  `power_lut_template` Report

**Internal Power Lookup Template:**

<table>
<thead>
<tr>
<th>Template_name</th>
<th>variable_1</th>
<th>variable_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_by_cap</td>
<td>total_output_net_capacitance</td>
<td></td>
</tr>
<tr>
<td>index_1</td>
<td>0.0000</td>
<td>5.0000</td>
</tr>
<tr>
<td></td>
<td>20.0000</td>
<td></td>
</tr>
<tr>
<td>output_by_cap_and_trans</td>
<td>total_output_net_capacitance input_transition_time</td>
<td></td>
</tr>
<tr>
<td>index_1</td>
<td>0.0000</td>
<td>5.0000</td>
</tr>
<tr>
<td></td>
<td>20.0000</td>
<td></td>
</tr>
<tr>
<td>index_2</td>
<td>0.1000</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td>5.0000</td>
<td></td>
</tr>
<tr>
<td>input_by_trans</td>
<td>input_transition_time</td>
<td></td>
</tr>
<tr>
<td>index_1</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td>5.0000</td>
<td></td>
</tr>
<tr>
<td>index_2</td>
<td>0.1000</td>
<td>0.5000</td>
</tr>
<tr>
<td></td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>index_3</td>
<td>0.1000</td>
<td>0.5000</td>
</tr>
<tr>
<td></td>
<td>1.0000</td>
<td></td>
</tr>
</tbody>
</table>

**domain Group**

In the case of a piecewise polynomial, use one or more `domain` groups in the `power_lut_template` group. The variables in a `domain` group are the same as the variables in the `power_lut_template` group. For more information about using a `domain` group, see "domain Group".

### 1.9.31  `power_poly_template` Group

Use the `power_poly_template` group to define a template of polynomials to be used by the `timing` group. For reference purposes, the `power_poly_template` group requires a name.

**Syntax**

```plaintext
library (library_nameid) {
...
    power_poly_template(power_poly_template_nameid) {
        variables(variable_1 Enum ... variable_n Enum) ;
        variable_1_range(min_value float, max_value float) ;
        ...
        variable_n_range(min_value float, max_value float) ;
        mapping(value Enum, power_rail_nameid) ;
        domain(domain_nameid) {
            calc_mode : nameid ;
            variables(variable_1 Enum ... variable_n Enum) ;
            variable_1_range(min_value float, max_value float) ;
            ...
            variable_n_range(min_value float, max_value float) ;
            mapping(value Enum, power_rail_nameid) ;
        }
    }
}
```

**power_poly_template_name**

The name of the template.
variables
variable_n_range
mapping

Group
domain

variables Complex Attribute

Use the variables attribute to specify the name of a variable or a list of the variables that characterize library cells for power; that is, the variables used in the polynomial equations.

Note:
At least one variable name is required. A maximum of seven variables is allowed.

Syntax
variables(variable_1enum,..., variable_nenum);

variable_1 , ..., variable_n

Valid values for polynomial variables are:
equal_or_opposite_output_net_capacitance, input_net_transition,
total_output_net_capacitance, output_net_length, temperature, voltage,
voltage_i, and parameter_i.

Example
variables(temperature, voltage,
total_output_net_capacitance);

variable_n_range Complex Attribute

Use the variable_n_range attribute to specify the range of the value for the n-th variable in the variables attribute.

Note:
A variable_n_range attribute is required for each variable listed in the variables attribute.

Syntax
variable_n_range(value_1float, value_2float);

value_1 , value_2

Floating-point number pairs that specify the value range.

Example
variable_2_range(1.5, 2.0);

mapping Complex Attribute

The mapping attribute specifies the relationship between the voltage attribute (as used in the polynomial) and the corresponding power_rail attribute defined in the power_supply attribute.

Note:
You can have no more than two mapping attributes in a power_poly_template group.
Syntax

mapping(value, enum power_rail_name);

value

Valid values are voltage and voltage1.

power_rail_name

Identifies the corresponding power rail.

Example

mapping(voltage, VDD2);

domain Group

In the case of a piecewise polynomial, use one or more domain groups in the poly_template group. The variables in a domain group are the same as the variables in the poly_template group.

Note:

A domain name is required and the name must be unique.

Syntax

library (library_nameid) {
  power_poly_template (power_poly_template_nameid) {
    ...
    domain (domain_nameid) {
      ...
    }
  }
}

Simple Attribute

calc_mode

Complex Attributes

variables
variable_n_range
mapping

calc_mode Simple Attribute

Use the calc_mode attribute to specify the associated process mode.

Syntax

calc_mode : name_id;

name

The name of the associated process mode.

variables, variable_n_range, and mapping Complex Attributes

For the description of how each of these attributes is used, see the "poly_template Group".

1.9.32 power_supply Group
The `power_supply` group is defined in the `library` group, as shown here:

```plaintext
library (name) {
  power_supply {name} {
    default_power_rail : string;
    power_rail(power_supply_name:string, voltage_value:float)
    power_rail(power_supply_name:string, voltage_value:float)
    ...
  }
}
```

**Simple Attribute**

`default_power_rail`

**Complex Attribute**

`power_rail`

The `power_supply` group captures all nominal information on voltage variation.

`default_power_rail Simple Attribute`

The `default_power_rail` attribute receives, by default, the value of the `voltage` attribute defined in the nominal `operating_conditions` group.

**Syntax**

`default_power_rail : power_supply_name:string ;`

`power_supply_name`

An identifier for the power supply.

**Example**

`default_power_rail : VDD0 ;`

`power_rail Complex Attribute`

The `power_rail` attribute identifies all power supplies that have the nominal operating conditions (defined in the `operating_conditions` group) and the nominal voltage values. The `power_supply` group can define one or more `power_rail` attributes.

**Syntax**

`power_rail (power_supply_name:string, voltage_value:float)`

`power_supply_name`

Specifies a power supply name that can be used later for reference. You can refer to it by assigning a string to the rail connection `input_signal_level` or `output_signal_level` attribute.

`voltage_value`

A floating-point number that identifies the voltage value associated with the `power_supply_name`. The value is in the units you define within the `library` group `voltage_unit` attribute.

**Example**

`power_rail (VDD1, 5.0) ;`
Example 1-10 shows a library containing a power_supply group.

Example 1-10   Library Example With power_supply Group

```liberty
class {multiple_power_supply} { 
  power_supply() 
  { /* Define before operating conditions and cells. */
    default_power_rail: VDD0;
    power_rail (VDD1, 5.0);
    power_rail (VDD2, 3.3);
    ... 
  } } 
```

Multiple Power Supply Library Requirements

Use the `report_lib -power` command to report multiple power supply information in the library. The report will resemble the one shown in Example 1-11.

Example 1-11   Multiple Power Supply Report

```
Power Supply Group:

  default_power_rail : VDD0
  power_rail Value
  -------------------
  VDD1 5.00
  VDD2 3.30

Power Rail in the Operating Conditions Groups:

Name : MPSS
  power_rail Value
  -------------------
  VDD1 4.80
  VDD2 2.90

Power Rail Connection Information:

Name : IBUF1
  RAIL CONNECTION (PV1): VDD1
  RAIL CONNECTION (PV2): VDD2

Signal Level Information:

CELL (IBUF1):
  INPUT_SIGNAL_LEVEL (A): VDD1
  END_PIN A;
  INPUT_SIGNAL_LEVEL (EN): VDD1
  END_PIN EN;
  OUTPUT_SIGNAL_LEVEL (Z): VDD2
  END_PIN Z;
  INPUT_SIGNAL_LEVEL (Z1): VDD1
  OUTPUT_SIGNAL_LEVEL (Z1): VDD2
  END_PIN Z1;
END_CELL IBUF1;
```

1.9.33 propagation_lut_template Group

The propagation_lut_template group defines a template for specifying noise propagation through a cell.

Syntax

```liberty
library (name_string) {
  propagation_lut_template (template_name_string) {
    ... 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

The values you can assign to variable_1 and variable_2 are input_noise_width, input_noise_height, and total_output_net capacitance.

index_1, index_2, and index_3 Complex Attributes

Along with variable_1, variable_2, and variable_3, you must define index_1, index_2, and index_3

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

Example

library (my_library) {
... propagation_lut_template (my_template) {
  variable_1 : input_noise_width;
  variable_2 : input_noise_height;
  variable_3 : total_output_net capacitance;
  index_1 ("0.01, 0.2, 2");
  index_2 ("0.2, 0.8");
  index_3 ("0, 2");
}
...
}

1.9.34 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, as shown in the following examples.

**Example 1-12** shows an example of the rise_net_delay group.

**Example 1-12 rise_net_delay Group**

```plaintext
rise_net_delay (net_delay_template_table) {
    index_1 (*0, 1, 2*)
    values (*0.00, 0.21*, *0.11, 0.23*)
}
```

See also **fall_net_delay Group**.

1.9.35 rise_transition_degradation Group

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

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

**Complex Attributes**

- `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 1-13** and shows an example of the rise_transition_degradation group.

**Example 1-13 rise_transition_degradation Group**

```plaintext
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"**.

1.9.36 scaled_cell Group

You define a scaled_cell group within the library group to supply an alternative set of
values for an existing cell, based on the set of operating conditions used.

Operating conditions are defined in the "operating_conditions Group".

Syntax

```plaintext
scaled_cell (existing_cell, operating_conditions_group) {
    ... scaled cell description ...
}
```

**existing_cell**

The name of a cell defined in a previous cell group.

**operating_conditions_group**

The library-level operating_conditions group with which the scaled cell is associated.

**Simple Attributes**

The following attributes are also defined in the cell group.

```plaintext
area : float;
auxiliary_pad_cell : true | false;
bus_naming_style : "string";
cell_footprint : footprint_typestring;
cell_leakage_power : float;
clock_gating_integrated_cell : string_value;
contention_condition : "Boolean expression";
dont_fault : sa0 | sa1 | sa01;
dont_touch : true | false;
dont_use : true | false;
geometry_print : string;
handle_negative_constraint : true | false;
is_clock_gating_cell : true | false;
map_only : true | false;
pad_cell : true | false;
pad_type : clock;
pin_equal ("name_list string");
pin_opposite ("name_list1 string","name_list2 string");
rail_connection (connection_name string,
    power_supply_name string);
scaling_factors : group_name;
single_bit_degenerate : string;
/* black box, bus, and bundle cells only*/
use_for_size_only : true | false;
vhdl_name : "string";
```

**Complex Attributes**

The following attributes are also defined in the cell group.

```plaintext
pin_equal ("name_list string");
pin_opposite ("name_list1 string","name_list2 string");
rail_connection (connection_name string,
    power_supply_name string);
```

**Group Statements**

The following groups are also defined in the cell group.

```plaintext
bundle
bus
ff
ff_bank
generated_clock
latch
latch_bank
leakage_power
lut
mode_definition
pin
```
1.9.37 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 will be 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

1.9.38 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);
```

1.9.39 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");
}

1.9.40 scaling_factors Group

A scaling_factors group is defined within the library group.

Syntax

library ("namestring") {
    scaling_factors ("name_string") {
        ... scaling factors ... 
    }
}

Simple Attributes

The scaling_factors group uses the simple scaling attributes (that is, those with the k_prefix) included in Example 1-1.

1.9.41 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 ... 
            }
        }
    }
}

1.9.42 timing_range Group

Use the timing_range group to specify scaling factors that control signal arrival time.

Syntax

library ("name_string") {
    timing_range (name_string) {
        ... timing range description ... 
    }
}

Simple Attributes

faster_factor
slower_factor
**faster_factor Simple Attribute**

Use this attribute to specify a scaling factor to apply to the signal arrival time to model the fastest-possible arrival time.

**Syntax**

```plaintext
faster_factor : valuefloat;
```

**value**

A floating-point number representing the scaling factor.

**Example**

```plaintext
faster_factor: 0.0;
```

**slowest_factor Simple Attribute**

Use this attribute to specify a scaling factor to apply to the signal arrival time to model the slowest-possible arrival time.

**Syntax**

```plaintext
slowest_factor : valuefloat;
```

**value**

A floating-point number representing the scaling factor.

**Example**

```plaintext
slowest_factor: 0.0;
```

### 1.9.43 type Group

If your library contains bused pins, you must define type groups and define the structural constraints of each bus type in the library. The type group is defined at the library group level, as shown here:

**Syntax**

```plaintext
library (namestring) {
  type (name) {
    ... type description ...
  }
}
```

**name**

Identifies the bus type.

**Simple Attributes**

```plaintext
  base_type : base;
  bit_from : integer;
  bit_to : integer;
  bit_width : integer;
  data_type : data;
  downto : true | false;
```

**base_type : base ;**

Only the array base type is supported.
bit_from : integer ;
    Indicates the member number assigned to the most significant bit (MSB) of successive array members. The default is 0.

bit_to : integer ;
    Indicates the member number assigned to the least significant bit (LSB) of successive array members. The default is 0.

bit_width : integer ;
    Designates the number of bus members. The default is 1.

data_type : data ;
    Indicates that only the bit data type is supported.

downto : true | false ;
    A true value indicates that member number assignment is from high to low. A false value indicates that member number assignment is from low to high.

Example 1-14 illustrates a type group statement.

Example 1-14 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 1-15 Alternative type Group Descriptions

Example 1-15 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.

1.9.44 user_parameters Group

Use the user_parameters group to specify default values for up to five user-defined process variables.

You 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;

1.9.45 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. The `create_wire_load` command generates these values automatically.

**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**

```liberty
library (example)
...
wire_load (small) {
  area : 0.0;
  capacitance : 1.0;
  resistance : 0.0;
  slope : 0.0;
  fanout_length (1,1.68);
}
```

```liberty
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");
  }
}
```

### 1.9.46 wire_load_selection Group

A `wire_load_selection` group is defined in a `library` group, as follows.

**Syntax**

```liberty
library (name) {
  wire_load_selection (name) {
    ... wire_load_selection criteria ...
  }
}
```

**Complex Attribute**

`wire_load_from_area (float, float, string)`

**Example**

```liberty
wire_load_selection (normal) {
  wire_load_from_area (100, 200, average);
}
```

### 1.9.47 wire_load_table Group

A `wire_load_table` group is defined in a `library` group, as follows.

**Syntax**

```liberty
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 separate 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 separate 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.

2.1 cell Group

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

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

2.1.1 Attributes and Values

Example 2-1 lists alphabetically all the attributes and groups that you can define within a cell group.

Example 2-1 Attributes and Values for a cell Group

/* Simple Attributes for cell group */

area : float;
auxiliary_pad_cell : true | false;
base_name : cell_base_name string;
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 | sai | sa01;
dont_touch : true | false;
dont_use : true | false;
driver_type : name_id;
edif_name : name_id;
em_temp_degradation_factor : value float;
fpga_domain_style : name_id;
geometry_print : string; /* bus and bundle cells only */
handle_negative_constraint : true | false;
interface_timing : true | false;
io_type : name_id;
is_clock_gating_cell : true | false;
map_only : true | false;
pad_cell : true | false;
pad_type : clock;
power_cell_type : ;
pREFERRED : true | false;
scaling_factors : group_name;
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;
vhdl_name : "string";

/* Complex Attributes for cell Group */

pin_equal ("name_list string");
pin_opposite ("name_list1 string", "name_list2 string");
rail_connection (connection_name string, power_supply_name string);
resource_usage (resource_name id, number_of_resources id);

/* Group Statements for cell Group */

bundle (name string) {
}
bus (name string) {
}
dynamic_current () {
}
ff (variable1 string, variable2 string) {
}
ff_bank (variable1 string, variable2 string, bits integer) {
}
functional_yield_metric () {
}
generated_clock () {
}
intrinsic_parasitic () {
}
latch (variable1 string, variable2 string) {
}
latch_bank (variable1 string, variable2 string, bits integer) {
}
leakage_current () {
}
leakage_power () {
}
lut (name string) {
}
mode_definition () {
}
pin (name string | name_list string) {
}
routing_track (routing_layer_name string) {
}
statetable ("input node names", "internal node names") {
}
test_cell () {
}
type (name string) {
}

Descriptions of the attributes and group statements follow.

2.1.2 Simple Attributes

This section lists, alphabetically, the simple attributes for the cell and model groups.

auxiliary_pad_cell Simple Attribute
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**

```plaintext
base_name : "cell_base_name";
```

*cell_base_name*

An alphanumeric string, enclosed in quotation marks, representing a name for the output cell.

**Example**

```plaintext
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**

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

**Example**

```plaintext
bus_naming_style : "Bus$Pin%d";
```

cell_footprint Simple Attribute

Use the `cell_footprint` attribute to assign a footprint class to a cell.

**Syntax**

```plaintext
cell_footprint : class_name;
```

*class_name*

A character string that represents a footprint class. The string is case-sensitive: And4 is different footprint from and4.

**Example**

```plaintext
cell_footprint : 5MIL;
```

Use this attribute to assign the same footprint class to all cells that have the same geometric layout characteristics.

Cells with the same footprint class are considered interchangeable and can be swapped during in-place optimization. Cells without `cell_footprint` attributes are not swapped during in-place optimization.

When you use `cell_footprint`, you also set the `in_place_swap_mode` attribute to `match_footprint`.

cell_leakage_power Simple Attribute

Use the `cell_leakage_power` attribute to define the leakage power of a cell. If missing or negative, the value of the `default_cell_leakage_power` attribute defined in the library is assumed.
Syntax

```plaintext
cell_leakage_power : value_float :
  value
```

A floating-point number indicating the leakage power of the cell.

Example

```plaintext
cell_leakage_power : 0.2
```

The `cell_leakage_power` attribute is also recognized in the `scaled_cell` group, where it allows you to model nonlinear scaling of leakage power relative to certain process, voltage, and temperature conditions.

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 software uses.

Syntax

```plaintext
clock_gating_integrated_cell: generic|value:

  generic
  When specified, the actual value is determined 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, flip-flop gating logic, and none.
  The second string specifies whether the logic is appropriate for rising- or falling-edge-triggered registers. The options are posedge and negedge.
  The third (optional) string specifies whether you want test control logic located before or after the latch or flip-flop, or not at all. The options for cells set to latch or flip-flop are precontrol (before), postcontrol (after), or no entry. The options for cells set to no gating logic are control and no entry.
  The fourth (optional) string, which exists only if the third string does, specifies whether you want observability logic or not. The options are obs and no entry. Table 2-1 lists some example values for the `clock_gating_integrated_cell` attribute:

<table>
<thead>
<tr>
<th>Value of clock_gating_integrated_cell</th>
<th>Integrated cell must contain</th>
</tr>
</thead>
<tbody>
<tr>
<td>latch_negedge</td>
<td>1. Latch-based gating logic2. Logic appropriate for falling-edge-triggered registers</td>
</tr>
<tr>
<td>latch_posedge_postcontrol</td>
<td>1. Latch-based gating logic2. Logic appropriate for rising-edge-triggered registers3. Test control logic located after the latch</td>
</tr>
<tr>
<td>latch.negedge.precontrol</td>
<td>1. Latch-based gating logic2. Logic appropriate for falling-edge-triggered registers3. Test control logic located before the latch</td>
</tr>
<tr>
<td>none_posedge_control.obs</td>
<td>1. Latch-free gating logic2. Logic appropriate for rising-edge-triggered registers3. Test control logic (no latch)4. Observability logic</td>
</tr>
</tbody>
</table>

For more details about the clock-gating integrated cells, see the "Modeling Power and Electromigration" chapter in the Liberty User Guide, Volume 1.
Setting Pin Attributes for an Integrated Cell

The clock-gating software requires setting the pins of your integrated cells using the attributes listed in Table 2-2. Setting some of the pin attributes, such as those for test and observability, is optional.

<table>
<thead>
<tr>
<th>Integrated cell pin name</th>
<th>Data direction</th>
<th>Required attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>in</td>
<td>clock_gate_clock_pin</td>
</tr>
<tr>
<td>enable</td>
<td>in</td>
<td>clock_gate_enable_pin</td>
</tr>
<tr>
<td>test_mode or scan_enable</td>
<td>in</td>
<td>clock_gate_test_pin</td>
</tr>
<tr>
<td>enable_clock</td>
<td>out</td>
<td>clock_gate_out_pin</td>
</tr>
</tbody>
</table>

Setting Timing for an Integrated Cell

You set both the setup and hold arcs on the enable pin by setting the clock_gate_enable_pin attribute for the integrated cell to true. The setup and hold arcs for the cell are determined by the edge values you enter for the clock_gating_integrated_cell attribute. Table 2-3 lists the edge values and the corresponding setup and hold arcs.

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

contention_condition Simple Attribute

Specifies the contention conditions for a cell. Contention is a clash of "0" and "1" signals. In certain cells, it may be a forbidden condition and cause circuits to short.

Syntax

```
contention_condition : "Boolean expression" ;
```

Example

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

dont_fault Simple Attribute

This attribute is used by test tools. It is a string attribute that you can set on a library cell or pin.

Syntax

```
dont_fault : sa0 | sa1 | sa01 ;
```

```
sa0, sa1, sa01
```

The value you enter determines whether the dont_fault attribute will be placed on stuck at 0 (sa0), stuck at 1 (sa1), or stuck on both faults (sa01).

Example
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**

```plaintext
dont_touch : valueBoolean;
```

**value**

Valid values are true and false.

**Example**

```plaintext
dont_touch : true;
```

In addition to defining `dont_touch` in a cell group or a model group, you can also apply the `dont_touch` attribute to a cell, by using the `set_dont_touch` command.

**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**

```plaintext
dont_use : valueBoolean;
```

**value**

Valid values are true and false.

**Example**

```plaintext
dont_use : true;
```

In addition to defining `dont_use` in a cell group or a model group, you can also apply the `dont_use` attribute to a cell, by using the `set_dont_use` command.

**driver_type Simple Attribute**

Use the `driver_type` attribute to specify the drive power of the output or the I/O cell.

**Syntax**

```plaintext
driver_type : nameid;
```

**name**

An alphanumeric string identifier, enclosed in quotation marks, representing the drive power.

**Example**

```plaintext
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

```plaintext
cell (cell_name) {
  --
  driver_waveform: string;
  driver_waveform_rise: string;
  driver_waveform_fall: string;
}
```

Example

```plaintext
cell (my_cell1) {
  driver_waveform: clock_driver;
  --
}
cell (my_cell2) {
  driver_waveform: bus_driver;
  --
}
cell (my_cell3) {
  driver_waveform_rise: rise_driver;
  driver_waveform_fall: fall_driver;
  --
}
```

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:

- 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.
- Use the cell-specific driver waveform (defined by the `driver_waveform`
  attribute).
- Use the library-level default driver waveform (defined by the
  normalized_driver_waveform table without the driver_waveform_name
  attribute).

The `driver_waveform_rise` attribute can refer to a normalized driver waveform
that is either rising or falling. It is possible to invert the waveform automatically during
runtime if necessary.

Syntax

```plaintext
cell (cell_name) {
  --
  driver_waveform: string;
  driver_waveform_rise: string;
  driver_waveform_fall: string;
}
```

Example

```plaintext
cell (my_cell1) {
  driver_waveform: clock_driver;
  --
}
cell (my_cell2) {
  driver_waveform: bus_driver;
  --
}
```
cell (my_cell3) {
  driver_waveform_rise : rise_driver;
  driver_waveform_fall : fall_driver;
}

edif_name Simple Attribute

Use the edif_name attribute to define the name of the output cell generated by EDIF. If you omit this attribute, the cell is given the name “io_cell_name”.

Syntax

```
edif_name : nameid ;
```

name

An alphanumeric string, enclosed in quotation marks, representing a name for the output EDIF cell.

Example

```
edif_name : "OBUF" ;
```

em_temp_degradation_factor Simple Attribute

The em_temp_degradation_factor attribute specifies the electromigration exponential degradation factor.

Syntax

```
em_temp_degradation_factor : valuefloat ;
```

value

A floating-point number in centigrade units consistent with other temperature specifications throughout the library.

Example

```
em_temp_degradation_factor : 40.0 ;
```

fpga_cell_type Simple Attribute

Specifies whether a tool 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 : valueenum ;
```

value

Valid values are rising_edge_clock_cell and falling_edge_clock_cell.

Example

```
fpga_cell_type : rising_edge_clock_cell ;
```

fpga_domain_style Simple Attribute

Use this attribute to reference a calc_mode value in a domain group in a polynomial table.

Syntax
name

The calc_mode value.

Example

```
fpga_domain_style : "speed";
```

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_nameid;
```

fpga_isd_nameid

The name of the library-level fpga_isd group.

Example

```
fpga_isd : part_cell_isd;
```

generate_print Simple Attribute

The generate_print attribute specifies certain bundle or bus multibit cells so that they can be sized together for layout requirements. At least two cells in the library must have the same name defined by this attribute, such as GP1 in the following example.

Syntax

```
generate_print : cell_nameid;
```

cell_nameid

A character string identifying two or more bus or bundle cells in a library.

Example

```
cell (FDX2) {
  area : 18;
  generate_print : GP1;
  bundle (D) {
    members (D0, D1);
    direction : input;
    ...
    timing () {
      ...
      ...
    }
  };
}
```

cell (FDX4) {
  area : 32;
  generate_print : GP1;
  bundle (E) {
    members (D1, D0);
    direction : input;
    ...
    timing () {
      ...
      ...
    }
  };
}
```

handle_negative_constraint Simple Attribute

You use this attribute during generation of VITAL models to indicate whether a cell needs negative constraint handling. It is an optional attribute for timing constraints in a cell or model group.

**Syntax**

```
handle_negative_constraint : true | false ;
```

**Example**

```
handle_negative_constraint : true ;
```

**interface_timing Simple Attribute**

Indicates that the timing arcs are interpreted according to interface timing specifications semantics. If this attribute is missing or its value is set to false, the timing relationships are interpreted as those of a regular cell rather than according to interface timing specification semantics.

**Syntax**

```
interface_timing : true | false ;
```

The following example shows a cell with `interface_timing` set to true, indicating that interface timing semantics are to be applied.

**Example**

```
interface_timing : true ;
```

**io_type Simple Attribute**

Use the `io_type` attribute to define the I/O standard used by this I/O cell.

**Syntax**

```
io_type : nameid ;
```

**Example**

```
io_type : "LVTTL" ;
```

**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.

**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**

\[ \text{is\_isolation\_cell} : \text{Boolean expression} ; \]

*Boolean expression*

Valid values are true and false.

**Example**

\[ \text{is\_isolation\_cell} : \text{true} ; \]

**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**

\[ \text{is\_level\_shifter} : \text{Boolean expression} ; \]

*Boolean expression*

Valid values are true and false.

**Example**

\[ \text{is\_level\_shifter} : \text{true} ; \]

**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**

\[ \text{level\_shifter\_type} : \text{level\_shifter\_type\_value} ; \]

**Example**

\[ \text{level\_shifter\_type} : \text{HL\_LH} ; \]

**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**

\[ \text{map\_only} : \text{true} | \text{false} ; \]

In addition to defining *map\_only* in a cell group or a model group, you can also apply the *map\_only* attribute to a cell by using the *set\_map\_only* command.
**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.

**Syntax**

```
   pad_type : value ;
```

**Example**

```
   pad_type : clock;
```

**power_cell_type Simple Attribute**

Use the `power_cell_type` attribute to specify the cell type.

**Syntax**

```
   power_cell_type : value_enum ;
```

**value**

Valid values are `stdcell` (standard cell) and `macro` (macro cell).

**Example**

```
   power_cell_type : stdcell;
```
Note:
The `power_gating_cell` attribute has been replaced by the `retention_cell` attribute. See "retention_cell Simple Attribute".

The cell-level `power_gating_cell` attribute specifies that a cell is a power gating cell. A power gating cell has two modes. When functioning in normal mode, the power gating cell functions as a regular cell. When functioning in power-saving mode, the power gating 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_nameid;
```

`power_gating_cell_name`
A string identifying the power gating 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;
```

scaling_factors Simple Attribute

Use the `scaling_factors` attribute to apply to a cell the scaling factor values defined in the `scaling_factors` group at the library level.

Syntax
```
scaling_factors : group_nameid;
```

`group_name`
Name of the set of special scaling factors in a `scaling_factors`
Example 2-2 shows one of these special scaling factors in the library description and cell description.

**Example 2-2  Individual Scaling Factors**

```plaintext
library (example) {
    k_volt_intrinsic_rise : 0.987;
    ...
    scaling_factors(IO_PAD_SCALING) {
        k_volt_intrinsic_rise : 0.846;
        ...
    }
    cell (INPAD_WITH_HYSTERESIS) {
        area : 0;
        scaling_factors : IO_PAD_SCALING;
        ...
    }
    ...
}
```

Example 2-2 defines a scaling factor group called IO_PAD_SCALING that contains scaling factors different from the library-level scaling factors. The `scaling_factors` attribute in the INPAD_WITH_HYSTERESIS cell is set to IO_PAD_SCALING, so all scaling factors set in the IO_PAD_SCALING group are applied to this 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 to link a multibit black box cell with the single-bit version of the cell.

**Syntax**

```
single_bit_degenerate : "cell_name";

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 () {
        ...
    }
```
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 provides a description of the switch cell so that the switch cell does not need to be inferred through the other information specified in the cell.

**Syntax**

```
switch_cell_type : coarse_grain;
```

**Example**

```
switch_cell_type : coarse_grain;
```

threshold_voltage_group Simple Attribute

The optional `threshold_voltage_group` attribute identifies a cell with a category, based on the voltage characteristics of the cell.

**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

Indicates not to infer transparent level-sensitive latch devices from timing arcs defined in the cell. To indicate that transparent level-sensitive latches should be inferred for input pins, use the tlatch group.

**Syntax**

    timing_model_type : "nameid" ;

**Example**

    timing_model_type : "abstracted" ;

use_for_size_only Simple Attribute

You use this attribute to specify the criteria for sizing optimization. You set this attribute on a cell at the library level.

**Syntax**

    use_for_size_only : valueBoolean :

**Example**

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

vhdl_name Simple Attribute

In cell, model, and pin groups, this attribute resolves conflicts of invalid object names when porting from .db to VHDL. Some .db object names might violate the more restrictive VHDL rules for identifiers.

**Note:**

The report_lib -vhdl_name command produces a table of .db names and VHDL name mapping.

**Syntax**

    vhdl_name : "nameid" ;

**Example**

    cell (INV) {
        area : 1 ;
    }
2.1.3 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 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");

rail_connection Complex Attribute

Use the rail_connection attribute to specify the presence of multiple power supplies in the cell. A cell with multiple power supplies contains two or more rail_connection attributes.

Syntax

rail_connection (connection_nameid, power_supply_nameid);

connection_name

A string that specifies a power connection for the pin name.

power_supply_name

A string that specifies the power supply group already defined in the power_supply group at the library level to be used as the value for the power_level attribute in the internal_power group (defined in the pin group).

Example

```liberty
cell (IBUF1) {
  ...
  rail_connection (PV1, VDD1);
  rail_connection (PV2, VDD2);
  ...
}
```

resource_usage Complex Attribute

Use the resource_usage attribute to specify the name and the number of resources the cell uses.

Syntax

resource_usage (resource_nameid, number_of_resourcesint);

resource_name

An alphanumeric identifier that matches the first argument in a max_count attribute in the library. You can specify multiple resource_usage attributes with different resource names.

number_of_resources

An integer representing the number of resources the cell uses.

Example

```liberty
resource_usage (RES1, 1);
```

2.1.4 Group Statements

This section lists, alphabetically, the group statements for the cell and model groups.

cell Group Example

Example 2-3 shows cell definitions that include some of the CMOS cell attributes described so far.

Example 2-3 cell Group Example

```liberty
library (cell_example) {
```
cell (in)
  vhdl_name : "inpad";
  area : 0; /* pads do not normally consume
  internal core area*/
  cell_footprint : 5MIL;
  scaling_factors : IO_PAD_SCALING;
  pin (A) {
    direction : input;
    capacitance : 0;
  }
  pin (Z) {
    direction : output;
    function : "A";
    timing () {...}
  }
}
cell (inverter_med)
  area : 3;
  preferred : true;
  /* tells Design Compiler to 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 */
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 three elements:

- The members complex attribute. It must be declared first in a bundle group.
- All simple attributes that also appear in a pin group.
- The pin group statement (including all the pin group simple and complex attributes, and group statements).

Syntax

```plaintext
library (name_id) {
  cell (name_id) {
    geometry_print : string ;
    single_bit_degenrate : 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, which are described later in this section.

- 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.

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

**Syntax**

\[
\text{capacitance : valuefloat;}
\]

**value**

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

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:

**Example**

```plaintext
bundle (D) {
    members(D0, D1, D2, D3);
    direction : input;
    capacitance : 1;
}
```

direction Simple Attribute

The direction attribute states the direction of member pins in a bundle group.

The direction listed for this attribute should be the same as that given for the pin in the same bundle group (see the bundle Z pin in Example 2-4).

**Syntax**

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

**Example**

In a bundle group, the direction of all pins must be the same. Example 2-4 shows two bundle groups. The first group shows two pins (Z0 and Z1) whose direction is output. The second group shows one pin (D0) whose direction is input.

**Example 2-4  Direction of Pins in bundle Groups**

```plaintext
cell(inv) {
    area : 16;
    cell_leakage_power : 8;
    bundle(Z) {
        members(Z0, Z1, Z2, Z3);
        direction : output;
        function : "D";
        pin(Z0) {
            direction : output;
            timing() {
                intrinsic_rise : 0.4;
                intrinsic_fall : 0.4;
                related_pin : "D0";
            }
        }
        pin(Z1) {
            direction : output;
            timing() {
                intrinsic_rise : 0.4;
                intrinsic_fall : 0.4;
                related_pin : "D1";
            }
        }
    }
    bundle(D) {
        members(D0, D1, D2, D3);
        direction : input;
        capacitance : 1;
        pin(D0) {
            direction : input;
        }
    }
}```
**function Simple Attribute**

The function attribute in a bundle group defines the value of an output pin or inout pin in terms of the input pins or inout pins in the cell group or model group.

**Syntax**

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

Table 2-4 lists the Boolean operators valid in a function statement.

**Table 2-4   Valid Boolean Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>invert previous expression</td>
</tr>
<tr>
<td>!</td>
<td>invert following expression</td>
</tr>
<tr>
<td>^</td>
<td>logical XOR</td>
</tr>
<tr>
<td>*</td>
<td>logical AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td>space</td>
<td>logical AND</td>
</tr>
<tr>
<td>+</td>
<td>logical OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td>0</td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>

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

**Pin Names as Function Arguments**

The following example describes bundle Q with the function A OR B:

```plaintext
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.

```plaintext
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.

```plaintext
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**

```plaintext
bundle (namestring) {
    members (member1, member2 ...);
    ...
}

member1, member2 ...
```

The number of bundle members defines the width of the bundle.

**Example**

```plaintext
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**

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

The previous example shows that the members of the A bundle have these values:

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

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

**Example 2-5** shows how to define a bundle group in a cell with a multibit latch.

**Example 2-5  Multibit Latch With Signal Bundles**

```plaintext
cell (latch4) {
    area:16;
    geometry_print : GP1;
    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;
```
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

```plaintext
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:

```plaintext
bundle (A) {
    pin (A0) {
        capacitance : 4 ;
    }
}
```

To identify the name of a pin in a pin group within a bundle group, use the full name of a pin, such as pin (A0) in the previous example.

All pin names within a single bundle group must be unique. Pin names are case-sensitive; for example, pins named A and a are different pins.

Example 2-4 shows a pin group within a bundle group.

bus Group

A bus group, defined in a cell group or a model group, defines the bused pins in the library. Before you can define a bus group you must first define a type group at the library level.

From the type group you define at the library level, use the type name (bus4 in Example 2-6) as the value for the bus_type attribute in the bus group in the same library.

Example 2-6 a bus group in a cell group.

Example 2-6 Bused Pins

```plaintext
library (ExamBus) {
    type (bus4) { /* bus name */
        bit_width : 4 ; /* number of bused pins */
        ...
        ...
    }
    cell (bused cell) {
        ...
        bus (A) {
            bus_type : bus4 ; /* bus name */
            ...
            ...
        }
    }
}```
Simple Attributes

You can use all of the pin simple attributes in bus group, plus the following attribute.

```
bus_type : name ;
```

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 Example 2-6.

pin Simple Attributes in a bus Group

The pin simple attributes in a bus group define default attribute values for all pins in the bus group.

Note:

* Bus names, bus members, bus pins, bused pins, pins, members, member numbers, and range of bus members are valid terms for pin names in a bus group.

All pin group simple attributes are valid within a bus group and within a pin group in a bus group.

The capacitance and direction attributes are frequently used in bus groups.

capacitance Simple Attribute

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

Syntax

```
capacitance : value_float ;
```

`value`

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

The following example shows a bus group that defines bus A with default values assigned for direction and capacitance.

```
Example

bus {A} {
  bus_type : bus1 ;
  direction : input ;
  capacitance : 3 ;
}
```

direction Simple Attribute
The direction attribute states the direction of bus members (pins) in a bus group.

The value of the direction attribute of all bus members (pins) in a bus group must be the same. (To see a bus group with more than one pin, refer to Example 2-7.)

Syntax

direction : input | output | inout | internal ;

Example

direction : inout ;

pin Group Statement in a bus Group

This group defines attribute values for specific bused pins or groups of bused pins in a pin group within a bus group. Values used in a pin group within a bus group override the defined default bus pin attribute values described previously.

Note:

You can use a defined bus or buses in Boolean expressions in the function attribute of a pin in a bus group, as shown in Example 2-7.

The following example shows a bus pin group that defines a new capacitance attribute value for member AO in bus A.

```
bus (A) {
   pin (AO) {
      capacitance : 4 ;
   }
}
```

To identify the name of a bused pin in a pin group within a bus group, use the full name of the pin. You can identify bus member numbers as single numbers or as a range of numbers separated by a colon. No spaces can appear between the colon and the member numbers.

Example

```
pin (A[0:2]) {}
```

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 ;
   }
}
```

Sample Bus Description–Technology Library

Example 2-7 illustrates a complete bus description that includes a library-defined type.
group and cell-defined bus groups. The example also illustrates the use of bus variables in a function attribute in a pin group and in a related_pin attribute in a timing group.

**Example 2-7  Bus Description**

```liberty
library (ExamBus) {
  date : "November 12, 2000";
  revision : 2.3;
  bus_naming_style : "%s[%d]";
  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;
    geometry_print : GP1;
    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 : "AB";
          /* 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] BE";
        }
      }
    }
    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]";
        }
      }
    }
}
```

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:

```liberty
library (name) {
  cell (name) {
    dynamic_current () {
      when: <boolean expression>
      related_inputs : <input_pin_name>;
      related_outputs : <output_pin_name>;
      typical_capacitances("<float>, ..");
      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

**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 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 "non-propagating 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 "non-propagating event." There is no `related_outputs` attribute for macro cells. Therefore, you do not need to distinguish between non-propagating and propagating event tables.

**Syntax**

```
related_outputs : <output_pin_name>;

name
```

Name of output pin.

**Example**

```
related_outputs : D;
```

**Typical_capacitances Simple Attribute**

The `typical_capacitances` attribute defines fixed capacitance values for all output pins specified in `related_outputs`. The order in the list of this attribute is mapped to the same order of output pins in 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 will be 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**

```plaintext
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.

```plaintext
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**

Use the `input_switching_condition` attribute to specify the sense of the toggling input. 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. Two values are allowed: `rise` represents a rising pin and can be applied for input/output switching conditions and `fall` represents a falling pin and can be applied for input/output switching conditions.

**Syntax**

```plaintext
input_switching_condition (<enum(rise, fall)>);
```

*enum(rise, fall)*

Enumerated type specifying rise or fall condition.

**Example**

```plaintext
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. Currently, two values are considered: `rise` represents a rising pin and can be applied for input/output switching conditions and `fall` represents a falling pin and can be applied for input/output switching conditions.

**Syntax**

```plaintext
output_switching_condition (rise);
```
output_switching_condition (<enum(rise, fall)>);

enum(rise, fall)

Enumerated type specifying 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.
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.

**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_2, 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**

```plaintext
index_output : <output_pin_name>;
```

`output_pin_name`

Name of the pin that the output capacitance is sweeping.

**Example**

```plaintext
index_output : "QN";
```

**reference_time Simple Attribute**

This attribute represents the time at which the input waveform crosses the reference voltage.

**Syntax**

```plaintext
reference_time : <float>;
```

`float`

Specifies the time at which the input waveform crosses the reference voltage.

**Example**

```plaintext
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**

```plaintext
values : (<float>, ...);
```

`float`

Defines a list of floating-point numbers that represent the dynamic current waveform of a specified power and ground pin.

**Example**

```plaintext
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”.

**Syntax**

```plaintext
```

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 Simple Attribute

The clear attribute gives the active value for the clear input.

Syntax

clear : "Boolean expression";

clear_preset_var1 : L | H | N | T | X;

clocked_on : "Boolean expression";

clocked_on_also : "Boolean expression";

Example

clear : "CD";

clear_preset_var1 : H;

clocked_on : "L";

clocked_on_also : "H";

next_state : "L";

Table 2-5 shows the valid variable values for the clear_preset_var1, clear_preset_var2, and clocked_on attributes.

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

<table>
<thead>
<tr>
<th>Variable values</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>No change¹</td>
</tr>
<tr>
<td>T</td>
<td>Toggle the current value from 1 to 0, 0 to 1, or X to X¹</td>
</tr>
<tr>
<td>X</td>
<td>Unknown¹</td>
</tr>
</tbody>
</table>
Use these values to generate VHDL models.

"Single-Stage Flip-Flop" 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**

```vhdl
clear_preset_var2 : L ;
```

"Single-Stage Flip-Flop" 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" for details.

When describing flip-flops that require both a master clock and a slave clock, use the `clocked_on` attribute for the master clock and the `clocked_on_also` attribute for the slave clock.

**Syntax**

```
clocked_on : "Boolean expression" ;
clocked_on_also : "Boolean expression" ;
```

**Boolean expression**

Active edge of a clock signal.

**Example**

```vhdl
clocked_on : "CP" ;
clocked_on_also : "CP'" ;
```

**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.

**Syntax**
next_state : "Boolean expression";

The following example shows an $ff$ group for a single-stage D flip-flop.

```plaintext
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:

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

The next_state and clocked_on attributes completely define the synchronous behavior of the flip-flop.

preset Simple Attribute

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

Syntax

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

Example

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

Single-Stage Flip-Flop

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

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

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

Table 2-6 shows the functions of the attributes in the $ff$ group for a single-stage flip-flop.

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

The 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. 
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 2-8 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 2-8 Single-Stage D Flip-Flop**

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

Example 2-9 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 2-9 Single-Stage JK Flip-Flop**

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

Example 2-10 is an ff group for a D flip-flop with synchronous negative clear.

**Example 2-10 D Flip-Flop With Synchronous Negative Clear**

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

Master-Slave Flip-Flop

The syntax for a master-slave flip-flop is the same as for a single-stage device, except that it includes the clocked_on_also attribute. Table 2-7 shows the functions of the attributes in the ff group for a master-slave flip-flop.

The `internal1` and `internal2` variables represent the output values of the master stage, and `variable1` and `variable2` represent the output values of the slave stage. The `variable1` and `variable2` variables have the same value as `internal1` and `internal2`, respectively, when clear and preset are both active at the same time.

**Table 2-7 Function Table for a Master-Slave Flip-Flop**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>active</td>
</tr>
<tr>
<td>preset</td>
<td>active</td>
</tr>
</tbody>
</table>
Example 2-11 shows an `ff` group for a master-slave D flip-flop with rising-edge sampling, falling-edge data transfer, negative clear and preset, and output values set high when clear and preset are both active.

**Example 2-11 Master-Slave D Flip-Flop**

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

**ff_bank Group**

An `ff_bank` group is defined within a `cell` or `test_cell` group, as shown in the following syntax, and in a `scaled_cell` group at the library level.

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".

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

**Syntax**

```plaintext
library (namestring) {
    cell (namestring) {
        ff_bank (variable1string, variable2string, bitsinteger) {
            ... multibit flip-flop register description ...
        }
    }
}
```

**Simple Attributes**

- `clocked_on`: "Boolean expression";
- `next_state`: "Boolean expression";
- `clear`: "Boolean expression";
- `preset`: "Boolean expression";
- `clear_preset_var1`: L | H | N | T | X;
- `clear_preset_var2`: L | H | N | T | X;
- `clocked_on_also`: "Boolean expression";

Example 2-12 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**

```plaintext
clocked_on : "Boolean expression" ;
clocked_on_also : "Boolean expression" ;
```

**Boolean expression**

Identifies the active edge of the clock signal.

**Examples**

```plaintext
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**

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

**Boolean expression**

Identifies the active edge of the clock signal.

**Example**

```plaintext
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**

```plaintext
clear : "Boolean expression" ;
```

**Example**

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

See "Single-Stage Flip-Flop" 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" 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.

    Table 2-8 shows the valid variable values for the clear_preset_var1 and
    clear_preset_var2 attributes.

Table 2-8  Valid Values for the clear_preset_var1 and clear_preset_var2 Attributes

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

† Use these values to generate VHDL models.

Syntax

    clear_preset_var1 : L | H | N | T | X ;

Table 2-8 shows the valid variable values for the clear_preset_var2 attribute.

Example

    clear_preset_var1 : L ;

See "Single-Stage Flip-Flop" for 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 ;

See "Single-Stage Flip-Flop" 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 (namestring) {
  cell (namestring) {
    ff_bank (variable1string, variable2string, bitsinteger) {
      ... multibit flip-flop register description ...
    }
  }
}
```

A multibit register containing four rising-edge-triggered D flip-flops with *clear* and *preset* is shown in Figure 2-1 and Example 2-12.

**Figure 2-1  Multibit Register**

![Multibit Register Diagram](image)

**Example 2-12  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;
      intrinsic_rise : 1.0;
      intrinsic_fall : 1.0;
    }
    timing() {
      related_pin : "CLK";
      timing_type : hold_rising;
      intrinsic_rise : 1.0;
      intrinsic_fall : 1.0;
    }
  }
  pin (CLR) {
    direction : input;
    capacitance : 0;
  }
```
fpga_condition Group

An fpga_condition group declares an fpga_condition group containing several fpga_condition_value groups.
Syntax

```plaintext
cell (nameID) {
    fpga_condition (nameID) {
        ... 
    }
}
```

(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

```plaintext
cell (nameID) {
    fpga_condition (condition_group_nameID) {
        fpga_condition_value (condition_nameID) {
            ... 
        }
    }
}
```

(condition_name)

Specifies the name of a condition.

**Simple Attribute**

`fpga_arc_condition`

**fpga_arc_condition Simple Attribute**

The `fpga_arc_condition` attribute specifies a Boolean condition that enables the associated `fpga_condition_value` group.

Syntax

```plaintext
cell (name_string) {
    fpga_condition (nameID) {
        fpga_condition_value (condition_nameID) {
            fpga_arc_condition : conditionBoolean ;
        }
    }
}
```

(condition)

Specifies a Boolean condition. Valid values are true and false.

**Example**

```plaintext
fpga_arc_condition : true ;
```

**functional_yield_metric Group**

To model yield information, use the `functional_yield_metric` group with the `faults_lut_template` group. For details on the `faults_lut_template` group, see "faults_lut_template".
functional_yield_metric () {
    average_number_of_faults ( name faults_lut_template ) {
        values ( "float, ..., float" );
    }
}

This group holds values for average_number_of_faults. The group name indicates that its values array is based on the specified faults_lut_template template. The average_number_of_faults group holds the array of fault values.

Example

library ( my_library_name ) {
    ...
    faults_lut_template ( my_faults_temp ) {
        variable_1 : fab_name;
        variable_2 : time_range;
        index_1 (* fab1, fab2, fab3 *");
        index_2 (* 2005.01, 2005.07, 2006.01, 2006.07 *);
    }
    ...
    cell ( and2 ) {
        ...
        functional_yield_metric () {
            average_number_of_faults ( my_faults_temp ) {
                values ( "73.5, 78.8, 85.0, 92.0",
                        "74.3, 78.7, 84.8, 92.2",
                        "72.2, 78.1, 84.3, 91.0" );
            }
        }
    }
    ...
    } /* end of cell */
    } /* end of library */

This example specifies fault data for three fabs (fab1, fab2, and fab3).

For fab1:

- 73.5 is the average number of faults due to functional yield loss mechanisms (that is, random defects) for the time range 2005.01 to 2005.06
- 78.8 is the average number of faults due to functional yield loss mechanisms for the time range 2005.07 to 2005.12
- 85.0 is the average number of faults due to functional yield loss mechanisms for the time range 2006.01 to 2006.07
- 92.0 is the average number of faults due to functional yield loss mechanisms for the time range 2006.07 or later

For fab2:

- 74.3 is the average number of faults due to functional yield loss mechanisms for the time range 2005.01 to 2005.06
- 78.7 is the average number of faults due to functional yield loss mechanisms for the time range 2005.07 to 2005.12
- 84.8 is the average number of faults due to functional yield loss mechanisms for the time range 2006.01 to 2006.07
- 92.2 is the average number of faults due to functional yield loss mechanisms for the time range 2006.07 or later

And so on for fab3.

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...
}
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 three 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.

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 (three). This shift modifies the ideal clock edges; it is not considered to be clock latency.

Syntax

    shifts (shift1, shift2, shift3);

Example
Example 2-13 shows a generated clock description.

**Example 2-13 Description of a Generated Clock**

```plaintext
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(clkl)
    {
        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**

```plaintext
cell (name_string) {
    intrinsic_parasitic (name_string) {
        when : <boolean expression>;
        intrinsic_resistance(<pg_pin_name>) {
            related_output : <output_pin_name>;
            value : <float>;
        }
        intrinsic_capacitance(<pg_pin_name>) {
            value : <float>;
        }
    }
}
```

**Simple Attribute**

**when**

**Groups**

intrinsic_capacitance
intrinsic_resistance
total_capacitance

**when Simple Attribute**
This attribute specifies the state-dependent condition that determines whether the intrinsic parameters are accessed. An `intrinsic_parasitic` group without a `when` attribute is defined as a default state. This state is associated with a non-state-dependent parasitic model. If all the state conditions of a cell are specified, a default state is not required. If some of the state conditions of a cell are missing, the default state will be assigned. If some state conditions of a cell are missing and no default state is given, the value of `intrinsic_resistance` defaults to +infinity and the value of `intrinsic_capacitance` defaults to 0.

**Syntax**

```
when : <boolean expression>;
```

*boolean expression*

Specifies the state-dependent condition.

**Example**

```
when : "A B";
```

**intrinsic_capacitance Group**

Use this group to specify a cell’s intrinsic capacitance.

**Syntax**

```
cell (name string) {
  intrinsic_parasitic (name string) {
    intrinsic_capacitance (<pg_pin_name>) {
      value : <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. Groups can be placed in any order within an `intrinsic_parasitic` group.

**value Simple Attribute**

Specifies the value of the intrinsic capacitance. If some of an `intrinsic_capacitance` group is not defined, the value of the intrinsic capacitance defaults to 0.

**Syntax**

```
value : <float>;
```

**Example**

```
value : 5;
```

**intrinsic_resistance Group**

Use this group to specify a cell’s intrinsic resistance between a power and ground pin and an output pin within the `intrinsic_parasitic` group in a cell.

**Syntax**

```
cell (name string) {
  intrinsic_parasitic (name string) {
    intrinsic_resistance (<pg_pin_name>) {
      related_output : <output_pin_name>;
      value : <float>;
    }
  }
}
```
The `pg_pin_name` specifies a power or ground pin. The `intrinsic_resistance` groups can be placed 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 pin and the output pin is defined as a closed channel if the resistance value is greater than 1 megohm. Otherwise, the channel is opened. The `intrinsic_resistance` group is not required if the channel is closed.

**Simple Attributes**

**related_output**

value

**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_outputs : "A B";

**value Simple Attribute**

Specifies the value of the intrinsic resistance. If some of an `intrinsic_resistance` group is not defined, the value of the intrinsic resistance defaults to +infinity.

**Syntax**

value : <float>;

**Example**

value : 5;

**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 macros 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 () {

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".

```
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 : valueBoolean;
```

**Example**

The following example defines a low-active clear signal.

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

**clear_preset_var1 and clear_preset_var2 Simple Attributes**

The clear_preset_var1 and clear_preset_var2 attributes give the value that variable1 and variable2 have when clear and preset are both active at the same time.

**Syntax**

```
clear_preset_var1 : L | H | N | T | X;
clear_preset_var2 : L | H | N | T | X;
```

Table 2-9 shows the valid values for the clear_preset_var1 and clear_preset_var2 attributes.
Table 2-9  Valid Values for the clear_preset_var1 and clear_preset_var2 Attributes

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

\(^1\) Use these values to generate VHDL models.

See "Single-Stage Flip-Flop" 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

```vhdl
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.

Syntax

```vhdl
data_in : valueBoolean;
```

value

State of data input.

Example

```vhdl
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

```vhdl
enable : valueBoolean;
```

value

State of enable input.

Example

```vhdl
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 `enable` and `data_in` attributes.

**Syntax**

```
enable_also : "valueBoolean" ;
```

**Value**

State of enable input for master-slave cells.

**Example**

```
enable_also : "G" ;
```

preset Simple Attribute

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

**Syntax**

```
preset : "valueBoolean" ;
```

**Example**

The following example defines a low-active clear signal.

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

**Attribute Functions in a latch Group**

The latch cell is activated whenever `clear`, `preset`, `enable`, or `data_in` changes.

Table 2-10 shows the functions of the attributes in the `latch` group.

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

**Example 2-14** shows a `latch` group for a D latch with active-high enable and negative clear.

**Example 2-14  D Latch With Active-High Enable and Negative Clear**

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

**Example 2-15** shows a `latch` group for an SR latch. The `enable` and `data_in` attributes...
are not required for an SR latch.

**Example 2-15  SR Latch**

```plaintext
latch(IQ, IQN) {
    clear : "S";
    preset : "R";
    clear_preset_var1 : L;
    clear_preset_var2 : L;
}
```

**latch_bank Group**

A *latch_bank* group is defined within a *cell*, *model*, or *test_cell* group and in a *scaled_cell* group at the library level to represent multibit registers. The syntax for a cell is shown here. For information about test cells, see "*test_cell Group*".

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*").

**Syntax**

```plaintext
library (namestring) {
    cell (namestring) {
        latch_bank(variable1string, variable2string, bitsinteger){
            ... multibit latch register description ...
        }
    }
}
```

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

**Simple Attributes**

- `enable : "Boolean expression";`
- `enable_also : "Boolean expression";`
- `data_in : "Boolean expression";`
- `clear : "Boolean expression";`
- `preset : "Boolean expression";`
- `clear_preset_var1 : L | H | N | T | X;`
- `clear_preset_var2 : L | H | N | T | X;`

**Example 2-16** shows a *latch_bank* group for a multibit register containing four rising-edge-triggered D latches.

**Example 2-16  Multibit D Latch**

```plaintext
cell (latch4) {
    area: 16;
    pin (G) { /* gate enable signal, active-high */
        direction : input;
        ...
    }
    bundle (D) { /* data input with four member pins */
        members(D1, D2, D3, D4); /* must be 1st bundle attribute*/
        direction : input;
        ...
    }
    bundle (Q) {
        members(Q1, Q2, Q3, Q4);
        direction : output;
    }
}
```
function "IQ";
...
}]
bundle (QN) {
  members (Q1N, Q2N, Q3N, Q4N);
  direction : output;
  function : "IQN";
  ... }
latch_bank (IQ, IQN, 4) {
  enable : "G";
  data_in : "D";
} ...

**clear Simple Attribute**

The `clear` attribute gives the active value for the clear input.

**Syntax**

```
clear : "Boolean expression";
```

The following example defines a low-active clear signal.

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

**clear_preset_var1 and clear_preset_var2 Simple Attributes**

The `clear_preset_var1` and `clear_preset_var2` attributes give the values that `variable1` and `variable2` have when `clear` and `preset` are both active at the same time.

**Syntax**

```
clear_preset_var1 : L | H | N | T | X;
clear_preset_var2 : L | H | N | T | X;
```

**Example**

See Table 2-9 for the valid values for the `clear_preset_var1` and `clear_preset_var2` attributes.

See “Single-Stage Flip-Flop” 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**

```plaintext
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**

```plaintext
enable : "Boolean expression";
```

**Example**

```plaintext
enable : "G";
```

**preset Simple Attribute**

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

**Syntax**

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

The following example defines a low-active clear signal.

```plaintext
preset : "PD";
```

**Attribute Functions in a latch_bank Group**

The `latch_bank` cell is activated whenever the value of `clear`, `preset`, `enable`, or `data_in` attribute changes.

**Figure 2-2** and **Example 2-17** show a multibit register containing four high-enable D latches with the `clear` attribute.
Example 2-17  Multibit Register With Four D Latches

cell (DLT2) {
  /* note: 0 hold time */
  area : 1;
  geometry_print : GP1;
  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;
      intrinsic_rise : 1.0;
      intrinsic_fall : 1.0;
    }
    timing() {
      related_pin : "EN";
      timing_type : hold_falling;
      intrinsic_rise : 0.0;
      intrinsic_fall : 0.0;
    }
  }
  bundle (CLR) {
    members (CLRA, CLRB, CLRC, CLRD);
    direction : input;
    capacitance : 0;
    timing() {
      related_pin : "EN";
      timing_type : recovery_falling;
      intrinsic_rise : 1.0;
      intrinsic_fall : 0.0;
    }
  }
  bundle (PRE) {
    members (PREA, PREB, PREC, PRED);
    direction : input;
    capacitance : 0;
    timing() {
      related_pin : "EN";
      timing_type : recovery_falling;
      intrinsic_rise : 1.0;
      intrinsic_fall : 0.0;
    }
  }
}
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 Group
leakage_current() {
    when : boolean expression;
    pg_current(<pg_pin_name>) {
        value : float;
    }
    ...
}

Simple Attribute

when
value

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 will be 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 only a single power and ground pin, you can omit the pg_current group and specify the leakage current value at this level. Otherwise, specify the value in the pc_current group as shown below. 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.

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 : valuefloat;

value

A floating-point number representing the leakage current.

**gate_leakage Group**

The `gate_leakage` group specifies the cell’s gate leakage current on input or inout pins within the `leakage_current` group in a cell. The following applies to `gate_leakage` groups:

- Groups can be placed in any order if there is more than one `gate_leakage` group within a `leakage_current` group. The leakage current of a cell is characterized with opened outputs, which means that modeling cell outputs do not drive any other cells. Outputs are assumed to have zero static current during the measurement. A missing `gate_leakage` group is allowed for certain pins. Current conservation is applicable if it can be applied to higher error tolerance.

**Syntax**

gate_leakage (<an input pin name>)

**Example**

```plaintext
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**

```plaintext
...`
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.

- A positive floating-point number value is required.
- The gate leakage current flow is measured from the power pin of its driver cell to the ground pin of the cell itself.
- The input pin is pulled up to high.
- The `input_high_value` attribute is not required for a `gate_leakage` group.

**Syntax**

```plaintext
input_high_value : <float> ;
```

**Example**

```plaintext
... 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.

**Syntax**

```plaintext
library (name) {
    cell (name) {
        leakage_power () {
            ...
        }
    }
}
```

**Simple Attributes**

- `power_level`
- `related_pg_pin`
- `when`
- `value`

`power_level` Simple Attribute

Use this attribute to specify the power consumed by the cell.

**Syntax**

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

```
name
```

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

```
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 will always be zero.

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;
```

`pg_pin`

The related power and ground pin name.

**Example**

```
Example

    related_pg_pin : G2;
```

**when Simple Attribute**

This attribute specifies the state-dependent condition that determines whether the leakage power is accessed.

**Syntax**

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

```
Boolean expression
```

Name of pin or pins in a cell for which `leakage_power` is different.

**Table 2-11** lists the Boolean operators valid in a `when` statement.

**Table 2-11  Valid Boolean Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>invert previous expression</td>
</tr>
<tr>
<td>!</td>
<td>invert following expression</td>
</tr>
<tr>
<td>^</td>
<td>logical XOR</td>
</tr>
<tr>
<td>*</td>
<td>logical AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td>space</td>
<td>logical AND</td>
</tr>
<tr>
<td>+</td>
<td>logical OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td>0</td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>
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 ;
}
```

**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 declares a `mode` group that contains several timing mode values. Each timing arc can be enabled based on the current mode of the design, which results in different timing for different modes. You can optionally put a condition on a `mode` value. When the condition is true, the `mode` group takes that value.

Syntax

```
cell (name_string) {
    mode_definition (name_string) (}
```
Group Statement

```plaintext
mode_value (namestring) {}
```

**mode_value Group**

Use this group to specify a timing mode.

**Simple Attributes**

```plaintext
when sdf_cond
```

**when Simple Attribute**

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

**Syntax**

```plaintext
when : Boolean expression;
```

**Example**

```plaintext
when: !R;
```

**sdf_cond Simple Attribute**

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

**Syntax**

```plaintext
sdf_cond : sdf_expressionstring;
```

**Example**

```plaintext
sdf_cond: "R == 0";
```

Example 2-18 shows a **mode_definition description**.

**Example 2-18  mode_definition Description**

```plaintext
cell(sample_cell) {
  ...
  mode_definition(rw) {
    mode_value(read) {
      when : "R";
      sdf_cond : "R == 1";
    }
    mode_value(write) {
      when : "!R";
      sdf_cond : "R == 0";
    }
  }
}
```

**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 (namestring)
  pg_pin (pg_pin_namestring)
    voltage_name : valueid;
    pg_type : valueenum;
  */ end pg_pin
*/ end cell */
```

**Simple Attributes**

- `voltage_name`
- `pg_type`

**voltage_name Simple Attribute**

Use the `voltage_name` attribute to specify an associated voltage.

```
voltage_name : valueid;
```

**Value**

A voltage defined in a library-level `voltage_map` attribute.

**Example**

```
voltage_name : VDD1;
```

**pg_type Simple Attribute**

Use the optional `pg_type` attribute to specify the type of power and ground pin.

```
pg_type : valueenum;
```

**Value**

The valid values are `primary_power`, `primary_ground`, `backup_power`, `backup_ground`, `internal_power`, and `internal_ground`.

**Example**

```
pg_type : primary_power;
```

**routing_track Group**

A `routing_track` group is defined at the `cell group` or `model group` level:

```
library (namestring)
  cell (namestring)
    routing_track (routing_layer_namestring)
    ... routing track description ...
  }
} routing_track_group
```

**Simple Attributes**
tracks : integer;
  total_track_area : float;

Complex Attribute

  short /* for model group only */

tracks Simple Attribute

  The tracks attribute indicates the number of tracks available for routing on any particular layer.

Syntax

  tracks : value_int;

  value

    A number larger than or equal to 0.

Example

  tracks : 2;

total_track_area Simple Attribute

  The total_track_area attribute specifies the total routing area of the routing tracks.

Syntax

  total_track_area : value_float;

  value

    A floating-point number larger than or equal to 0.0 and less than or equal to the area on the cell.

Example

  total_track_area : 0.2;

statetable Group

  The statetable group captures the function of more-complex sequential cells. It is defined in a cell group, model group, scaled_cell group, or test_cell group.

  The purpose of this group is to define a state table. A state table is a sequential lookup table. It takes an arbitrary number of inputs and their delayed values and an arbitrary number of internal nodes and their delayed values to form an index to new internal node values.

  Note:

    In the statetable group, table is a keyword.

Syntax

  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 (namestring) {
  cell (namestring) {
    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_list) {}
statetable ("input node names", "internal node names") {} 
```

ff Group

For a discussion of the ff group syntax, see “ff Group”.

ff_bank Group

For a discussion of the ff_bank group syntax, see “ff_bank Group”.

latch Group

For a discussion of the latch group syntax, see “latch Group”.

latch_bank Group

For a discussion of the latch_bank group syntax, see “latch_bank Group”.

pin Group in a test_cell Group

Both test pin and nontest pin groups appear in pin groups within a test_cell group:

```
library (namestring) {
  cell (namestring) {
    test_cell (namestring) {
      pin (namestring | name_liststring) {
        ... pin description ...
      }
    }
  }
}
```
These groups are similar to pin groups in a cell group or model group but can contain only direction, function, signal_type, and test_output_only attributes. They cannot contain timing, capacitance, fanout, or load information.

**Simple Attributes**

- **direction**: input | output | inout;
- **function**: Boolean expression;
- **signal_type**: test_scan_in | test_scan_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;

```plaintext
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**".

**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 will scan
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**

```plaintext
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.

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

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

**Syntax**

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

**Example**

```plaintext
test_output_only : true ;
```

**statetable Group**

For a discussion of the `statetable` group syntax, see "statetable Group ".

**type Group**

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

**Syntax**

```plaintext
cell (name_string) {
    type (name_string) {
        ... type description ...
    }
}
```

**Simple Attributes**

```plaintext
base_type : array;
bit_from : integer;
bit_to : integer;
bit_width : integer;
data_type : bit;
downto : Boolean ;
```

**base_type Simple Attribute**

The only valid base type value is `array` .

**Example**

```plaintext
base_type : array;
```
bit_from Simple Attribute

The `bit_from` attribute specifies the member number assigned to the most significant bit (MSB) of successive array members.

Syntax

```plaintext
bit_from : value_int;
```

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

Example

```plaintext
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

```plaintext
bit_to : value_int;
```

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

Example

```plaintext
bit_to : 3;
```

bit_width Simple Attribute

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

Syntax

```plaintext
bit_width : value_int;
```

- **value**
  - Designates the number of bus members. The default is 1.

Example

```plaintext
bit_width : 4;
```

data_type Simple Attribute

Only the bit data type is supported.

Example

```plaintext
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 2-19** illustrates a type group statement in a cell.

**Example 2-19  type Group Within a Cell**

```plaintext
cell (buscell4) {
  type (BUS4) {
    base_type : array ;
    data_type : bit ;
    bit_width : 4 ;
    bit_from : 0 ;
    bit_to : 3 ;
    downto : true ;
  }
}
```

### 2.2  model Group

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

**Syntax**

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

#### 2.2.1  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".

**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**

```plaintext
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 shorted port is a feedthrough, where an input port is directly connected to an output port. Another example is two output ports that fan out from the same gate.

Syntax

short ("name_liststring") ;

Example

short(b, y);

Example 2-20 shows how to use a short attribute in a model group.

Example 2-20 Using the short Attribute in a model Group

```plaintext
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, scaled_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

3.1 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:

```plaintext
library (name) {
    cell (name) {
        pin (name | name_list) {
            ... pin description ...
        }
    }
}
```
3.1.1 Pin Group Example

Example 3-1 shows pin groups with CMOS library attributes and a timing group.

Example 3-1 CMOS pin Group Example

```plaintext
library(example){
  date: "November 12, 2002";
  revision: 2.3;
  ...
  cell(AN2) {
    area: 2;
    pin(A) {
      direction: input;
      dont_fault: true;
      capacitance: 1.3;
      fanout_load: 2; /* internal fanout load */
      max_transition: 4.2; /* design-rule constraint */
    }
    pin(B) {
      direction: input;
      capacitance: 1.3;
    }
    pin(Z) {
      direction: output;
      function: "A * B";
      max_transition: 5.0;
      timing() {
        intrinsic_rise: 0.58;
        intrinsic_fall: 0.69;
        rise_resistance: 0.1378;
        fall_resistance: 0.0465;
        related_pin: "AB";
      }
    }
  }
}
```

3.1.2 Simple Attributes

Example 3-2 lists alphabetically a sampling of the attributes and groups that you can define within a pin group.

Example 3-2 Attributes and Values in a Pin Group

```plaintext
/* Simple Attributes in a pin Group */

  bit_width: integer; /* bus cells */
  capacitance: float;
  clock: true | false;
  clock_gate_clock_pin: true | false;
  clock_gate_enable_pin: true | false;
  clock_gate_test_pin: true | false;
  clock_gate_obs_pin: true | false;
  clock_gate_out_pin: true | false;
  complementary_pin: "string";
  connection_class: "name1 [name2 name3 ...]";
  direction: input | output | inout | internal;
  dontFault: sa0 | sa1 | sa01;
  drive_current: float;
  driver_type: pull_up | pull_down | open_drain | open_source | bus_hold | resistive
            | resistive_0 | resistive_1;
  fall_capacitance: float;
  fall_current_slope_after_threshold: float;
  fall_current_slope_before_threshold: float;
  fall_time_after_threshold: float;
```
fall_time_before_threshold: float;
fanout_load: float;
fault_model: "two-value string";
function: "Boolean expression";
has_builtin_pad: Boolean expression;
hysteresis: true | false;
input_map: "name string | name_list";
input_signal_level: string;
inverted_output: true | false; /* Required in statetable cells */
is_pad: true | false;
max_capacitance: float;
max_fanout: float;
max_input_noise_width: float;
min_capacitance: float;
min_fanout: float;
min_input_noise_width: 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;
prefer_tied: "0" | "1";
primary_output: true | false;
pulling_current: current value;
pulling_resistance: resistance value;
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;
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";
vhd1_name: "string";
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 () {};
hyperbolic_noise_above_high () {};
hyperbolic_noise_below_low () {};
hyperbolic_noise_high () {};
hyperbolic_noise_low () {};
internal_power () {};
max_trans () {};
in_min_pulse_width () {};
minimum_period () {};
timing () {};
slatch () {};

bit_width Simple Attribute
An integer that 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;
        vhdl_name : "AND2";
        pin (A, B) {
            direction : input;
            capacitance : 1;
        }
    }

clock Simple Attribute

The clock attribute indicates whether an input pin is a clock pin.

Syntax

    clock : true | false;

A true value labels a pin as a clock pin. A false value labels a pin as not a clock pin, even though it may otherwise have such 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;

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".

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".

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

collection_class : "internal";

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.

```vhdl
cell (AND) {
area : 3;
vhdl_name : "AND2";
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 : valuefloat ;

    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.

**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 0 (L).

**pull_down**

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

**open_drain**

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

**open_source**

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

**bus_hold**

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

**resistive**

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

**resistive_0**

The pin is an output pin connected to power through a pull-up transistor that has a control port EN. When EN is disabled, the pull-up transistor is turned off and has no effect on the pin. When EN is enabled, a functional value of 1 evaluated at the pin turns into a weak 1, but a functional value of 0 or Z is not affected.

**resistive_1**

The pin is an output pin connected to ground through a pull-down transistor that has a control port EN. When EN is disabled, the pull-down transistor is turned off and has no effect on the pin. When EN is enabled, a functional value of 0 evaluated at the pin turns into a weak 0, but a functional value of 1 or Z is not affected.

Table 3-1 lists the driver types, their signal mappings, and the applicable pin types.

**Table 3-1 Pin Driver Types**

<table>
<thead>
<tr>
<th>Driver type</th>
<th>Signal mapping</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>pull_up</td>
<td>01Z-&gt;01H</td>
<td>in, out</td>
</tr>
<tr>
<td>pull_down</td>
<td>01Z-&gt;01L</td>
<td>in, out</td>
</tr>
<tr>
<td>open_drain</td>
<td>01Z-&gt;0ZZ</td>
<td>out</td>
</tr>
<tr>
<td>open_source</td>
<td>01Z-&gt;0Z1Z</td>
<td>out</td>
</tr>
<tr>
<td>bus_hold</td>
<td>01Z-&gt;01S</td>
<td>inout</td>
</tr>
<tr>
<td>resistive</td>
<td>01Z-&gt;LHZ</td>
<td>out</td>
</tr>
<tr>
<td>resistive_0</td>
<td>01Z-&gt;0HZ</td>
<td>out</td>
</tr>
<tr>
<td>resistive_1</td>
<td>01Z-&gt;L1Z</td>
<td>out</td>
</tr>
</tbody>
</table>
Following are important concepts to keep in mind when interpreting Table 3-1.

- The signal modifications a **driver_type** attribute defines divide into two categories, transformation and resolution.
  - Transformation specifies an actual signal transition from 0/1 to L/H/Z. This signal transition performs a function on an input signal and requires only a straightforward mapping.
  - Resolution resolves the value Z on an existing circuit node without actually performing a function and implies a constant (0/1) signal source as part of the resolution.

In Table 3-1, the pull_up, pull_down, and bus_hold driver types define a resolution scheme. The remaining driver types define transformations.

**Example 3-3** describes an output pin with a pull-up resistor and the bidirectional pin on a bus_hold cell.

**Example 3-3 Pin Driver Type Specifications**

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

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.

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**".

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**".

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.

```vhdl
cell (AND) {
  area : 3;
  vhdl_name : "AND2";
  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

```vhdl
fall_current_slope_after_threshold : valuefloat;
```

value

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

Example

```vhdl
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

```vhdl
fall_current_slope_before_threshold : valuefloat;
```

value

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

Example

```vhdl
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\text{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 \texttt{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\text{float} :

value

A floating-point number that represents the time interval.

Example

fall_time_before_threshold : 0.55 ;

fanout_load Simple Attribute

The \texttt{fanout_load} attribute gives the internal fanout load for an input pin.

Syntax

fanout_load : value\text{float} :

value

A floating-point number that represents the internal fanout of the input pin. There are no fixed units for \texttt{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 \texttt{pin} group designated with the \texttt{complementary_pin} attribute.

The \texttt{fault_model} attribute defines a two-value string when both differential inputs are driven to the same value. The first value represents the value when both input pins are at logic 0, and the second value represents the value when both input pins are at logic 1.

For details on the \texttt{complementary_pin} attribute, see "\texttt{complementary_pin Simple Attribute}".

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**

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

**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**

```plaintext
function : "Boolean expression"
```

**Table 3-2 Valid Boolean Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`'</td>
<td>invert previous expression</td>
</tr>
<tr>
<td><code>!</code></td>
<td>invert following expression</td>
</tr>
<tr>
<td><code>^</code></td>
<td>logical XOR</td>
</tr>
<tr>
<td><code>*</code></td>
<td>logical AND</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>logical AND</td>
</tr>
<tr>
<td><code>space</code></td>
<td>logical AND</td>
</tr>
<tr>
<td><code>+</code></td>
<td>logical OR</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>1</code></td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td><code>0</code></td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>

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

**Example**

```plaintext
pin(Q) {
  direction : output;
  function : "A + B";
}
```

**Note:**

Pin names beginning with a number, and pin names containing special characters, must be enclosed in double quotation marks preceded by a backslash (\), as shown here:
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. 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')";

**Grouped Pins in a function Statement**

Grouped pins can be used as variables in the function attribute statement.

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.

Ranges of buses or bundles are valid as long as 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


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

function: "(A & B)*";

the function for Z[0] is interpreted as

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

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


**has_builtin_pad Simple Attribute**

Use this attribute in the case of an FPGA containing an ASIC core connected to the chip’s port. When set to true, this attribute specifies that an output pin has a built-in pad, which prevents pads from being inserted on the net connecting the pin to the chip’s port.
Syntax

```
has_builtin_pad : Boolean ;
```

**Example**

```
has_builtin_pad : true ;
```

**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 ;
```

**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 : nameid ;
```

```
  name
  A string representing a name or a list of port names, separated by spaces, that correspond to the input pin names, followed by the internal node names.
```

**Example**

```
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. If the `input_signal_level` or `output_signal_level` attribute is missing, you can apply the default power supply name to the cell.

**Syntax**

```
input_signal_level : nameid ;
output_signal_level : nameid ;
```

```
  name
  A string representing the name of the power supply already defined at the library level. 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.
```

**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 value for all pins.

**Syntax**

```
input_threshold_pct_fall : trip_pointfloat ;
```

**trip_point**

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

**Syntax**

```
input_threshold_pct_rise : trip_pointfloat ;
```

**trip_point**

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

**Example**

```
input_threshold_pct_rise : 40.0 ;
```

input_voltage Simple Attribute

You can define a special set of voltage thresholds in the library group with the `input_voltage` or `output_voltage` attribute. You can then apply the default voltage ranges in the group to selected cells with the `input_voltage` or `output_voltage` attribute in the pin definition.

**Syntax**

```
input_voltage : nameid ;
output_voltage : nameid ;
```

**name**

A string representing the name of the voltage range group defined at the library level. The `input_voltage` attribute is used for an input pin definition, and the `output_voltage` attribute is used for an output pin definition.

**Example**

```
input_voltage : CMOS_SCHMITT ;
output_voltage : GENERAL ;
```

internal_node Simple Attribute

The `internal_node` attribute describes the sequential behavior of an internal pin or an output pin. It indicates the relationship between an internal node in the statetable group and a pin of a cell. Each output or internal pin with the `internal_node` attribute can also
have the optional `input_map` attribute.

**Syntax**

```plaintext
internal_node : pin_name;id;
```

`pin_name`

Name of either an internal or output pin.

**Example**

```plaintext
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**

```plaintext
inverted_output : Boolean expression;
```

**Example**

```plaintext
inverted_output : true
```

`is_pad` Simple Attribute

The `is_pad` attribute indicates which pin represents the pad.

**Syntax**

```plaintext
is_pad : Boolean expression;
```

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

**Example**

```plaintext
cell(INBUF) {
  ...
  pad_cell : true;
  ...
  pin(PAD) {
    direction : input;
    is_pad : true;
    ...
  }
}
```

`isolation_cell_enable_pin` Simple Attribute

The `isolation_cell_enable_pin` attribute specifies the enable input pin on an isolation cell. For more information about isolation cells, see "is_isolation_cell Simple Attribute".

**Syntax**

```plaintext
isolation_cell_enable_pin : Boolean expression;
```

**Boolean expression**

Valid values are true and false.

**Example**

```plaintext
isolation_cell_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 ".

Syntax

\[
\text{level_shifter_enable_pin} : \text{Boolean expression} ;
\]

\text{Boolean expression}

Valid values are true and false.

Example

\[
\text{level_shifter_enable_pin} : \text{true} ;
\]

map_to_logic Simple Attribute

The map_to_logic attribute specifies which logic level to tie a pin to when a power gating cell functions as a normal cell. For more information about power gating cells, see "power_gating_cell Simple Attribute ".

Syntax

\[
\text{map_to_logic} : \text{Boolean expression} ;
\]

\text{Boolean expression}

Valid values are 1 and 0.

Example

\[
\text{map_to_logic} : 1 ;
\]

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

\[
\text{max_capacitance} : \text{value}_{\text{float}} ;
\]

\text{value}

A floating-point number that represents the capacitive load.

Example

\[
\text{max_capacitance} : 1 ;
\]

max_fanout Simple Attribute

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

Syntax

\[
\text{max_fanout} : \text{value}_{\text{float}} ;
\]

\text{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.

```plaintext
pin (X) {
   direction : output;
   max_fanout : 11.0;
}
```

max_input_noise_width Simple Attribute

The `max_input_noise_width` attribute allows you to specify a maximum value for the input noise width on an input pin or an output pin.

**Note:**

When you specify a `max_input_noise_width` value, you must also specify a `min_input_noise_width` value that is less than or equal to the `max_input_noise_width` value.

**Syntax**

```plaintext
max_input_noise_width : value float;
```

**value**

A floating-point number that represents the maximum input noise width.

Example

```plaintext
max_input_noise_width : 0.0;
```

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**

```plaintext
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:

```plaintext
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**

```plaintext
min_capacitance : value float;
```

**value**

A floating-point number that represents the capacitive load.

Example

```plaintext
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_input_noise_width Simple Attribute

The min_input_noise_width attribute allows you to specify a minimum value for the input noise width on an input pin or an output pin.

Note:

When you specify a min_input_noise_width value, you must also specify a max_input_noise_width value that is equal to or greater than the min_input_noise_width value.

Syntax

min_input_noise_width : value_float;

value

A floating-point number that represents the minimum input noise width.

Example

min_input_noise_width : 0.0;

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**

```vhdl
min_pulse_width_high : valuefloat;
```

`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.

```vhdl
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_high` 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**

```vhdl
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**

```vhdl
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 sync set (or sync reset) pin and the D pin of sequential cells are not swapped when the design is instantiated.

**Syntax**

```vhdl
nextstate_type : data | preset |
  clear | load | scan_in | scan_enable ;
```

where

```vhdl
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. Format Example & Page not handled yet shows a `nextstate_type` attribute in a bundle group.

output_signal_level Simple Attribute
See "input_signal_level Simple Attribute".

output_voltage Simple Attribute
See "input_voltage Simple Attribute".

pg_function Simple Attribute
The `pg_function` attribute is used for the coarse-grain switch cells’ virtual VDD output pins to represent the propagated power level through the switch as a function of input `pg_pins`. This is normally a logical buffer and is useful in cases where the VDD and VSS connectivity may be erroneously reversed.

Syntax
```
pg_function : "<function_string>" ;
```

Example
```
pg_function : "VDD" ;
```

pin_func_type Simple Attribute

The `pin_func_type` attribute describes the functionality of a pin.

Syntax
```
pin_func_type : clock_enable | active_high | active_low |
active_rising | active_falling ;
```

`clock_enable`
Enables the clocking mechanism.

`active_high` and `active_low`
Describes the clock active edge or the level of the enable pin of the latches.

`active_rising` and `active_falling`
Describes the clock active edge or level of the clock pin of the flip-flops.

Example
```
pin_func_type : clock_enable ;
```
power_down_function Simple Attribute

The `power_down_function` string attribute is used to identify the condition when an output pin is switched off by `pg_pin` and to specify the Boolean condition under which the cell’s output pin is switched off (when the cell is in "off" mode due to the external power pin states).

If the `power_down_function` is "1" then X is assumed on the pin.

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";
```

You can have as many `prefer_tied` attributes as possible while it is still able to implement D functionality. However, not all will be honored.

For example, if the library developer specifies

```
    prefer_tied : "0";
```

on all the inputs, as many as possible are honored and the rest are ignored. If they are ignored, a message indicating this fact is issued during execution of the `read_lib` command.

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 \texttt{pulling_current} instead of \texttt{pulling_resistance}.

**Example**

\begin{verbatim}
pin(Y) {
  direction : output;
  driver_type : pull_up;
  pulling_resistance : 1000;
  ...
}
\end{verbatim}

\texttt{pulling_resistance} Simple Attribute

The \texttt{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 \texttt{driver_type} attribute set to \texttt{pull_up} or \texttt{pull_down}.

**Syntax**

\begin{verbatim}
pulling_resistance : resistance value;
\end{verbatim}

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

**Example**

\begin{verbatim}
pin(Y) {
  direction : output;
  driver_type : pull_up;
  pulling_resistance : 1000;
  ...
}
\end{verbatim}

\texttt{pulse_clock} Simple Attribute

Use the \texttt{pulse_clock} attribute to model edge-derived clocks at the pin level.

**Syntax**

\begin{verbatim}
pulse_clock : pulse_typeenum:
  pulse_type
  The valid values are \texttt{rise_triggered_high_pulse}, \texttt{rise_triggered_low_pulse}, \texttt{fall_triggered_high_pulse}, and \texttt{fall_triggered_low_pulse}.
\end{verbatim}

**Example**

\begin{verbatim}
pin(Y) {
  ...
  pulse_clock : rise_triggered_low_pulse;
  ...
}
\end{verbatim}

\texttt{related_ground_pin} Simple Attribute

The optional \texttt{related_power_pin} and \texttt{related_ground_pin} attributes, defined at the pin level for output pins and inout pins, replace the \texttt{output_signal_level} attribute. These attributes can also be defined at the pin level for input/inout pins to replace the \texttt{input_signal_level} within a cell, except when you have input overdrive models. In this case, you need to use the \texttt{input_signal_level} in the pin overdrive voltage, which cannot be modeled with \texttt{related_power_pin}. In power and ground pin syntax, the \texttt{pg_pin} groups are mandatory for each cell, and a cell must have at least one \texttt{primary_power pg_pin} and at least one \texttt{primary_ground pg_pin}. Therefore, the default \texttt{related_power_pin} and \texttt{related_ground_pin} of a cell will always exist.
Syntax

related_ground_pin : pg_pin_nameid;

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".

Syntax

related_power_pin : pg_pin_nameid;

pg_pin_name

Name of the related power pin.

Example

pin(Y) {
  ...
  related_power_pin : P1;
  ...
}

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;
  vhdl_name : "AND2";
  pin (A,B) {
    direction : input;
    fall_capacitance : 1;
    rise_capacitance : 2;
    capacitance : 2;
  }
}
rise_current_slope_after_threshold Simple Attribute

The `rise_current_slope_after_threshold` attribute represents a linear approximation of the change in current over time from the point at which the rising transition reaches the threshold to the end of the transition.

**Syntax**

```plaintext
rise_current_slope_after_threshold : value float;
```

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

**Example**

```plaintext
rise_current_slope_after_threshold : -0.09;
```

rise_current_slope_before_threshold Simple Attribute

The `rise_current_slope_before_threshold` attribute represents a linear approximation of the change in current over time, from the beginning of the rising transition to the threshold point.

**Syntax**

```plaintext
rise_current_slope_before_threshold : value float;
```

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

**Example**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
rise_time_before_threshold : value float;
```

- **value**
  
  A floating-point number that represents the time interval for the rise transition from threshold to finish (after).
A floating-point number that represents the time interval for the rise transition from start to threshold (before).

Example

\[
\text{rise\_time\_before\_threshold} : 0.8;
\]

signal_type Simple Attribute

In a test_cell group, signal_type identifies the type of test pin.

Syntax

\[
\text{signal\_type : test\_scan\_in | test\_scan\_in\_inverted | test\_scan\_out | test\_scan\_out\_inverted | test\_scan\_enable | test\_scan\_enable\_inverted | test\_scan\_clock | test\_scan\_clock\_a | test\_scan\_clock\_b | test\_clock ;}
\]

**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 will scan 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.

**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 falling from 1 to 0. You can specify this attribute at the library-level to set a default value for all 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 rising from 0 to 1. You can specify this attribute at the library-level to set a default value for all 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 value for all 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 lower threshold point used to model the delay of a pin rising from 0 to 1. 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 in modeling the delay of a pin rising from 0 to 1. You can specify this attribute at the library-level to set a default value for all 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 rising from 0 to 1. The default value is 80.0.

Example

slew_upper_threshold_pct_rise : 80.0;
state_function Simple Attribute

The state_function attribute defines output logic. The attribute describes a function of input and inout ports that can be made three-state, or ports with an internal_node attribute. 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

\[
\text{state_function} : \text{"Boolean expression" ;}
\]

Example

\[
\text{state_function} : \text{"ON" ;}
\]

For a list of Boolean operators, see Table 3-2.

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

\[
\text{std_cell_main_rail} : \text{true | false ;}
\]

Example

\[
\text{std_cell_main_rail} : \text{true ;}
\]

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 a Z to) the controlled design partitions, including the traditional signal input pins only (with no related power pins to this output).

Syntax

\[
\text{switch_function} : \text{function_string ;}
\]

Example

\[
\text{switch_function} : \text{"CTL" ;}
\]

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

\[
\text{switch_pin} : \text{valueBoolean ;}
\]

Example
switch_pin : true;

test_output_only Simple Attribute

This attribute can be set for any output port described in statetable format.

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

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

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".

The three_state attribute can be used in both combinational and sequential pin groups, with bus or bundle variables.

Example

three_state : "!E" ;

For a list of Boolean operators, see Table 3-2.

vhdl_name Simple Attribute

The vhdl_name attribute defines valid VHDL object names. In cell and pin groups, use vhdl_name to resolve conflicts of invalid object names when porting from .db to VHDL. Some .db object names might violate the more restrictive VHDL rules for identifiers.

Syntax

vhdl_name : "namestring" ;

name

A string that represents a valid VHDL object name.

Example

vhdl_name : "INb";

Example 3-4 shows a vhdl_name attribute in a cell group.
Example 3-4  Use of the vhdl_name Attribute in a Cell Description

cell (INV) {
    area : 1;
    pin(IN) {
        vhdl_name : "INb";
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "IN";
        timing () {
            intrinsic_rise : 0.23;
            intrinsic_fall : 0.28;
            rise_resistance : 0.13;
            fall_resistance : 0.07;
            related_pin : "IN";
        }
    }
}

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";

3.1.3  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_1 float, value_2 float) ;

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".

The power_gating_pin attribute specifies a pair of pin values for a power gating cell. The first value represents the power gating pin class. The second value specifies which logic level (default) the power gating cell is tied to when the power gating cell is functioning in normal mode. For more information about specifying power gating cells, see "power_gating_cell Simple Attribute".
power_gating_pin ("value_1enum", value_2Boolean) :

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
gating 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 (power_pin_1, 0);

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);

3.2 Group Statements

You can use the following group statements in a pin group:

ccsn_first_stage () {}
ccsn_last_stage () {}
dc_current () {}
electromigration () {}
3.2.1 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 CCB parameters: is_needed, is_inverting, stage_type, miller_cap_rise, and miller_cap_fall
- 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>; 
      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 () {
... 
  }
... 
}
Simple Attributes

is_inverting
is_needed
miller_cap_fall
miller_cap_rise
stage_type
when

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. Note that this attribute is different from the "invertness" (or timing_sense) of a timing arc, which may consist of multiple channel-connecting blocks.

Syntax

```plaintext
is_inverting : valueBoolean;
```

value

Valid values are true and false. Set the value to true when the channel-connecting block is inverting.

Example

```plaintext
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

```plaintext
is_needed : valueBoolean;
```

value
Valid values are `true` and `false`. The default is `true`. Set the value to `false` for cells such as diodes, antennas, and clod cells that do not need current-based data.

**Example**

```plaintext
is_needed : true;
```

**miller_cap_fall** Simple Attribute

Use the `miller_cap_fall` attribute to specify the Miller capacitance value for the channel-connecting block.

**Syntax**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
miller_cap_rise : 0.00055;
```

**stage_type** Simple Attribute

Use the `stage_type` attribute to specify the stage type of the channel-connecting block output voltage.

**Syntax**

```plaintext
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**

```plaintext
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.

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_templateid ) {} { }

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

dc_current_template

The name of the dc current lookup table.

Use index_1 to represent the input voltage and index_2 to represent the output voltage. The values attribute of the group lists the relative channel-connecting block dc current values in library units measured at the channel-connecting block output node.

output_voltage_fall Group

Use the output_voltage_fall group to specify vector groups that describe three-dimensional output_voltage tables of the channel-connecting block whose output node’s voltage values are falling.

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

The index_1 attribute of the vector group lists the input_net_transition (slew) values, in library "time" units, of the channel-connecting block. The index_2 attribute of the vector group lists the total_output_net_capacitance (load) values, in library "capacitance" units, of the channel-connecting block. The index_3 attribute of the vector group lists the sampling time values, in library "time" units, of the channel-connecting block. The values attribute of the vector group list the voltage values, in library "voltage" units, 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 raising.

For details, see the output_voltage_fall group description.

propogated_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

    The index_1 attribute of the vector group lists the input_noise_height values, in library “voltage” units, of the channel-connecting block. The index_2 attribute of the vector group lists the input_noise_width values, in library “time” units, of the channel-connecting block. The index_3 attribute of the vector group lists the total_output_net_capacitance values, in library “capacitance” units, of the channel-connecting block. The index_4 attribute of the vector group lists the sampling time values, in library “time” units, of the channel-connecting block. The values attribute of the vector group lists the voltage values, in library “voltage” units, measured at the channel-connecting block output node.

propogated_noise_low Group

Use the propogated_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 "propogated_noise_high Group".

3.2.2 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".

3.2.3 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) {} 

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 Synopsys logic expression format.

Syntax

when : "Boolean expression" ;

Boolean expression

A Synopsys logic expression.

For a list of Boolean operators, see Table 3-3.

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

\[
\begin{align*}
\text{index}_1 & ("f\text{l}o\text{a}\text{t}, \ldots, f\text{l}o\text{a}\text{t}"); /* optional */ \\
\text{index}_2 & ("f\text{l}o\text{a}\text{t}, \ldots, f\text{l}o\text{a}\text{t}"); /* optional */ \\
\end{align*}
\]

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

\[
\begin{align*}
\text{index}_1 & ("0.0, 5.0, 20.0"); \\
\text{index}_2 & ("0.0, 1.0, 2.0"); \\
\end{align*}
\]

values Complex Attribute

You use this complex attribute to specify the nets’ maximum toggle rates.

Syntax

\[
\begin{align*}
\text{values} & : ("f\text{l}o\text{a}\text{t}, \ldots, f\text{l}o\text{a}\text{t}"); \\
\end{align*}
\]

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)

\[
\begin{align*}
\text{values} & : ("1.5, 1.0, 0.5"); \\
\end{align*}
\]

Example (Two-Dimensional Table)

\[
\begin{align*}
\text{values} & : ("2.0, 1.0, 0.5", "1.5, 0.75, 0.33","1.0, 0.5, 0.15"); \\
\end{align*}
\]

desk_max_toggle_rate Group

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

\[
\begin{align*}
\text{library} (\text{name}) | \\
\text{cell} (\text{name}) | \\
\text{pin} (\text{name}) | \\
\text{electromigration} () | \\
\text{em_max_toggle_rate}(\text{em_template_name}) | \\
\text{... em_max_toggle_rate description...} \\
\end{align*}
\]
3.2.4 hyperbolic_noise_above_high Group

This optional group describes a noise immunity region as a hyperbolic curve when the input is high and the noise is over the high voltage rail.

You specify a hyperbolic_noise_above_high group in a pin group, as shown here:

```
library (name) {
  cell (name) {
    pin (name) {
      direction : input | inout ;
      hyperbolic_noise_above_high () {
        ... hyperbola form description ... 
      }
    }
  }
}
```

Simple Attributes

area_coefficient Simple Attribute

The area_coefficient attribute specifies the area coefficient used to describe a noise immunity curve in hyperbola form.

Syntax

```
area_coefficient: valuefloat ;
```

value

A positive floating-point number. The unit is calculated as the library unit of voltage times the library unit of time.

Example

```
area_coefficient : 1.1 ;
```

height_coefficient Simple Attribute

The height_coefficient attribute specifies the height coefficient used to describe a noise immunity curve in hyperbola form.

Syntax

```
height_coefficient: valuefloat ;
```

value

A positive floating-point number. The unit is the library unit of voltage.

Example

```
height_coefficient : 0.4 ;
```

width_coefficient Simple Attribute

The width_coefficient attribute specifies the width coefficient used to describe a noise immunity curve in hyperbola form.
Syntax

width_coefficient: value_float;

value

A positive floating-point number. The unit is the library unit of time.

Example

width_coefficient : 0.01;

Example

hyperbolic_noise_above_high () {
  area_coefficient : 1.1;
  height_coefficient : 0.4;
  width_coefficient : 0.01;
}

3.2.5 hyperbolic_noise_below_low Group

This optional group describes a noise immunity region as a hyperbolic curve when the input is low and the noise is below the low voltage rail.

For information about the group syntax and attributes, see "hyperbolic_noise_above_high Group".

3.2.6 hyperbolic_noise_high Group

This optional group describes a noise immunity region as a hyperbolic curve when the input is high and the noise is below the high voltage rail.

For information about the group syntax and attributes, see "hyperbolic_noise_above_high Group".

3.2.7 hyperbolic_noise_low Group

This optional group describes a noise immunity region as a hyperbolic curve when the input is low and the noise is over the low voltage rail.

For information about the group syntax and attributes, see "hyperbolic_noise_above_high Group".

3.2.8 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 ...
      }
      ...
    }
  }
}

Simple Attributes

equal_or_opposite_output
falling_together_group
power_level
related_pin
rising_together_group
switching_interval
switching_together_group
when
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 three 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.

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

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

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

```plaintext
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:

```plaintext
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:

```plaintext
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:

```plaintext
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:

```plaintext
internal_power() {

```
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.

calling_together_group Simple Attribute

The `calling_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" for details.

Together, the `calling_together_group` and `switching_interval` attribute settings determine the level of power consumption.

Syntax

```
calling_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 () {
      calling_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**

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

- **name**
  
  Name of the power rail defined in the power supply group.

**Example**

```plaintext
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**

```plaintext
related_pin : "name | name_list" ;
```

- **name | name_list**
  
  The name of the input or output pin or pins.

**Example**

```plaintext
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.

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" for details.

Together, the `rising_together_group` attribute and `switching_interval` attribute settings determine the level of power consumption.

**Syntax**

```plaintext
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**

```plaintext
cell (foo) {
```
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", "rising_together_group Simple Attribute", and "switching_together_group Simple Attribute".

**Syntax**

```
switching_interval : valuefloat;
```

**value**

A floating-point number that represents the time interval during which two or more pins that share logic are transitioning together.

**Example**

```
pin (Z) {
  direction : output;
  internal_power () {
    switching_together_group : "AB";
    /*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" for details.

**Syntax**

```
switching_together_group : "list of pins";
```

**list of pins**

The names of the input or output pins that share logic, are either falling or rising during the same time period, and are not affecting power consumption.
when Simple Attribute

The `when` attribute specifies the state-dependent condition that determines whether this power table is accessed.

You can use the `when` attribute to define one-, two-, or three-dimensional tables in the `internal_power` group. You can also use the `when` attribute in the `power`, `fall_power`, and `rise_power` groups.

Note:

If you want to use the same Boolean expression for multiple `when` statements in an `internal_power` group, you must specify a different power rail for each `internal_power` group.

**Syntax**

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

`Boolean expression`

The name or names of the input and output pins with corresponding Boolean operators.

Table 3-3 lists the Boolean operators valid in a `when` statement.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.</code></td>
<td>invert previous expression</td>
</tr>
<tr>
<td><code>!</code></td>
<td>invert following expression</td>
</tr>
<tr>
<td><code>^</code></td>
<td>logical XOR</td>
</tr>
<tr>
<td><code>*</code></td>
<td>logical AND</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>logical AND</td>
</tr>
<tr>
<td>space</td>
<td>logical AND</td>
</tr>
<tr>
<td><code>+</code></td>
<td>logical OR</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>1</code></td>
<td>signal tied to logic 1</td>
</tr>
<tr>
<td><code>0</code></td>
<td>signal tied to logic 0</td>
</tr>
</tbody>
</table>

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

**Example**

```
when : "A B" ;
```

**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 (namestring) {  
  pin (namestring) {  
    internal_power () {  
      fall_power (template name) {  
        ... fall power description ...  
      }  
    }  
  }  
}  
```
Complex Attributes

```c
index_1 (*float, ..., float*) /* lookup table */
index_2 (*float, ..., float*) /* lookup table */
index_3 (*float, ..., float*) /* lookup table */
values (*float, ..., float*) /* lookup table */
orders (*integer, ..., integer*) /* polynomial */
coefs (*float, ..., float*) /* polynomial */
```

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.

Example 3-5 shows cells that contain internal power information in the `pin` group.

Group Statement

```c
domain (name) {}
```

```c
name

References a domain group defined in the `power_poly_template` group or the `power_lut_template` group.
```

power Group

Use the power group to define power when the rise power equals the fall power for a particular pin. You specify a `power` group within an `internal_power` group in a `pin` group at the cell level, as shown here:

Syntax

```c
library (name) {
  cell (name) {
    pin (name) {
      internal_power () {
        power (template name) {
          ... power template description ...
        }
      }
    }
  }
}
```

Complex Attributes

```c
index_1 (*float, ..., float*)
index_2 (*float, ..., float*)
index_3 (*float, ..., float*)
values (*float, ..., float*)
orders (*integer, ..., integer*) /* polynomial */
coefs (*float, ..., float*) /* polynomial */
```
**float**

Floating-point numbers that identify the amount of energy per transition, either rise or fall, the cell consumes internally.

You convert the *values* attribute to power consumption by multiplying the unit by the factor transition or per-unit time, as follows:

- nindex_1 floating-point numbers if the table is one-dimensional
- nindex_1 x nindex_2 floating-point numbers if the table is two-dimensional
- nindex_1 x nindex_2 x nindex_3 floating-point numbers if the table is three-dimensional

nindex_1, nindex_2, and nindex_3 are the size of index_1, index_2, and index_3 in this group or in the power_lut_template group it inherits. Quotation marks (" ") enclose a group. Each group represents a row in the table.

This power is accessed when the pin has a rise transition or fall transition. The values in the table specify the average power per transition.

**Example 3-5** shows cells that contain power information in the internal_power group in a pin group.

**Group**

domain (name) {
}

**name**

References a domain group defined in the power_poly_template group or the power_lut_template group.

**rise_power Group**

The *rise_power* group defines the power associated with a rise transition on a pin. You specify a *rise_power* group in an internal_power group in a pin group, as shown here:

**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");
orders ("integer, ..., integer"); /* polynomial */
coefs ("float, ..., float"); /* polynomial */

**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:
floating-point numbers if the table is one-dimensional
floating-point numbers if the table is two-dimensional
floating-point numbers if the table is three-dimensional

\( n_{index_1}n_{index_2}, \text{ and } n_{index_3} \) 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.

**Example 3-5** shows cells that contain internal power information in the `pin` group.

```plaintext
Group

domain (name) {}

name

References a domain group defined in either the `power_poly_template` group or the `power_lut_template` group.

**Example 3-5** A Library With Internal Power

```
pin(D) {
  direction : input;
  ...
}

pin(S) {
  direction : input;
  ...
}

pin(R) {
  direction : input;
  ...
}

pin(Q) {
  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", \\
        "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) {
    related_pin : "SR";
  }
  ...
}

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 : "SR";
  }
  ...
  ...
}

3.2.9 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.

Example

max_cap {} {
  ...
}
3.2.10 max_trans Group

Use the `max_trans` group to describe the maximum transition information for output and inout pins.

**Syntax**

```plaintext
library (name) {
  cell (name) {
    pin (name) {
      max_trans (template_name) {
        ... transition description ...
      }
    }
  }
}
```

*template_name*

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

**Complex Attributes**

- variable_1_range
- variable_2_range
- variable_n_range
- orders
- coefs

**Example**

```plaintext
max_trans () {
  ...
}
```

3.2.11 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**

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

**Example**

```plaintext
pin(A) {
  ...
  min_pulse_width() {
    constraint_high : 3.0;
    constraint_low : 3.5;
    when : "SE";
    sdf_cond : "SE == 1'B1";
  }
}
```

**Simple Attributes**
constraint_high
constraint_low
when
sdf_cond

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 Synopsys logic expression format.

Syntax

when : "Boolean expression";

Boolean expression
A logic expression.
For a list of Boolean operators, see Table 3-3.

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";

3.2.12 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";
}

Simple Attributes

constraint

when

sdf_cond

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 Synopsys logic expression format.

Syntax

when : "Boolean expression";

Boolean expression

A logic expression.

For a list of Boolean operators, see Table 3-3.

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 == 'b1'" ;```

### 3.2.13 pin_capacitance Group

In a pin group, the pin_capacitance group supports polynomial equation modeling to represent capacitance, rise_capacitance, fall_capacitance, rise_capacitance_range, and fall_capacitance_range.

The existing single value capacitance, rise_capacitance, fall_capacitance, rise_capacitance_range and fall_capacitance_range attributes and the new pin_capacitance group can co-exist on one pin. The syntax for the pin_capacitance group is the same used for the delay model, except that the variables used in the format are temperature and voltage (including power rails).

The pin_capacitance group supports only the scalable polynomial delay model.

**Note:**

The capacitance group is required in the pin_capacitance group.

**Group Statements**

```capacitance
rise_capacitance
fall_capacitance
fall_capacitance_range
rise_capacitance_range```

**Syntax**

```pin_capacitance() {...pin_capacitance description ...}
```

**capacitance Group**

Use the capacitance group to define the load of an input, output, inout, or internal pin. This group is required in the pin_capacitance group.

**Syntax**

```capacitance() {...capacitance description ...}
```

**Example**

```capacitance(cap) {
    orders (*1, 1*);
    coefs("1, 2, 3, 4");
}
```

**fall_capacitance Group**

Use the fall_capacitance group to define the load of an input, output, inout, or internal pin when its signal is falling. You must set a value for both the rise_capacitance and fall_capacitance groups.

The value you set for a fall_capacitance group overrides the value you set for a fall_capacitance simple attribute.
Syntax

`fall_capacitance() { ...capacitance description ...}`

Example

```lisp
fall_capacitance(fall_cap) {
  orders (*1, 1*);
  coefs (*1, 2, 3, 4*);
}
```

`rise_capacitance Group`

Use the `rise_capacitance` group to define the load of an input, output, inout, or internal pin when its signal is rising. For more information, see "Fall Capacitance Group".

`fall_capacitance_range Group`

This group describes the range for temperature and voltage (including voltage rails) during fall transitions. Only one `fall_capacitance_range` or `rise_capacitance_range` group is allowed inside the `pin_capacitance` group. The rise and fall capacitance range groups are optional.

Syntax

`fall_capacitance_range() { ... values description ...}`

Groups

`lower upper`

`lower Group`

For `pin_capacitance`, use this group to specify a range of minimum and maximum float values or polynomials as a function of temperature and voltage (including power rails).

You must define both the `lower` and `upper` groups in a `rise_capacitance_range` or `fall_capacitance_range` group.

Syntax

`lower (poly_template_name id) { ... values description ...}`

Example

```lisp
lower(cap) {
  ...
  orders (*1, 1*);
  coefs (*1, 2, 3, 4*);
}
```

`Complex Attributes`

- `variable_1_range`
- `variable_2_range`
- `variable_n_range`
- `orders`
- `coefs`

`variable_n_range Complex Attribute`

Use the `variable_n_range` attribute to specify the range of the value for the n th variable in the `variables` attribute.

Syntax
variable_n_range(min_1float, max_2float);

min, max

Floating-point number pairs that specify the value range.

**Example**

```plaintext
fall_capacitance_range () {
  lower(cap) {
    ...
    orders("1, 1");
    coefs("1, 2, 3, 4");
  }
  upper(cap) {
    ...
    orders("1, 1");
    coefs("1, 2, 3, 4");
  }
}
```

**coefs Complex Attribute**

Use the `coefs` attribute to specify a list of the coefficients you use in a polynomial. For more information, see "coefs Complex Attribute".

**orders Complex Attribute**

Use the `orders` attribute to specify the order for the variables for the polynomial. For more information, see "orders Complex Attribute".

**upper Group**

For `pin_capacitance`, use this group to specify a range of minimum and maximum float values or polynomials as a function of temperature and voltage (including power rails). For more information, see "lower Group".

**rise_capacitance_range Group**

This group describes the range of pin capacitance as a function of temperature and voltage (including voltage rails) during rise transitions for the signal. For more information, see "fall_capacitance_range Group".

**Example**

**Example 3-6** shows a sample library with extended `rise_capacitance_range` and `fall_capacitance_range` syntax.

**Example 3-6  Sample Library With pin_capacitance Group Values**

```plaintext
library(new_lib) {
  ...
  poly_template (PPT) {
    variables("temperature", "voltage");
    variable_1_range (-40.0, 100.0);
    variable_2_range (0.5, 3.5);
    domain (D1) {
      calc_mode : best;
    }
    domain (D2) {
      calc_mode : worst;
    }
  }
  ...
  cell (AN2) {
    pin(Y) {
      ...
      pin_capacitance () {
        /* default poly capacitance */
        capacitance(PPT) {
          orders("1, 1");
          coefs("0.11, 0.12, 0.13, 0.14");
          domain (D1) {
```
orders (*1, 1*);
  coefs (*0.21, 0.22, 0.23, 0.24*);
}

domain (D2) {
  orders (*1, 1*);
  coefs (*0.31, 0.32, 0.33, 0.34*);
}

rise_capacitance(PPT) {
  orders (*1, 1*);
  coefs (*0.11, 0.12, 0.13, 0.14*);
  domain (D1) {
    orders (*1, 1*);
    coefs (*0.21, 0.22, 0.23, 0.24*);
  }
  domain (D2) {
    orders (*1, 1*);
    coefs (*0.31, 0.32, 0.33, 0.34*);
  }
}

fall_capacitance(PPT) {
  orders (*1, 1*);
  coefs (*0.11, 0.12, 0.13, 0.14*);
  domain (D1) {
    orders (*1, 1*);
    coefs (*0.21, 0.22, 0.23, 0.24*);
  }
  domain (D2) {
    orders (*1, 1*);
    coefs (*0.31, 0.32, 0.33, 0.34*);
  }
}

rise_capacitance_range() {
  lower(PPT) {
    orders (*1, 1*);
    coefs (*0.01, 0.02, 0.03, 0.04*);
    domain (D1) {
      orders (*1, 1*);
      coefs (*0.11, 0.12, 0.13, 0.14*);
    }
    domain (D2) {
      orders (*1, 1*);
      coefs (*0.21, 0.22, 0.23, 0.24*);
    }
  }
  upper(PPT) {
    orders (*1, 1*);
    coefs (*0.21, 0.22, 0.23, 0.24*);
    domain (D1) {
      orders (*1, 1*);
      coefs (*0.31, 0.32, 0.33, 0.34*);
    }
    domain (D2) {
      orders (*1, 1*);
      coefs (*0.41, 0.42, 0.43, 0.44*);
    }
  }
}

fall_capacitance_range() {
  lower(PPT) {
    orders (*1, 1*);
    coefs (*0.01, 0.02, 0.03, 0.04*);
    domain (D1) {
      orders (*1, 1*);
      coefs (*0.11, 0.12, 0.13, 0.14*);
    }
    domain (D2) {
      orders (*1, 1*);
      coefs (*0.21, 0.22, 0.23, 0.24*);
    }
  }
  upper(PPT) {
    orders (*1, 1*);
    coefs (*0.21, 0.22, 0.23, 0.24*);
    domain (D1) {
      orders (*1, 1*);
      coefs (*0.31, 0.32, 0.33, 0.34*);
    }
    domain (D2) {
      orders (*1, 1*);
    }
  }
}
3.2.14 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

- receiver_capacitance1_fall
- receiver_capacitance1_rise
- receiver_capacitance2_fall
- receiver_capacitance2_rise

receiver_capacitance1_fall Group

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 composite current source (CCS) template. For information about using the group at the timing level, see "receiver_capacitance1_fall Group".

Syntax

```
receiver_capacitance1_fall (lu_template_nameid) {
  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, 0.0);
  }
  receiver_capacitance1_fall (LTT1) {
  ...
  }
  ...
}
```
For information about using the receiver_capacitance1_rise group, see the
description of the.

receiver_capacitance2_fall Group

For information about using the receiver_capacitance2_fall group, see the
description of.

receiver_capacitance2_rise Group

For information about using the receiver_capacitance2_rise group, see the
description of the.

3.2.15 timing Group in a pin Group

A timing group is defined within a pin group, as shown here. Note that the syntax
presents the attributes in alphabetical order by type of attribute.

Entering the names in the timing group attribute to identify timing arcs is optional.

Syntax

library (namestring) {
  cell (namestring) {
    pin (namestring) {
      timing (namestring) {
        ... timing description ...
      }
    }
  }
}

Simple Attributes

clock_gating_flag : true|false;
default_timing : true|false;
fall_resistance : float;
fpga_arc_condition : "Boolean expression";
fpga_domain_style : name;
interdependence_id : integer;
intrinsic_rise : float;
related_bus_equivalent : "name1 [name2 name3 ... ]";
related_bus_pins : "name1 [name2 name3 ... ]";
related_output_pin : name;
related_pin : "name1 [name2 name3 ... ]";
rise_resistance : float;
sdf_cond : "SDF expression";
sdf_cond_end : "SDF expression";
sdf_cond_start : "SDF expression";
sdf_edges : SDF edge type;
slope_fall : float;
slope_rise : float;
steady_state_resistance_above_high : float;
steady_state_resistance_below_low : float;
steady_state_resistance_high : float;
steady_state_resistance_low : float;
tied_off : Boolean;
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 | three_state_enable_rise |
  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";
}
Complex Attributes

```
fall_delay_intercept (integer, float); /* piecewise model only */
fall_pin_resistance (integer, float); /* piecewise model only */
mode
rise_delay_intercept (integer, float); /* piecewise model only */
rise_pin_resistance (integer, float); /* piecewise model only */
```

Group Statements

```
cell_degradation () {}
cell_fall () {}
cell_rise () {}
fall_constraint () {}
fall_propagation () {}
fall_transition () {};
noise_immunity_above_high () {};
noise_immunity_below_low () {};
noise_immunity_high () {};
noise_immunity_low () {};
output_current_fall () {};
output_current_rise () {};
propogated_noise_height_above_high () {};
propogated_noise_height_below_low () {};
propogated_noise_height_high () {};
propogated_noise_height_low () {};
propogated_noise_peak_time_ratio_above_high () {};
propogated_noise_peak_time_ratio_below_low () {};
propogated_noise_peak_time_ratio_high () {};
propogated_noise_peak_time_ratio_low () {};
propogated_noise_width_above_high () {};
propogated_noise_width_below_low () {};
propogated_noise_width_high () {};
propogated_noise_width_low () {};
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 () {};
steady_state_current_high () {};
steady_state_current_low () {};
steady_state_current_tristate () {};
```

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 ;
```

fall_resistance Simple Attribute

The `fall_resistance` attribute represents the load-dependent output resistance, or drive capability, for a logic 1-to-0 transition.

**Note:**

You cannot specify a resistance unit in the library. Instead, the resistance unit is derived from the ratio of the `time_unit` value to the `capacitive_load_unit` value.

**Syntax**

```
fall_resistance : valuefloat ;
```

**value**

A positive floating-point number in terms of delay time per load unit.

**Example**

```
fall_resistance : 0.18 ;
```

fpga_arc_condition Simple Attribute

The `fpga_arc_condition` attribute specifies a Boolean condition that enables a timing arc.

**Syntax**

```
fpga_arc_condition : conditionBoolean ) ;
```

**condition**

Specifies a Boolean condition. Valid values are true and false.

**Example**

```
fpga_arc_condition ;
```

fpga_domain_style Simple Attribute

Use this attribute to reference a `calc_mode` value in a `domain` group in a polynomial table.

**Syntax**

```
fpga_domain_style : "name" ;
```

**name**

The `calc_mode` value.

**Example**
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 interdependence_id attribute increases independently for each condition. Interdependence data can be specified in pin, bus, and bundle groups.

Syntax

interdependence_id : "nameenum" ;

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 non-conditional setup/hold constraints */
   setup/hold constrains
   /* new interdependence data for non-conditional 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 could 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 (non-conditional categorized as a special
  condition), a timing arc with a specific interdependence_id should 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 should be in consecutive order. That is, 1, 2, 3 is allowed,
  but 1, 2, 4 is not.

intrinsic_fall Simple Attribute

On an output pin, intrinsic_fall defines the 1-to-Z propagation time for a three-state-
disable timing type and the Z-to-0 propagation time for a three-state-enable timing type.

On an input pin, intrinsic_fall defines a setup, hold, or recovery timing requirement
for a logic 1-to-0 transition.

With intrinsic_rise, intrinsic_fall defines timing checks (rising and falling
transitions).

Syntax

\`
\text{intrinsic\_fall} : \text{value}\text{float} ;
\` 

\text{value}

A floating-point number that represents a timing requirement.

Example

\`
intrinsic\_fall : 0.75 ;
\`

intrinsic_rise Simple Attribute

On an output pin, intrinsic_rise defines the 0-to-Z propagation time for a three-state-
disable timing type and a Z-to-1 propagation time for a three-state-enable timing type.

On an input pin, intrinsic_rise defines a setup, hold, or recovery timing requirement
for a logic 0-to-1 transition.

With intrinsic_fall, intrinsic_rise defines timing checks (rising and falling
transitions).

Syntax

\`
\text{intrinsic\_rise} : \text{value}\text{float} ;
\` 

\text{value}

A floating-point number that represents a timing requirement.

Example

\`
intrinsic\_rise : 0.17 ;
\`

related_bus_equivalent Simple Attribute

The related_bus_equivalent attribute generates a single timing arc for all paths from
points in a group through an internal pin (I) to given endpoints.

Syntax

\`
\text{related\_bus\_equivalent} : "\text{name1 [name2 name3 ...]}" ;
\` 

Example

\`
related\_bus\_equivalent : a ;
\`
Example 3-7 shows an example using equivalent bus pins.

**Example 3-7 Equivalent Bus Pins**

cell(scell) {
    ... 
    bus(y) {
        bus_type : bus4;
        direction : output;
        timing() {
            related_bus_equivalent : a;
            ... 
        }
    }
    bus(a) {
        bus_type : bus4;
        direction : input;
        ... 
    }
}

related_bus_pins Simple Attribute

The `related_bus_pins` attribute defines the pin or pins that are the startpoint of the timing arc. The primary use of `related_bus_pins` is for module generators.

Note:

When a `related_bus_pins` attribute is within a `timing` group, the `timing` group must be within a `bus` or `bundle` group.

Syntax

```
related_bus_pins : "name1 [name2 name3 ...]" ;
```

Example

```
related_bus_pins : "A" ;
```

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 beginning 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 * B";
  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, it is necessary to define only one timing arc with two related pins.

**Example**

```plaintext
pin (Z) {
  direction: output;
  function: "(A * B) ^";
  timing () {
    related_pin: "A B";
    ... timing information ...
  }
}
```

When a bus name appears in a `related_pin` attribute, the bus members or range of members is distributed across all members of the parent bus. The width of the bus or the range must be the same as the width of the parent bus.

Pin names used in a `related_pin` statement can start with a nonalphabetic character.

**Example**

```plaintext
related_pin: "A 1B 2C";
```

**Note:**

It is not necessary to use the escape character, \ (backslash), with nonalphabetic characters.

rise_resistance Simple Attribute

The `rise_resistance` attribute represents the load-dependent output resistance, or drive capability, for a logic 0-to-1 transition.

**Note:**

You cannot specify a resistance unit in the library. Instead, the resistance unit is derived from the ratio of the `time_unit` value to the `capacitive_load_unit` value.

**Syntax**

```plaintext
rise_resistance : value float :

value

A positive floating-point number in terms of delay time per load unit.
```

**Example**

```plaintext
rise_resistance : 0.15 ;
```

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 Synopsys Boolean expression syntax. For a complete description of the valid syntax for these expressions, refer to 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**

One of these four edge types: noedge, start_edge, end_edge, or both_edges. The default is noedge.

**Example**

```
sdf_edges : sdf_edge_type;
```
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;

slope_fall Simple Attribute

The slope_fall attribute represents the incremental delay to add to the slope of the input waveform for a logic 1-to-0 transition.

Syntax

slope_fall : value_float;

value

A positive floating-point number multiplied by the transition delay resulting in slope delay.

Example

slope_fall : 0.8;

slope_rise Simple Attribute

The slope_rise attribute represents the incremental delay to add to the slope of the input waveform for a logic 0-to-1 transition.

Syntax

slope_rise : value_float;

value

A positive floating-point number multiplied by the transition delay resulting in slope delay.

Example

slope_rise : 1.0;

steady_state_resistance_above_high Simple Attribute

The steady_state_resistance_above_high attribute specifies a steady-state resistance value for a region of a current-voltage (I-V) curve when the output is high and the noise is over the high voltage rail.

Syntax
steady_state_resistance_above_high : value float;

value

A positive floating-point number that represents the resistance. The resistance unit is a function of the unit of time divided by the library unit of capacitance.

Example

steady_state_resistance_above_high : 200;

steady_state_resistance_below_low Simple Attribute

The steady_state_resistance_below_low attribute specifies a steady-state resistance value for a region of a current-voltage (I-V) curve when the output is low and the noise is below the low voltage rail.

Syntax

steady_state_resistance_below_low : value float;

value

A positive floating-point number that represents the resistance. The resistance unit is a function of the unit of time divided by the library unit of capacitance.

Example

steady_state_resistance_below_low : 100;

steady_state_resistance_high Simple Attribute

The steady_state_resistance_high attribute specifies a steady-state resistance value for a region of a current-voltage (I-V) curve when the output is high and the noise is below the high voltage rail.

Syntax

steady_state_resistance_high : value float;

value

A positive floating-point number that represents the resistance. The resistance unit is a function of the unit of time divided by the library unit of capacitance.

Example

steady_state_resistance_high : 1500;

steady_state_resistance_low Simple Attribute

The steady_state_resistance_low attribute specifies a steady-state resistance value for a region of a current-voltage (I-V) curve when the output is low and the noise is over the low voltage rail.

Syntax

steady_state_resistance_low : value float;

value

A positive floating-point number that represents the resistance. The resistance unit is a function of the unit of time divided by the library unit of capacitance.
Example

steady_state_resistance_low : 1100;

tied_off Simple Attribute

Used for noise modeling, the tied_off attribute allows you to specify the I-V characteristics and steady-state resistance values on tied-off cells.

Syntax

tied_off : Boolean;

Boolean

Valid values are true and false.

Example

tied_off : true;

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.

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 said to be unate if a rising (falling) change on a positive (negative) unate input variable causes the output function variable to rise (fall) or not change. For a non-unate 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 | preset | clear | hold_rising | hold_falling | setup_rising | setup_falling | recovery_rising | recovery_falling | skew_rising | skew_falling | removal_rising | removal_falling | min_pulse_width | minimum_period | max_clock_tree_path | min_clock_tree_path | non_seq_setup_rising | non_seq_setup_falling | non_seq_hold_rising | non_seq_hold_falling | nochange_high_high | nochange_high_low | nochange_low_high | nochange_low_low ;

Combinational Timing Arcs

The timing type and timing sense define the signal propagation pattern. The default timing type is combinational.

Table 3-4 Combinational Timing Arcs

<table>
<thead>
<tr>
<th>Timing type</th>
<th>Positive_Unate</th>
<th>Negative_Unate</th>
<th>Non_Unate</th>
</tr>
</thead>
<tbody>
<tr>
<td>combinational</td>
<td>R-&gt;R,F-&gt;F</td>
<td>R-&gt;F,F-&gt;R</td>
<td>{R,F}&gt;({R,F})</td>
</tr>
<tr>
<td>combinational_rise</td>
<td>R-&gt;R</td>
<td>F-&gt;R</td>
<td>{R,F}&gt;R</td>
</tr>
<tr>
<td>combinational_fall</td>
<td>F-&gt;F</td>
<td>R-&gt;F</td>
<td>{R,F}&gt;F</td>
</tr>
<tr>
<td>three_state_disable</td>
<td>R-&gt;{0Z,1Z}</td>
<td>F-&gt;{0Z,1Z}</td>
<td>{R,F}-&gt;{0Z,1Z}</td>
</tr>
<tr>
<td>three_state_disable_rise</td>
<td>R-&gt;{Z0,Z1}</td>
<td>F-&gt;{Z0,Z1}</td>
<td>{R,F}&gt;{Z0,Z1}</td>
</tr>
<tr>
<td>three_state_disable_fall</td>
<td>R-&gt;0Z</td>
<td>F-&gt;1Z</td>
<td>{R,F}&gt;1Z</td>
</tr>
<tr>
<td>three_state_enable_rise</td>
<td>R-&gt;Z1</td>
<td>F-&gt;Z1</td>
<td>{R,F}&gt;Z1</td>
</tr>
<tr>
<td>three_state_enable_fall</td>
<td>R-&gt;Z0</td>
<td>F-&gt;Z0</td>
<td>{R,F}&gt;Z0</td>
</tr>
</tbody>
</table>

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 output pin.
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.

**minimum_pulse_width**

This value, together with the minimum_period value, lets you specify the
minimum pulse width for a clock pin. The timing check is performed on the pin itself, so the related pin should be the same. As with other timing checks, you can include rise and fall constraints.

**minimum_period**

This value, together with the `minimum_pulse_width` value, lets you specify the minimum pulse width for a clock pin. The timing check is performed on the pin itself, so the related pin should be the same. As with other timing checks, you can include rise and fall constraints.

**max_clock_tree_path**

Used in timing groups under a clock pin. Defines the maximum clock tree path constraint.

**min_clock_tree_path**

Used in timing groups under a clock pin. Defines the minimum clock tree path constraint.

**Example***

Example 3-8 shows a sample library with the `timing_type` attribute and `minimum_pulse_width` and `minimum_period` values.

**Example 3-8 Sample Library with timing_type Statements**

```liberty
library(ASIC) {
  ...
  delay_model : table_lookup;
  ....
  lu_table_template(pulse_width_template) {
    variable_1 : related_pin_transition;
    index_1 (*1.0, 2.0, 3.0*);
  }
  cell(flop) {
    ...
    pin(CK) {
      direction : input;
      capacitance : 0.00707171;
      timing() {
        timing_type : "min_pulse_width";
        related_pin : "CK";
        ...
        rise_constraint("pulse_width_template") {
          index_1 (*0.000000, 1.000000, 2.000000*);
          values (*6.000000, 6.250000, 7.2500000*);
        }
        fall_constraint("pulse_width_template") {
          index_1 (*0.000000, 1.000000, 2.000000*);
          values (*6.000000, 6.250000, 7.2500000*);
        }
      }
      timing() {
        timing_type : "minimum_period";
        related_pin : "CK";
        rise_constraint("pulse_width_template") {
          index_1 (*0.000000, 1.000000, 2.000000*);
          values (*6.000000, 6.250000, 7.2500000*);
        }
        fall_constraint("pulse_width_template") {
          index_1 (*0.000000, 1.000000, 2.000000*);
          values (*6.000000, 6.250000, 7.2500000*);
        }
      }
    }
  }
  /* end cell */
}/* end 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 you do this, the delay is measured from the second last transition of the sensitization
vector to the final output transition, in other words from the transition of vector 5 to vector 7.

Note:
You will get an error if you specify a value of 0 for the wave_rise_sampling_index
attribute.

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 and , respectively). Both groups can be placed in
pin, bus, and bundle groups. The when attribute also appears in the power,
fall_power, and rise_power groups.

Syntax

when : "Boolean expression" ;

Boolean expression

A Boolean expression containing names of input, output, inout, and internal
pins.

Example

when : "CD * SD" ;

State-Dependent Timing

In the timing group of a technology library, you can specify state-dependent delays that
correspond to entries in OVI SDF 2.1 syntax. In state-dependent timing, the when attribute
defines a conditional expression on which a timing arc is dependent to activate a path.

Conditional Timing Check

In a conditional timing check, the when attribute defines check-enabling conditions for
timing checks such as setup, hold, and recovery.

Conditional Timing Check in VITAL Models

The when attribute is used in modeling timing check conditions for VITAL models, where, if
you define when, you must also define sdf_cond.

Syntax

when : "Boolean expression" ;

Boolean expression

A valid logic expression as defined in Table 3-3.

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**

```vhdl
when_end : "Boolean expression" ;
```

**Boolean expression**

A Boolean expression containing names of input, output, inout, and internal pins.

**Example**

```vhdl
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**

```vhdl
when_start : "Boolean expression" ;
```

**Boolean expression**

A Boolean expression containing the names of input, output, inout, and internal pins.

**Example**

```vhdl
when_start : "CD * SD" ;
```

**fall_delay_intercept Complex Attribute**

For piecewise models only, the `fall_delay_intercept` attribute defines the intercept for vendors using slope- or intercept-type timing equations. The value of the attribute is added to the falling edge in the delay equation.

**Syntax**

```vhdl
fall_delay_intercept ("integer, float") ;
```

**Examples from a CMOS library:**

```vhdl
fall_delay_intercept (0,"1.0") ; /* piece 0 */
f fall_delay_intercept (1,"0.0") ; /* piece 1 */
f fall_delay_intercept (2,"-1.0") ; /* piece 2 */
```

**fall_pin_resistance Complex Attribute**

For piecewise models only, the `fall_pin_resistance` attribute defines the drive resistance applied to pin loads in the falling edge in the transition delay equation.

**Syntax**

```vhdl
fall_pin_resistance (integer, "float") ;
```

**Examples From a CMOS library:**

```vhdl
fall_pin_resistance (0,"0.25") ; /* piece 0 */
f fall_pin_resistance (1,"0.50") ; /* piece 1 */
f fall_pin_resistance (2,"1.00") ; /* piece 2 */
```

**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 3-9 shows a `mode` description.

**Example 3-9  A mode Description**

```
pin(my_outpin) {
    direction : output;
    timing() {
        related_pin : b;
        timing_sense : non_unate;
        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 3-10 shows multiple `mode` descriptions.

**Example 3-10  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) {
```

Liberty Reference Manual (Version 2007.03) 220
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() {
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);

rise_delay_intercept Complex Attribute

For piecewise models only, the rise_delay_intercept attribute defines the intercept for vendors using slope- or intercept-type timing equations. The value of the attribute is added to the rising edge in the delay equation.

Syntax

rise_delay_intercept (integer, "float");

Examples from a CMOS library:

rise_delay_intercept (0,"1.0"); /* piece 0 */
rise_delay_intercept (1,"0.0"); /* piece 1 */

rise_pin_resistance Complex Attribute

For piecewise models only, the rise_pin_resistance attribute defines the drive resistance applied to pin loads in the rising edge in the transition delay equation.

Syntax

rise_pin_resistance (integer, "float");

Examples from a CMOS library:

rise_pin_resistance (0,"0.25"); /* piece 0 */
rise_pin_resistance (1,"0.50"); /* piece 1 */
rise_pin_resistance (2,"1.00"); /* piece 2 */

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) {
... sensitzation(sensi_2in_1out) {
    pin_names (IN1, IN2, OUT);
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 a short duration between one vector to the next in order to characterize special purpose cells and pessimistic timing characterization.

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).

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

```plaintext
pin (output pin name) {
    timing () {
        cell_degradation (template name) {
            ...cell_degradation description...
        }
    }
    ...
}
```

Complex Attributes

- `coefs` /* polynomial model */
- `orders` /* polynomial model */
- `index_1` /* lookup table */
- `values` /* lookup table */
- `variable_n_range` /* polynomial model */

Group

domain

coops Complex Attribute

Use the `coefs` attribute to specify a list of the coefficients you use in a polynomial to characterize timing information. This attribute is required when you specify a scalable polynomial delay model. The coefficients are represented in the .lib file and saved in the database in column-first order. If any is term missing in the polynomial, you must insert a 0 (zero) in the corresponding position in the `coefs` attribute to ensure correct processing of the coefficients.

Note:

For a piecewise polynomial, define the `coefs` attribute inside the `domain` group inside the timing group that defines the range of coefficients.

Syntax

```plaintext
pin (output pin name) {
    timing () {
        ...coefs("float, ..., float")
        ...
    }
}
```

Example

```plaintext
timing () {
    coefs ("1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,
           10.0, 11.0, 12.0")
}
```

orders Complex Attribute
Use the `orders` attribute to specify the order of the variables you use in a polynomial to characterize timing information. This attribute is required in the `timing` group when you specify a scalable polynomial delay model. The `timing` group can be any timing group using polynomial delay modeling.

**Note:**

For a piecewise polynomial, define the `orders` attribute inside the `domain` group inside the `timing` group.

**Syntax**

```plaintext
pin (output_pin_name) {
  timing () {
    orders("integer, ..., integer")
    ...
  }
  ...
}
```

**Example**

```plaintext
timing () {
  orders("2, 1, 1") ;
  ...
}
```

**variable_n_range Complex Attribute**

Use the `variable_n_range` attribute to specify the order of the variables for the polynomial to characterize timing information. This attribute is required in the `timing` group when you specify a scalable polynomial delay model. The `timing` group can be any timing group using polynomial delay modeling.

**Note:**

For a piecewise polynomial, define the `orders` attribute inside the `domain` group inside the `timing` group.

**Syntax**

```plaintext
pin (output_pin_name) {
  timing () {
    variable_n_range(float, float) ;
    ...
  }
  ...
}
```

**Example**

```plaintext
timing () {
  variable_n_range () ;
}
```

**domain Group**

In the case of a piecewise polynomial and multiple tables defined by the `calc_mode` attribute, use one or more `domain` groups in the `timing` group to specify subsets of the polynomial template and tables of the lookup table templates.

**Note:**

For a piecewise polynomial, define the `orders` and `coefs` attributes inside the `domain` group inside the `timing` group.

If the table is specified by `calc_mode`, the `values` attribute must be used inside the `timing` group. You can also use the `calc_mode` and `values` attributes inside the
timing group to override the template values.

Note:

A domain name is required.

Syntax

library (namestring)
    cell (namestring)
    pin (namestring)
        timing ()
            domain (namestring)
                ...
                domain description...
            }
    }
}

Simple Attribute

calc_mode

Complex Attributes

coeefs
orders
variable_n_range

For information about the syntax and usage of the above attributes see the "cell_degradation Group".

Example 3-11 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");
        }
    ...
}

Example 3-12 Specifying cell_degradation in a Polynomial

domain (D1)
    orders ("2, 1, 1");
    coefs ("1.000, 2.000, 3.000, 4.000, 5.000, 6.000, 7.000,
            8.000, 9.000, 10.000, 11.000, 12.000");
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
}

cell_fall Group

The cell_fall 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.
The cell_fall group is defined at the timing group level, as shown here:

**Syntax**

```
library (namestring) {
  cell (namestring) {
    pin (namestring) {
      timing () {
        cell_fall (namestring) {
          ... cell fall description...
        }
        }
      }
    }
  }
}
```

**Complex Attributes**

- index_1 ("float, ..., float")
- index_2 ("float, ..., float")
- index_3 ("float, ..., float")
- values ("float, ..., float", ..., "float, ..., float")

**Group**

- domain

**domain Group**

For information about the domain group syntax and usage, see the description in the "domain Group".

**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".

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

```plaintext
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing (){
        cell_rise (name_string){
          ... cell rise description ...
        }
      }
    }
  }
}
```

Complex Attributes

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

Group
domain
domain Group

For information about the domain group syntax and usage, see the description in the "domain Group".

Examples from a CMOS library

```plaintext
cell_rise(cell_template) {
  values("0.00, 0.23", "0.11, 0.28");
}

cell_rise(cell_template) {
  values("0.00, 0.25", "0.11, 0.28");
}
```

Each lookup table has an associated string name to indicate where in the library group it is to be used. The name must be the same as the string name you previously defined in the library lu_table_template. For information about the lu_table_template syntax, see the description in "lu_table_template Group".

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, \( n_{\text{index}_1} \) and \( n_{\text{index}_2} \) are the sizes of \( \text{index}_1 \) and \( \text{index}_2 \) of the \( \text{lu_table_template} \) group. Group together \( n_{\text{index}_1} \) and \( n_{\text{index}_2} \) by using quotation marks (" ").

In a three-dimensional table, \( n_{\text{index}_1} \times n_{\text{index}_2} \times n_{\text{index}_3} \) are the sizes of \( \text{index}_1, \text{index}_2, \) and \( \text{index}_3 \) of the \( \text{lu_table_template} \) group. Group together \( n_{\text{index}_1}, n_{\text{index}_2}, \) and \( n_{\text{index}_3} \) by using quotation marks (" ").

Each group represents a row in the table. The number of floating-point numbers in a group must equal \( n_{\text{index}_2} \), and the number of groups in the \( \text{values} \) complex attribute must equal \( n_{\text{index}_1} \). The floating-point \( n_{\text{index}_2} \) 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 \( \text{index}_3 \) attribute is part of the functionality that supports three-dimensional tables.

**compact_ccs_rise and compact_ccs_fall Groups**

The \( \text{compact_ccs_rise} \) and \( \text{compact_ccs_fall} \) 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**

- \( \text{base_curves_group} \)

**Complex Attribute**

- \( \text{values} \)

The \( \text{base_curves_group} \) attribute is optional at this level when \( \text{base_curves_name} \) is the same as that defined in the \( \text{compact_lut_template} \) that is being referenced by the \( \text{compact_ccs_rise} \) or \( \text{compact_ccs_fall} \) group.

**Syntax**

```
base_curves_group : "base_curves_name" ;
```

**Example**

```
base_curves_group : "ctbct1" ;
```

**values Complex Attribute**

The \( \text{values} \) attribute defines the compact CCS timing data values. The values are determined by the \( \text{index}_3 \) values.

**Syntax**
Example

```plaintext
values ("0.1, 0.5, 0.6, 0.8, 1.0", 
  "0.15, 0.55, 0.65, 0.85, 2.0", 
  "0.2, 0.6, 0.7, 0.9, 2.0", 
  "0.25, 0.65, 0.75, 0.95, 4.0");
```

fall_constraint Group

With the `rise_constraint` group, the `fall_constraint` group defines timing constraints (cell delay lookup tables) sensitive to clock or data input transition times. These constraint tables take the place of the `intrinsic_rise` and `intrinsic_fall` attributes used in other delay models.

The `fall_constraint` group is defined in a `timing` group, as shown here:

```plaintext
library (namestring) {
  cell (namestring) {
    pin (namestring) {
      timing () {
        fall_constraint (namestring) {
          ... fall constraint description...
        }
      }
    }
  }
}
```

Complex Attributes

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

Group

domain

domain Group

For information about the `domain` group syntax and usage, see the description in the "domain Group".

Example

```plaintext
fall_constraint (constraint_template) {
  values ("0.0, 0.14, 0.20", 
          "0.22, 0.24, 0.42", 
          "0.34, 0.38, 0.51");
}
```

```plaintext
rise_constraint (constraint_template) {
  values ("0.0, 0.13, 0.19", 
          "0.21, 0.23, 0.41", 
          "0.33, 0.37, 0.50");
}
```

Example 3.13 shows constraint in a timing model.

fall_propagation Group

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.

```plaintext
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*)

**Group**

- `domain`

**domain Group**

For information about the domain group syntax and usage, see the description in the "domain Group".

**Example**

```plaintext
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:

```plaintext
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.

**Group**

```plaintext
domain
```

**domain Group**

For information about the `domain` group syntax and usage, see the description in the "domain Group".

**Example**

```plaintext
fall_transition(tran_template) {
    values (*0.01, 0.11, 0.18, 0.40*);
}
```

**noise_immunity_above_high Group**

Use this optional group to describe a noise immunity curve when the input is high and the noise is over the high voltage rail.

You define the `noise_immunity_above_high` group in a timing group, as shown here:

```plaintext
library (name_string) {
    cell (name_string) {
        pin (name_string) {
            timing () {
                noise_immunity_above_high (template_name_string){
                    ... values description...
                }
            }
        }
    }
}
```

**template_name**

The name of a `noise_lut_template` group or a `poly_template` group.

**Complex Attributes**

- `coefs /* scalable polynomial only */`
- `orders /* scalable polynomial only */`
- `values /* lookup table only */`

**Group**

```plaintext
domain /* scalable polynomial only */
```

**coefs Complex Attribute**

Use the `coefs` attribute to specify a list of the coefficients you use in a polynomial to characterize noise immunity information. This attribute is required in the `noise_immunity_above_high` group when you specify a scalable polynomial model. The coefficients are represented in the .lib file and saved in the database in column-first order. If any term is missing in the polynomial, you must insert a 0 (zero) in the corresponding position in the `coefs` attribute to ensure correct processing of the coefficients.

**Note:**
For a piecewise polynomial, the `coefs` attribute must be defined inside the `domain` group inside the `noise_immunity_above_high` group that defines the range of coefficients.

**Syntax**

```plaintext
pin (input_pin_name) {
    timing () {
        noise_immunity_above_high (poly_template_namestring){
            coefs("float, ..., float")
        }
    }
    ...
}
```

**orders Complex Attribute**

Use the `orders` attribute to specify the order for the variables for the polynomial to characterize noise immunity information. This attribute is required in the `noise_immunity_above_high` group when you specify a scalable polynomial model.

**Note:**

For a piecewise polynomial, define the `orders` attribute inside the `domain` group inside the `noise_immunity_above_high` group.

**Syntax**

```plaintext
pin (input_pin_name) {
    timing () {
        noise_immunity_above_high (poly_template_namestring){
            orders("integer, ..., integer")
        }
    }
    ...
}
```

**Example**

```plaintext
pin (Z) {
    timing () {
        noise_immunity_above_high (){
            orders("2, 1, 1");
            coefs("1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,
            10.0, 11.0, 12.0");
        }
    }
    ...
}
```

**domain Group**

For information about the `domain` group syntax and usage, see the description in the "domain Group".

**noise_immunity_below_low Group**

Use this optional group to describe a noise immunity curve when the input is low and the noise is below the low voltage rail.

For information about the `noise_immunity_below_low` group syntax and attributes, see "noise_immunity_above_high Group".

**noise_immunity_high Group**

Use this optional group to describe a noise immunity curve when the input is high and the noise is below the high voltage rail.

For information about the `noise_immunity_high` group syntax and attributes, see "noise_immunity_above_high Group".
noise_immunity_low Group

Use this optional group to describe a noise immunity curve when the input is low and the noise is over the low voltage rail.

For information about the noise_immunity_low group syntax and attributes, see "noise_immunity_above_high Group".

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 here.

```
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, as shown here.

```
library (name_string) {
  cell (name_string) {
    pin (name_string) {
      timing () {
        ... ...
      }
    }
  }
}
```
output_current_fall (namestring){
  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, 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".

propagated_noise_height_above_high Group

Use this group to describe noise propagation through a cell when the input is high and the noise is over the high voltage rail.

You define the propagated_noise_above_high group in a timing group, as shown here:

... timing () {
  propagated_noise_height_above_high (template_namestring){
    ... values description...
  }
}

template_name

The name of a propagation_lut_template group or a poly_template group.

Complex Attributes

coops /* scalable polynomial only */
orders /* scalable polynomial only */
values /* lookup table only */

Group

domain /* scalable polynomial only */

coops Complex Attribute

Use the coops attribute to specify a list of the coefficients you use in a polynomial to characterize propagated noise information. This attribute is required in the propagated_noise_height_above_high group when you specify a scalable polynomial model. The coefficients are represented in the .lib file and saved in the database in column-first order. If any term is missing in the polynomial, you must insert a 0 (zero) in the corresponding position in the coops attribute to ensure correct processing of the coefficients.
Note:

For a piecewise polynomial, define the `coefs` attribute inside the `domain` group inside the `propagated_noise_height_above_high` group that defines the range of coefficients.

Syntax

```plaintext
pin (input_pin_name)
{
    timing()
    {
        propagated_noise_height_above_high
        (poly_template_name)
        {
            coefs("float,...,float")
        }
    }
}
```

`orders` Complex Attribute

Use the `orders` attribute to specify the order of the variables for the polynomial to characterize noise immunity information. This attribute is required in the `propagated_noise_height_above_high` group when you specify a scalable polynomial model. For a piecewise polynomial, the `orders` attribute should be used inside the `domain` group inside the `propagated_noise_height_above_high` group.

Syntax

```plaintext
pin (input_pin_name)
{
    timing()
    {
        propagated_noise_height_above_high
        (poly_template_name)
        {
            orders("integer,...,integer")
        }
    }
}
```

Example

```plaintext
pin (Z)
{
    timing()
    {
        orders("2,1,1");
        coefs("1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,
               10.0,11.0,12.0");
    }
}
```

`domain` Group

For information about the `domain` group syntax and usage, see the description in the "domain Group".

`propagated_noise_height_below_low` Group

Use this group to describe noise propagation through a cell when the input is low and the noise is below the low voltage rail.

For information about the `propagated_noise_height_below_low` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_height_high` Group

Use this group to describe noise propagation through a cell when the input is high and the noise is below the high voltage rail.

For information about the `propagated_noise_height_high` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_height_low` Group

For information about the `propagated_noise_height_low` group syntax and attributes, see "propagated_noise_height_above_high Group".
Use this group to describe noise propagation through a cell when the input is low and the noise is over the low voltage rail.

For information about the `propagated_noise_height_low` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_peak_time_ratio_above_high Group`

Use this group to describe noise propagation through a cell when the input is high and the noise is over the high voltage rail.

For information about the `propagated_noise_peak_time_ratio_above_high` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_peak_time_ratio_below_low Group`

Use this group to describe noise propagation through a cell when the input is low and the noise is below the low voltage rail.

For information about the `propagated_noise_peak_time_ratio_below_low` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_peak_time_ratio_high Group`

Use this group to describe noise propagation through a cell when the input is high and the noise is below the high voltage rail.

For information about the `propagated_noise_peak_time_ratio_high` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_peak_time_ratio_low Group`

Use this group to describe noise propagation through a cell when the input is low and the noise is over the low voltage rail.

For information about the `propagated_noise_peak_time_ratio_low` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_width_above_high Group`

Use this group to describe noise propagation through a cell when the input is high and the noise is over the high voltage rail.

For information about the `propagated_noise_width_above_high` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_width_below_low Group`

Use this group to describe noise propagation through a cell when the input is low and the noise is below the low voltage rail.

For information about the `propagated_noise_width_below_low` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_width_high Group`

Use this group to describe noise propagation through a cell when the input is high and the noise is below the high voltage rail.

For information about the `propagated_noise_width_high` group syntax and attributes, see "propagated_noise_height_above_high Group".

`propagated_noise_width_low Group`

Use this group to describe noise propagation through a cell when the input is low and the noise is over the low voltage rail.

For information about the `propagated_noise_width_low` group syntax and attributes, see "propagated_noise_height_above_high Group".

`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".

**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");
```
Group

    domain


domain Group

For information about the domain group syntax and usage, see the description in the "domain Group".

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 (name_string) {
            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");

Group

    domain


domain Group

For information about the domain group syntax and usage, see the description in the "domain Group".
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_templatestring) { 
  values (index1float, index2float, index3float) ;
}
retain_fall_slew (retaining_time_templatestring) { 
  orders("variable1integer, variable2integer") ; 
  coefs("coefficient_1float ..., coefficient_nfloat") 
  variable_n_range(float, float) ;
}

retain_time_template

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

index1, index2, index3

Values to use for indexing the lookup table.

variable1, variable2

The orders of the variables for the polynomial.

coefficient_1, ..., coefficient_n

Specifies the coefficients used in the polynomial to characterize timing information.

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*);
      }
      ...
    }
  }
}
cell (cell_name) { 
  ...
  pin (pin_name) { 
    direction : output :
    ...
    timing () { 
      related_pin : "related_pin";
      ...
... retaining_fall (retaining_table_template) {
    orders ("1, 1");
    coefs (0.2407, 3.1568, 0.0129, 0.0143);
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
}
...
retain_fall_slew (retaining_time_template) {
    orders ("1, 1");
    coefs (0.2407, 3.1568, 0.0129, 0.0143);
    variable_1_range (0.01, 3.00);
    variable_2_range (0.01, 3.00);
}
...
}
}

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 (index1float, index2float, index3float);
}
retain_rise_slew (retaining_time_template string) {
    orders ("variable1integer, variable2integer");
    coefs ("coefficient_1float, ..., coefficient_nfloat");
    variable_n_range (float, float);
}

 retaining_time_template

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

index1float, index2float, index3float

Values to use for indexing the lookup table.

variable1, variable2

The orders of the variables for the polynomial.

coefficient_1float, ..., coefficient_nfloat

Specifies the coefficients used in the polynomial to characterize timing information.

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");
            }
            ...
        }
    }
}
cell (cell_name) {
... pin (pin_name) {
  direction: output:
... timing () {
    related_pin: "related_pin"
... retaining_rise (retaining_table_template) {
      orders ("1, 1");
      coefs ("0.2407, 3.1568, 0.0129, 0.0143");
      variable_1_range (0.01, 3.00);
      variable_2_range (0.01, 3.00);
    }
... retain_rise_slew (retaining_time_template) {
      orders ("1, 1");
      coefs ("0.2407, 3.1568, 0.0129, 0.0143");
      variable_1_range (0.01, 3.00);
      variable_2_range (0.01, 3.00);
    }
...}
}
}

rise_constraint Group

With the fall_constraint group, the rise_constraint group defines timing constraints (cell delay lookup tables) sensitive to clock or data input transition times. These constraint tables take the place of the intrinsic_rise and intrinsic_fall attributes used in the other delay models.

Syntax

library (namestring) {
  cell (namestring) {
    pin (namestring) {
      timing () {
        rise_constraint (namestring) {
          ...
        }
      }
    }
  }
}

Complex Attributes

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

Group
domain
domain Group

For information about the domain group syntax and usage, see the description in the "domain Group ".

Example

rise_constraint (constraint_template) {
  values ("0.0, 0.13, 0.19", \
Example 3-13 shows constraints in a timing model.

**Example 3-13 CMOS Nonlinear Timing Model Using Constraints**

```plaintext
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**

```plaintext
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")`

**Group**

- `domain`
domain Group

For information about the domain group syntax and usage, see the description in the "domain Group".

Example

```plaintext
fall_propagation (prop_template)
  values("0.00, 0.21", "0.14, 0.38");
rise_propagation (prop_template)
  values("0.05, 0.25", "0.15, 0.48");
```

rise_transition Group

The rise_transition group is defined in the timing group, as shown here:

```plaintext
library (namestring) {
  cell (namestring) {
    pin (namestring) {
      timing () {
        rise_transition (namestring) {
          ... 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.

Group domain

domain Group

For information about the domain group syntax and usage, see the description in the "domain Group".

Examples

```plaintext
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*);
}
```
steady_state_current_high Group

This optional group defines the current-voltage (I-V) characteristic for a cell timing arc by holding the output signal high.

You define the steady_state_current_high group in a timing group, as shown here:

```plaintext
library (name_string) {
    cell (name_string) {
        pin (name_string) {
            timing () {
                steady_state_current_high (template_name_string){
                    ...
                    values description...
                }
            }
        }
    }
}
```

**Template_name**

The name of an iv_lut_template group or a poly_template group.

**Complex Attributes**

- coefs /* scalable polynomial only */
- orders /* scalable polynomial only */
- values /* lookup table only */

**Group**

- domain /* scalable polynomial only */

**coefs Complex Attribute**

Use the coefs attribute to specify a list of the coefficients you use in a polynomial to characterize steady-state current information. This attribute is required in the steady_state_current_high group when you specify a scalable polynomial model. The coefficients are represented in the .lib file and saved in the database in column-first order. If any term is missing in the polynomial, you must insert a 0 (zero) in the corresponding position in the coefs attribute to ensure correct processing of the coefficients.

**Note:**

For a piecewise polynomial, define the coefs attribute inside the domain group inside the steady_state_current_high group that defines the range of coefficients.

**Syntax**

```plaintext
pin (output_pin_name) {
    timing () {
        steady_state_current_high (poly_template_string){
            coefs("float, ..., float")
        }
    }
    ...
}
```

**orders Complex Attribute**

Use the orders attribute to specify the order of the variables for the polynomial to characterize steady state current information. This attribute is required in the steady_state_current_high group when you specify a scalable polynomial model.

**Note:**

Liberty Reference Manual (Version 2007.03) 246
For a piecewise polynomial, define the `orders` attribute inside the `domain` group inside the `steady_state_current_high` group.

**Syntax**

```plaintext
pin (output_pin_name) {
    timing () {
        steady_state_current_high (poly_template_name) {
            orders("integer, ..., integer")
            }
        }
    ...
}
```

**Example**

```plaintext
pin(Z) {
    timing () {
        orders("2, 1, 1");
        coefs("1.0, 2.0, 3.0, 4.0. 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0");
        variable_n_range
        }
    ...
}
```

**Syntax**

```plaintext
pin (output_pin_name) {
    timing () {
        steady_state_current_high (lut_table_template) {
            orders("integer, ..., integer")
            coefs("float, ..., float")
            values (integer, integer);
            }
        }
    ...
}
```

**Example**

```plaintext
pin(Z) {
    timing () {
        steady_state_current_high () {
            values("2, 1.8, ...-0.8");
        }
    ...
}
```

domain Group

For information about the `domain` group syntax and usage, see the description in the "domain Group".

steady_state_current_low Group

This optional group defines the current-voltage (I-V) characteristic for a cell timing arc by holding the output signal low.

Complex Attribute

```plaintext
coefs
orders
values
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.

Group

domain

For information about the `steady_state_current_tristate` group syntax and attributes, see "steady_state_current_high Group".

steady_state_current_tristate Group

This optional group defines the current-voltage (I-V) characteristic for tri-state timing arcs.

Complex Attributes

coeffs  
orders  
values

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.

Group

domain

For information about the `steady_state_current_tristate` group syntax and attributes, see "steady_state_current_high Group".

pin_based_variation Group

The `pin_based_variation` group is similar to the `timing_based_variation` group in that it specifies the rising and falling output transitions for variation parameters (by the `va_compact_ccs_rise` and `va_compact_ccs_fall` groups) and specifies variation-aware receiver capacitance information. If a receiver capacitance group exists in a pin group, the variation-aware CCS receiver model groups, such as the following, are required in the `pin_based_variation` group.

- `va_receiver_capacitance1_rise`
- `va_receiver_capacitance1_fall`
- `va_receiver_capacitance2_rise`
- `va_receiver_capacitance2_fall`

See Also

timing_based_variation Group

Syntax

```
pin(pin_name)
...
pin_based_variation(){
  va_parameters (<string>, ...);
  nominal_va_values (<float>, ...);
  va_receiver_capacitance1_rise (template_name) {
    va_values (<float>, ...);
  }
```
values ("<float>, ..", ..);
...
va_receiver_capacitance2_rise (template_name) {
    ...
}
va_receiver_capacitance2_fall (template_name) {
    ...
}
va_receiver_capacitance1_fall (template_name) {
    ...
}
va_receiver_capacitance1_rise (template_name) {
    ...
}

Example

pin_based_variation(){
va_parameters (channel_length, threshold_voltage) ;
nominal_va_values (0.5, 0.5) ;
va_receiver_capacitance1_rise (LUT3) {
    va_values (0.50, 0.45) ;
    values ("0.29, 0.30, 0.31") ;
}
...
va_receiver_capacitance2_rise (LUT3) {
    va_values (0.50, 0.45) ;
    ...
}
va_receiver_capacitance1_fall (LUT3) {
    va_values (0.50, 0.45) ;
    ...
}
va_receiver_capacitance2_fall (LUT3) {
    va_values (0.50, 0.45) ;
    ...
}
}

Complex Attributes

va_parameters
nominal_va_values

For information about the va_parameters and nominal_va_values attributes, see "va_parameters Complex Attribute" and "nominal_va_values Complex Attribute".

Groups

va_receiver_capacitance1_rise
va_receiver_capacitance1_fall
va_receiver_capacitance2_rise
va_receiver_capacitance2_fall
va_compact_ccs_rise
va_compact_ccs_fall

For information about the va_compact_ccs_rise and va_compact_ccs_fall groups, see "va_compact_ccs_rise and va_compact_ccs_fall Groups".

Variation-Aware CCS Receiver Model Groups

The following variation-aware CCS receiver model groups specify characterization corners with variation values in timing_based_variation and pin_based_variation groups:

- va_receiver_capacitance1_rise
- va_receiver_capacitance1_fall
- va_receiver_capacitance2_rise
- va_receiver_capacitance2_fall

Syntax

va_receiver_capacitance1_rise (template_name) {
    va_values (<float>, ..) ;
    values ("<float>, ..", ..) ;
    ...
}
va_receiver_capacitance2_rise (template_name) {
    ...
}
va_receiver_capacitance1_fall (template_name) {
    ...
}
va_receiver_capacitance2_fall (template_name) {
    ...
}
Example

```plaintext
pin_based_variation()
  va_parameters (channel_length, threshold_voltage);
  nominal_va_values (0.5, 0.5);
  va_receiver capacitance1_rise (LUT3) {
    va_values (0.50, 0.45);
    values ("0.29, 0.30, 0.31");
  }
  ...
  va_receiver capacitance2_rise (LUT3) {
    va_values (0.50, 0.45);
    ...
  }
  va_receiver capacitance1_fall (LUT3) {
    va_values (0.50, 0.45);
    ...
  }
  va_receiver capacitance2_fall (LUT3) {
    va_values (0.50, 0.45);
    ...
  }
}
```

Complex Attributes

- `va_values`
- `values`

For information about the `va_values` and `values` attributes, see "va_values Complex Attribute" and "values Complex Attribute".

timing_based_variation Group

The `timing_based_variation` group is similar to the `pin_based_variation` group in that it specifies variation-aware receiver capacitance information (but in a timing group rather than in a pin group), and it specifies the rising and falling output transitions for variation parameters. The rising and falling output transitions are specified in the `va_compact_ccs_rise` and `va_compact_ccs_fall` groups, respectively.

The following information applies to the `timing_based_variation` group:

- The `va_compact_ccs_rise` group is required only if a `compact_ccs_rise` group exists within a timing group.
- The `va_compact_ccs_fall` group is required only if a `compact_ccs_fall` group exists within a timing group.

See Also

- `pin_based_variation Group`

Syntax

```plaintext
timing()
  ...
  timing_based_variation(){
    va_parameters (<string>, ...);
    nominal_va_values (<float>, ...);
    va_compact_ccs_rise (template_name) {
      va_values (<float>, ...);
      values (<float>, ..., ...);
    }
    ...
  }
```

Example

```plaintext
timing_based_variation()
  va_parameters (channel_length, threshold_voltage);
  nominal_va_values (0.50, 0.50);
  va_compact_ccs_rise (LUT4x4) {
    va_values (0.50, 0.45);
    values ("0.1, 0.5, 0.6, 0.8, 1, 3", \
            "0.15, 0.55, 0.65, 0.85, 2, 4");
    ...
  }
```

Liberty Reference Manual (Version 2007.03) 250
Complex Attributes

va_parameters
  nominal_va_values

Groups

  va_compact_ccs_rise
  va_compact_ccs_fall
  va_receiver_capacitance1_rise
  va_receiver_capacitance1_fall
  va_receiver_capacitance2_rise
  va_receiver_capacitance2_fall
  va_rise_constraint
  va_fall_constraint

For information about the variation-aware CCS receiver model groups (such as va_receiver_capacitance1_rise), see "Variation-Aware CCS Receiver Model Groups".

va_parameters Complex Attribute

The va_parameters attribute specifies a list of variation parameters within the timing_based_variation or pin_based_variation groups. The following information applies to the va_parameters attribute:

- One or more variation parameters is allowed.
- The variation parameters are represented by a string.
- All va_parameters values must be unique.
- The va_parameters attribute must be defined before it is referenced by nominal_va_values and va_values.

The va_parameters attribute can be specified within a variation group or at the library level. (The variation groups include timing_based_variation and pin_based_variation.)

- If va_parameters is specified at the library level, all cells under the library default to the same variation parameters.
- If va_parameters is defined in a variation group, all va_values and nominal_va_values attribute values under the same variation group refer to va_parameters.

The va_parameters values can be parameters that are user-defined or predefined. The parameters defined in default_operating_conditions are process, temperature, and voltage.

The voltage names are defined using the voltage_map complex attribute. For more information about the voltage_map attribute, see "voltage_map Complex Attribute".

Syntax

  va_parameters (<string>), ... );

Example

  timing_based_variation()
    va_parameters (channel_length, threshold_voltage);

nominal_va_values Complex Attribute

The nominal_va_values attribute characterizes nominal values for all variation parameters. The following information applies to the nominal_va_values attribute.

- The attribute is required for every timing_based_variation group.
- The nominal_va_values attribute values map one-to-one to the corresponding va_parameters values.
- If the nominal compact CCS driver and the variation-aware compact CCS driver
model groups are defined under the same timing group, the
nominal_va_values values are applied to the nominal compact CCS driver
and the variation-aware compact CCS driver groups.

**Syntax**

```plaintext
nominal_va_values (<float>, ...);
```

**Example**

```plaintext
timing_based_variation()
  va_parameters (channel_length, threshold_voltage)
  nominal_va_values (0.50, 0.50);
```

**va_compact_ccs_rise and va_compact_ccs_fall Groups**

The `va_compact_ccs_rise` and `va_compact_ccs_fall` groups specify
characterization corners with variation parameter values. The following information applies
to the `va_compact_ccs_rise` and `va_compact_ccs_fall` groups.

- The groups can be specified under different `timing_based_variation`
groups if they cannot share the same `va_parameters`
- The `template_name` value refers to the `compact_lut_template` group.
- You must characterize two corners at each side of the nominal value for all
variation parameters specified in `va_parameters`. When corners are
characterized for one of the parameters, all other variations are assumed to be
nominal values. Therefore, for a `timing_based_variation` group with `n`
variation parameters exactly `2^n` characterization corners are required.

**Syntax**

```plaintext
va_compact_ccs_rise (template_name) {
  va_compact_ccs_fall (template_name) {
```

**Example**

```plaintext
timing_based_variation()
  va_parameters (channel_length, threshold_voltage);
  nominal_va_values (0.50, 0.50);
  va_compact_ccs_rise (LUT4x4) {
    va_values (0.50, 0.45);
    values ("0.1, 0.5, 0.6, 0.8, 1, 3",
           "0.15, 0.55, 0.65, 0.85, 2, 4",
           ...);
  }
  ...}
  va_compact_ccs_fall (LUT4x4) {
    values ("-0.1, -0.5, 0.6, 0.8, 1, 3",
           "-0.15, -0.55, 0.65, 0.85, 2, 4",
           ...);
```

**Complex Attributes**

```plaintext
va_values
```

**va_values Complex Attribute**

The `va_values` attribute defines the values of each variation parameter for all corners
characterized in the variation-aware compact CCS driver and receiver model groups, such
as the following groups:

- `va_compact_ccs_rise`
- `va_compact_ccs_fall`
- `va_receiver_capacitance1_rise`
- `va_receiver_capacitance1_fall`
- `va_receiver_capacitance2_rise`
- `va_receiver_capacitance2_fall`

**Syntax**

```plaintext
```
va_values (<float>, ...);

Example

va_compact_ccs_rise (LUT4x4) {
va_values (0.50, 0.45);
}

values Complex Attribute

The `values` attribute follows the same rules as the nominal compact CCS driver model groups (such as `compact_ccs_rise` and `compact_ccs_fall`) with the following exceptions:

- `left_id` and `right_id` are optional.
- The `left_id` and `right_id` values must be represented in a pair; they can either be omitted or included in the `compact_lut_template`.
- If `left_id` and `right_id` are not defined in the variation-aware compact CCS driver groups, the values default to the values defined in the nominal compact CCS driver model groups. For more information, see "compact_ccs_rise and compact_ccs_fall Groups".

Syntax

values ("..., <float>, ..., <integer>, ...");

Example

va_compact_ccs_rise (LUT4x4) {
va_values (0.50, 0.45);
values ("0.1, 0.5, 0.6, 0.8, 1, 3", 
"0.15, 0.55, 0.65, 0.85, 2, 4", 
"...");
}

va_rise_constraint and va_fall_constraint Groups

The `va_rise_constraint` and `va_fall_constraint` groups specify characterization corners with variation values in the `timing_based_variation` group. The attributes under these groups undergo screening and checking in the same way as the nominal timing constraint models. The following information applies to the `va_rise_constraint` and `va_fall_constraint` groups.

- All variation-aware constraint groups in the `timing_based_variation` group share the same parameters.
- Both groups can be specified under different `timing_based_variation` groups if they cannot share the same `va_parameters`.
- The `template_name` value refers to the `lu_table_template` group.

Syntax

va_rise_constraint (template_name) {
va_values (<float>, ...);
values (<float>, ...);
...
va_fall_constraint (template_name) {
...
}

Example

va_rise_constraint (LUT5x5) {
va_values (0.50, 0.45);
values ("-0.1452, -0.1452, -0.1452, -0.1452,
..." undertake the same format and structure as the `va_compact_ccs_rise` example);
}
...
va_fall_constraint (LUT5x5) {
va_values (0.55, 0.50);
...
}
Complex Attribute

va_values

For information about the va_values attribute, see "va_values Complex Attribute".

3.2.16 t latch Group

In timing analysis, use a t latch group to describe the relationship between the data pin and the enable pin on a transparent level-sensitive latch.

You define the t latch 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".

Syntax

library (namestring) {
    cell (namestring) {
        ...
        timing_model_type : valueenum :
        ...
        pin (data_pin_namestring) {
            t latch (enable_pin_namestring) {
                tlatch description ...
            }
        }
    }
}

Simple Attributes

date_type
disable

date_type Simple Attribute

Use the date_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

date_type : rising;

disable Simple Attribute

Use the disable attribute to disable transparency in a latch.

Syntax

disable : valueBoolean

value

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

Example

disable : FALSE;
Index


A

area_coefficient attribute  3.2.4

area attribute
  in wire_load group  1.9.45

array value, of base_type  1.9.43

attributes, technology library
  clear_preset_var1  2.1.4
  clear_preset_var2  2.1.4

attributes
  See attributes, default values
    technology library
      output_threshold_pct_fall  1.6.19

attributes, default values
  pin group  1.7  1.7  1.7
  timing group  1.7  1.7  1.7

attributes, related_outputs  2.1.4

attributes, technology library
  area  1.9.45
    area_coefficient  3.2.4
  auxiliary_pad_cell  2.1.2
  base_name  2.1.2
  base_type  2.1.4
  bit_from  1.9.43  2.1.4
  bit_to  1.9.43  2.1.4
  bit_width  1.9.43  2.1.4  3.1.2
  bus_naming_style  1.6.1  2.1.2
  bus_type  2.1.4
  calc_mode  1.9.19  1.9.24  1.9.29  1.9.31
  capacitance  1.9.45  1.9.45  2.1.4  2.1.4  3.1.2
  capacitive_load_unit  1.8.1
  cell_footprint  2.1.2
  cell_leakage_power  2.1.2
  cell_name  2.2.1
    clear  2.1.4  2.1.4  2.1.4  2.1.4
    clear_preset_var1  2.1.4
  clock  3.1.2
    clock_gate_clock_pin  3.1.2
    clock_gate_enable_pin  3.1.2
    clock_gate_obs_pin  3.1.2
    clock_gate_out_pin  3.1.2
    clock_gate_test_pin  3.1.2
    clock_gating_integrated_cell  2.1.2
  clock_pin  2.1.4
    clocked_on  2.1.4  2.1.4
    clocked_on_also  2.1.4  2.1.4
    coefs  3.2.13  3.2.13  3.2.15  3.2.15  3.2.15
    constraint  3.2.12
    constraint_high  3.2.11
    constraint_low  3.2.11
    contention_condition  2.1.2
  complement_pin  3.1.2
  connection_class  3.1.2
  constraint  3.2.12
  constraint_high  3.2.11
  constraint_low  3.2.11
  contention_condition  2.1.2
<table>
<thead>
<tr>
<th>Term</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>pulling_resistance_unit</td>
<td>1.6.26</td>
</tr>
<tr>
<td>pulse_clock</td>
<td>3.1.2</td>
</tr>
<tr>
<td>rail_connection</td>
<td>2.1.3</td>
</tr>
<tr>
<td>reference_time</td>
<td>2.1.4 2.1.4 3.2.15</td>
</tr>
<tr>
<td>related_bus_pins</td>
<td>3.2.3</td>
</tr>
<tr>
<td>related_ground_pin</td>
<td>3.1.2</td>
</tr>
<tr>
<td>related_inputs</td>
<td>2.1.4</td>
</tr>
<tr>
<td>related_outputs</td>
<td>2.1.4</td>
</tr>
<tr>
<td>related_pg_pin</td>
<td>2.1.4</td>
</tr>
<tr>
<td>related_pin</td>
<td>3.2.3</td>
</tr>
<tr>
<td>related_power_pin</td>
<td>3.1.2</td>
</tr>
<tr>
<td>resistance</td>
<td>1.9.45 1.9.45</td>
</tr>
<tr>
<td>resource_usage</td>
<td>2.1.3</td>
</tr>
<tr>
<td>revision</td>
<td>1.6.27</td>
</tr>
<tr>
<td>rise_capacitance</td>
<td>3.1.2</td>
</tr>
<tr>
<td>rise_capacitance_range</td>
<td>3.1.3</td>
</tr>
<tr>
<td>rise_current_slope_after_threshold</td>
<td>3.1.2</td>
</tr>
<tr>
<td>rise_current_slope_before_threshold</td>
<td>3.1.2</td>
</tr>
<tr>
<td>rise_resistance</td>
<td>3.2.15</td>
</tr>
<tr>
<td>rise_time_after_threshold</td>
<td>3.1.2</td>
</tr>
<tr>
<td>rise_time_before_threshold</td>
<td>3.1.2</td>
</tr>
<tr>
<td>routing_layers</td>
<td>1.8.8</td>
</tr>
<tr>
<td>scaling_factors</td>
<td>2.1.2</td>
</tr>
<tr>
<td>sdf_cond</td>
<td>2.1.4 3.2.11 3.2.12 3.2.12 3.2.15</td>
</tr>
<tr>
<td>sdf_cond_end</td>
<td>3.2.15</td>
</tr>
<tr>
<td>sdf_cond_start</td>
<td>3.2.15</td>
</tr>
<tr>
<td>shifts</td>
<td>2.1.4</td>
</tr>
<tr>
<td>short (model group)</td>
<td>2.2.1</td>
</tr>
<tr>
<td>signal_type</td>
<td>2.1.4 3.1.2</td>
</tr>
<tr>
<td>simulation</td>
<td>1.6.28</td>
</tr>
<tr>
<td>single_bit_degenerate_attribute</td>
<td>2.1.2</td>
</tr>
<tr>
<td>slew</td>
<td>1.9.17</td>
</tr>
<tr>
<td>slew_derate_from_library</td>
<td>1.6.29</td>
</tr>
<tr>
<td>slew_lower_threshold_pct_fall</td>
<td>1.6.30 3.1.2</td>
</tr>
<tr>
<td>slew_lower_threshold_pct_rise</td>
<td>1.6.31 3.1.2</td>
</tr>
<tr>
<td>slew_type</td>
<td>2.1.2</td>
</tr>
<tr>
<td>slew_upper_threshold_pct_fall</td>
<td>1.6.32 3.1.2</td>
</tr>
<tr>
<td>slew_upper_threshold_pct_rise</td>
<td>1.6.33 3.1.2</td>
</tr>
<tr>
<td>slope_fall</td>
<td>3.2.15</td>
</tr>
<tr>
<td>slope_rise</td>
<td>3.2.15</td>
</tr>
<tr>
<td>slowest_factor</td>
<td>1.9.42</td>
</tr>
<tr>
<td>stage_type</td>
<td>3.2.1 3.2.1</td>
</tr>
<tr>
<td>state_function</td>
<td>3.1.2</td>
</tr>
<tr>
<td>steady_state_resistance_above_high</td>
<td>3.2.15</td>
</tr>
<tr>
<td>steady_state_resistance_below_low</td>
<td>3.2.15</td>
</tr>
<tr>
<td>steady_state_resistance_high</td>
<td>3.2.15</td>
</tr>
<tr>
<td>steady_state_resistance_low</td>
<td>3.2.15</td>
</tr>
<tr>
<td>step_level</td>
<td>1.9.27</td>
</tr>
<tr>
<td>switching_interval</td>
<td>3.2.8</td>
</tr>
<tr>
<td>switching_together</td>
<td>3.2.8</td>
</tr>
<tr>
<td>tdisable</td>
<td>3.2.16</td>
</tr>
<tr>
<td>technology</td>
<td>1.8.9</td>
</tr>
<tr>
<td>temperature</td>
<td>1.9.24</td>
</tr>
<tr>
<td>test_output_only</td>
<td>3.1.2</td>
</tr>
<tr>
<td>three_state</td>
<td>3.1.2</td>
</tr>
<tr>
<td>threshold_voltage_group</td>
<td>2.1.2</td>
</tr>
<tr>
<td>tied_off</td>
<td>3.2.15</td>
</tr>
<tr>
<td>time_unit</td>
<td>1.6.34</td>
</tr>
<tr>
<td>timing_model_type</td>
<td>2.1.2</td>
</tr>
<tr>
<td>timing_sense</td>
<td>3.2.15</td>
</tr>
<tr>
<td>timing_type</td>
<td>3.2.15</td>
</tr>
<tr>
<td>total_track_area</td>
<td>2.1.4</td>
</tr>
<tr>
<td>tracks</td>
<td>2.1.4</td>
</tr>
<tr>
<td>tree_type</td>
<td>1.9.24</td>
</tr>
<tr>
<td>typical_capacitances</td>
<td>2.1.4</td>
</tr>
<tr>
<td>use_for_size_only</td>
<td>2.1.2</td>
</tr>
<tr>
<td>valid_speed_grade</td>
<td>1.9.27</td>
</tr>
<tr>
<td>valid_step_levels</td>
<td>1.9.27</td>
</tr>
<tr>
<td>value</td>
<td>2.1.4 2.1.4 2.1.4</td>
</tr>
<tr>
<td>variable_n_range</td>
<td>1.9.29 1.9.29 3.2.13 3.2.13</td>
</tr>
<tr>
<td>variables</td>
<td>1.9.29 1.9.29 1.9.31 1.9.31</td>
</tr>
</tbody>
</table>
vhdl_name 2.1.2 3.1.2
voltage 1.9.24
voltage_map 1.8.10
voltage_name 2.1.4
voltage_unit 1.6.35
when 2.1.4 3.2.3 3.2.11 3.2.12 3.2.15
   in dynamic_current group 2.1.4
   in intrinsic_parasitic group 2.1.4
   in leakage_current group 2.1.4
when_end 3.2.15 3.2.15
width_coefficient 3.2.4
x_function 3.1.2

attributes, user-defined 1.8.3
auxiliary_pad_cell attribute 2.1.2

B
backslash, as escape character 2.1.4 3.1.2
base_name attribute 2.1.2
base_type attribute 1.9.43 2.1.4
bit_from attribute 1.9.43 2.1.4
bit_to attribute 1.9.43 2.1.4
bit_width attribute 1.9.43
   in pin group 3.1.2
   in type group 2.1.4
bit width of multibit cell 2.1.4

Boolean operators, valid 2.1.4 3.1.2 3.2.8
bundle group
   bundle members 2.1.4
   bundle names 2.1.4
capacitance attribute 2.1.4
direction attribute 2.1.4
function attribute 2.1.4
members attribute 2.1.4 2.1.4
pin attributes 2.1.4
pin group in 2.1.4
pin names 2.1.4
bus_hold pin 3.1.2
bus_naming_style attribute 1.6.1 2.1.2
   characters in 1.6.1
   symbols in 1.6.1
bus_type attribute 2.1.4
bus, reversing order 3.1.2
bus group
   bus_type attribute 2.1.4
   bus pin 2.1.4 2.1.4
capacitance attribute 2.1.4 2.1.4
direction attribute 2.1.4
   in multibit flip-flop registers 2.1.4
   in multibit latch registers 2.1.4
   pin attributes 2.1.4
   pin group 2.1.4
type group, use in 1.9.43
bus pin
in bundle group
  example 2.1.4
  specifying default attributes 2.1.4
  overriding default attributes 2.1.4
  specifying default attributes 2.1.4

bus pin group
  bus members in flip-flop bank 2.1.4
  example 2.1.4
  naming convention 2.1.4

C

calc_mode attribute
  in lu_table_template group 1.9.19
  in operating_conditions group 1.9.24
  in poly_template group 1.9.29
  in power_poly_template group 1.9.31

capacitance
  in pin group in bundle group 2.1.4
  load units 1.8.1
  wire length 1.8.7

capacitance attribute
  in bundle group 2.1.4
  in bus group 2.1.4
  in pin group 3.1.2
  in wire_load group 1.9.45 1.9.45

  capacitance group 3.2.13
  capacitive_load_unit attribute 1.8.1

ccsn_first_stage group 3.2.1
ccsn_last_stage group 3.2.2

cell_degradation group 3.2.15

cell_rise group 3.2.15

cell_footprint attribute 2.1.2

cell_leakage_power attribute 2.1.2

cell_name attribute 2.2.1

cell_rise group 3.2.15

cell_delay
  cell_rise group 3.2.15
  cell_rise group 3.2.15

cell group
  clock_gating_integrated_cell attribute 2.1.2
  example, CMOS 2.1.4
  ff_bank group 2.1.4
  ff group 2.1.4
  group statements
    bundle 2.1.4
    latch 2.1.4
    latch_bank 2.1.4
    leakage_current 2.1.4
    leakage_power 2.1.4
    lut 2.1.4
    pin 3.1
    routing_track 2.1.4 2.1.4
    statetable 2.1.4
    test_cell 2.1.4
    type 2.1.4
is_isolation_cell attribute 2.1.2
is_level_shifter attribute 2.1.2
lut group 2.1.4
syntax 2.1

cell swapping
  in_place_swap_mode attribute 1.6.11

clear_preset_var1 attribute
  in ff_bank group 2.1.4
  in ff group 2.1.4

clear_preset_var2 attribute
  in ff_bank group 2.1.4
  in ff group 2.1.4

clear, timing_type value 3.2.15

clear attribute
  in ff_bank group 2.1.4
  in ff group 2.1.4
  in latch_bank group 2.1.4
  in latch group 2.1.4

clock_gate_clock_pin attribute 3.1.2

clock_gate_enable_pin attribute 3.1.2

clock_gate_obs_pin attribute 3.1.2

clock_gate_out_pin attribute 3.1.2

clock_gate_test_pin attribute 3.1.2

clock_gating_flag attribute 3.2.15

clock_gatingIntegrated_cell attribute
  defined 2.1.2
  setting pin attributes 2.1.2

clock_pin attribute 2.1.4

clock attribute 3.1.2

clocked_on_also attribute 2.1.4  2.1.4

clocked_on attribute 2.1.4  2.1.4

clock pin
  active edge 2.1.4  2.1.4
  min_period attribute 3.1.2

CMOS, library group example 1.5

coeffs attribute 3.2.13  3.2.13  3.2.15  3.2.15  3.2.15
  in lower group 3.2.13

comment attribute 1.6.2

complex sequential cells 2.1.4

complimentary_pin attribute 3.1.2

composite current source
  template variables 3.2.15  3.2.15

conditional timing check
  in timing group 3.2.15
  in VITAL models 3.2.15

connection_class attribute 3.1.2

constraint_high attribute 3.2.11
constraint_low attribute  3.2.11  
constraint attribute  3.2.12  

constraint table  
   lu_table_template group 
      constrained_pin_transition  1.9.19  1.9.19  
contention_condition attribute  2.1.2  
control signals in multibit register  2.1.4  2.1.4  
copyright information  1.6.2  
current_unit attribute  1.6.3  

D  

data_in attribute  
   for latches  2.1.4  2.1.4  2.1.4  
      in latch_bank group  2.1.4  
data_type attribute  1.9.43  2.1.4  
data attribute  1.6.4  
dc_current_template group  1.9.11  
dc_current group  3.2.1  
dc_shell  
   set_dont_touch command  2.1.2  
      set_map_only command  2.1.2  
default_cell_leakage_power attribute  1.7  
default_connection_class attribute  1.7  
default_fall_delay_intercept attribute  1.7  
default_fall_pin_resistance attribute  1.7  
default_fanout_load_attribute  1.7  
default_fpga_isd attribute  1.6.5  
default_inout_pin_cap attribute  1.7  
default_inout_pin_cap attribute  1.7  
default_inout_pin_fall_res attribute  1.7  
default_inout_pin_rise_res attribute  1.7  
default_input_pin_cap attribute  1.7  
default_intrinsic_fall attribute  1.7  
default_intrinsic_rise attribute  1.7  
default_leakage_power_density attribute  1.7  
default_max_capacitance attribute  1.7  
default_max_fanout_attribute  1.7  
default_max_transition attribute  1.7  
default_max_utilization attribute  1.7  
default_min_porosity attribute  1.7  
default_operating_conditions attribute  1.7
default_output_pin_cap attribute 1.7
default_output_pin_fall_res attribute 1.7
default_output_pin_rise_res attribute 1.7
default_part attribute 1.8.2
default_rise_delay_intercept attribute 1.7
default_rise_pin_resistance attribute 1.7
default_slope_fall attribute 1.7
default_slope_rise attribute 1.7
default_step_level attribute 1.9.27
default_threshold_voltage_group attribute 1.6.6
default_timing attribute 3.2.15
default_wire_load_area attribute 1.7
default_wire_load_capacitance attribute 1.7
default_wire_load_model attribute 1.7
default_wire_load_resistance attribute 1.7
default_wire_load_selection attribute 1.7
default_wire_load attribute 1.7
default attributes overriding values 1.7
default pin attributes 1.7
define_cell_area attribute 1.8.4
define_group attribute 1.8.5
define attribute 1.8.3
delay_model attribute 1.6.7
delay models supported 1.6.7
design translation 2.1.2
D flip-flop 2.1.4

differential I/O complementary_pin attribute 3.1.2
definition 3.1.2
fault_model attribute 3.1.2
direction attribute in bundle group 2.1.4
in bus group 2.1.4
in pin group 3.1.2
in test_cell group 2.1.4
divided_by attribute 2.1.4
domain group
in cell_fall group 3.2.15
in cell_rise group 3.2.15
in fall_constraint group 3.2.15
in fall_power group 3.2.8
in fall_propagation group 3.2.15
in fall_transition group 3.2.15
in lu_table__template group 1.9.19
in noise_immunity_above_high group 3.2.15
in poly_template group 1.9.29 1.9.30
in power_poly_template group 1.9.31
in power group 3.2.8
in propagated_noise_height_above_high group 3.2.15
in retaining_fail group 3.2.15
in retaining_rise group 3.2.15
in rise_power group 3.2.8
in rise_propagation group 3.2.15
in rise_transition group 3.2.15
in steady_state_current_high group 3.2.15
in timing group 3.2.15
dont_fault attribute 2.1.2 3.1.2
dont_touch attribute 2.1.2
dont_use attribute 2.1.2
downto attribute 1.9.43 2.1.4
drive_current attribute 3.1.2
drive_type attribute 2.1.2
drive attribute 1.9.17
drive capability 3.2.15
driver_type attribute 3.1.2
driver types, multiple 3.1.2
duty_cycle attribute 2.1.4
dynamic_current group 2.1.4

E

driver_type attribute 3.2.16
dedges attribute 2.1.4
edif_name attribute 2.1.2
electromigration group 3.2.3
   when attribute 3.2.3
ealut_template group 1.9.12
ealut_template group 1.9.12
em_max_toggle_rate group 3.2.3
em_temp_degradation_factor attribute 1.6.8 2.1.2
enable attribute
   for latches 2.1.4
   in latch_bank group 2.1.4
   in latch group 2.1.4 2.1.4

F

fall_capacitance_range attribute 3.1.3
fall_capacitance_range group 3.2.13 3.2.13
fall_capacitance attribute
   in pin group 3.1.2
fall_capacitance group 3.2.13
fall_constraint group 3.2.15
fall_current_slope_after_threshold attribute 3.1.2
fall_current_slope_before_threshold attribute 3.1.2
fall_delay_intercept attribute 3.2.15
fall_net_delay group 1.9.13
fall_pin_resistance attribute 3.2.15
fall_power group 3.2.8
fall_propagation group 3.2.15
fall_resistance attribute 3.2.15
fall_time_after_threshold attribute 3.1.2
fall_time_before_threshold attribute 3.1.2
fall_transition_degradation group 1.9.14
fall_transition group 3.2.15
falling_edge, timing_type value 3.2.15
falling_together_group attribute 3.2.8
falling-edge-triggered-devices 2.1.4
fanout
  control signals in multibit register 2.1.4 2.1.4
fanout_length attribute 1.9.45
fanout_load attribute 3.1.2
faster_factor attribute 1.9.42
fault_model attribute 3.1.2
ff_bank group 2.1.4
  clear_preset_var1 attribute 2.1.4
  clear_preset_var2 attribute 2.1.4
  clear attribute 2.1.4
  data_in attribute 2.1.4
  in test cell group 2.1.4
  next_state attribute 2.1.4
  preset attribute 2.1.4
ff group
  clear_preset_var1 attribute 2.1.4
  clear_preset_var2 attribute 2.1.4
  in test_cell_group 2.1.4
  master-slave flip-flop 2.1.4
  next_state attribute 2.1.4
  preset attribute 2.1.4
  single-stage D flip-flop 2.1.4 2.1.4
file size, reducing 1.3
flip-flop
  D 2.1.4 2.1.4
  JK 2.1.4
  JK with scan 2.1.4 2.1.4
  master-slave 2.1.4
  single-stage 2.1.4
footprint class 2.1.2
fpga_arc_condition attribute 2.1.4 3.2.15
fpga_cell_type attribute 2.1.2

fpga_condition_value group 2.1.4

fpga_condition group 2.1.4

fpga_domain_style attribute
    in cell group 2.1.2
    in library group 1.6.9
    in pin group 3.2.15

fpga_isd attribute
    in cell group 2.1.2
    in part group 1.9.27
    in speed_grade group 1.9.27

fpga_isd group 1.9.17

fpga_technology attribute 1.6.10

function attribute
    bused pin names 2.1.4
    in bundle group 2.1.4
    in pin group 3.1.2
    of bus pins 2.1.4 3.1.2 3.1.2
    pin names as arguments 2.1.4

G

gate mapping 2.1.2

generated_clock Group 2.1.4

geometry_print attribute 2.1.2

group statements
    cell group
    dc_current 3.2.1
domain 1.9.31
dynamic_current 2.1.4
electromigration 3.2.3
intrinsic_capacitance 2.1.4
intrinsic_parasitic 2.1.4
intrinsic_resistance 2.1.4
lu_table_template 1.9.19
out_put_rise 3.2.1
output_fall 3.2.1
propogated_noise_high 3.2.1
propogated_noise_low 3.2.1
switching_group 2.1.4 2.1.4
syntax
    capacitance group 3.2.13
cell_degradation 3.2.15
cell_fall 3.2.15
cell_rise 3.2.15
cell group
dc_current_template 1.9.11
domain 1.9.19 1.9.29 1.9.30
fall_capacitance 3.2.13
fall_capacitance_range 3.2.13
fall_net_delay 1.9.13
fall_transition_degradation 1.9.14
fpga_condition 2.1.4
fpga_condition_value 2.1.4
fpga_isd 1.9.17
hyperbolic_noise_above_high 3.2.4
hyperbolic_noise_below_low 3.2.5
hyperbolic_noise_high 3.2.6
hyperbolic_noise_low 3.2.7
iv_lut_template 1.9.18
handle_negative_constraint attribute 2.1.2
has_builtin_pad attribute 3.1.2
height_coefficient attribute 3.2.4
high-active clock signal 2.1.4
high-impedance state 3.1.2
hold_falling, timing_type value 3.2.15
hold_rising, timing_type value 3.2.15
hyperbolic_noise_above_high group 3.2.4
hyperbolic_noise_below_low group 3.2.5
hyperbolic_noise_high group 3.2.6
hyperbolic_noise_low group 3.2.7
hysteresis attribute 3.1.2

in_place_swap_mode attribute 1.6.11
in_place_swap_mode attribute 2.1.4
report_lib, use of 1.6.11
include_file attribute 1.3
index_1 attribute
  em_lut_template group 1.9.12
  in_table_template group 1.9.18
  power_lut_template group 1.9.30
index_2 attribute
  em_lut_template group 1.9.12
  in_table_template group 1.9.18
  power_lut_template group 1.9.30
index_3 attribute
  in_table_template group 1.9.18
  power_lut_template group 1.9.30
index_output attribute 2.1.4 2.1.4
in-place optimization 1.6.11
input_map attribute 3.1.2
input_signal_level attribute 3.1.2
input_switching_condition attribute 2.1.4
input_threshold_pct_fall attribute 1.6.12 3.1.2
input_threshold_pct_rise attribute 1.6.13 3.1.2
input_voltage group 1.9.16
  variables 1.9.16
interdependence_id attribute
  in_pin group 3.2.15
interface_timing attribute 2.1.2
internal_node attribute 3.1.2
internal_power group 3.2.8
  equal_or_opposite_output attribute 3.2.8
fall_power group 3.2.8
falling_together_group attribute 3.2.8
one-dimensional table 3.2.8
power_level attribute 3.2.8
power group 3.2.8
related_pin attribute 3.2.8
rise_power attribute 3.2.8
rising_together_group attribute 3.2.8
switching_interval attribute 3.2.8
switching_together_group attribute 3.2.8
three-dimensional table 3.2.8
two-dimensional table 3.2.8
when attribute 3.2.8

intrinsic_capacitance group 2.1.4
intrinsic_parasitic group 2.1.4
intrinsic_resistance group 2.1.4
intrinsic_rise attribute 3.2.15
invert attribute 2.1.4
inverted_output attribute 3.1.2
io_type attribute 1.9.17 2.1.2
is_inverting attribute 3.2.1
is_isolation_cell attribute 2.1.2
is_level_shifter attribute 2.1.2
is_needed attribute 3.2.1
is_pad attribute 3.1.2
isolation_cell_enable_pin attribute 3.1.2
iv_lut_template group 1.9.18

J
JK flip-flop 2.1.4 2.1.4 2.1.4

L
latch_bank group 2.1.4
enable attribute 2.1.4
in_test_cell group 2.1.4
preset attribute 2.1.4
latch group
in_cell group 2.1.4
in_test_cell group 2.1.4
leakage_current group 2.1.4
leakage_power_unit attribute 1.6.14
leakage_power group 2.1.4
leakage_power, defining cell 2.1.2
level_shifter_enable_pin attribute 3.1.2
level-sensitive memory devices 2.1.4
libraries, power units in 1.6.14
library_features attribute 1.8.6
library group, technology library examples, CMOS 1.5
  naming 1.4
  syntax 1.2
library groups, technology library
  em_lut_template 1.9.12
  fall_net_delay 1.9.13
  fall_transition_degradation 1.9.14
  input_voltage 1.9.16
  lu_table_template 1.9.19
  operating_conditions 1.9.24
  output_current_template 1.9.25
  output_voltage 1.9.26
  part 1.9.27
  power_lut_template 1.9.30
  power_supply_group 1.9.32
  rise_net_delay 1.9.34
  rise_transition_degradation 1.9.35
  scaled_cell 1.9.36
  scaling_factors 1.9.40
  timing 1.9.41
  timing_range 1.9.42
  type 1.9.43
  user_parameters 1.9.44
  wire_load 1.9.45
  wire_load_selection 1.9.46
  wire_load_table 1.9.47
low-active
  clear_signal 2.1.4
  clock_signal 2.1.4
lower group 3.2.13
LSSD methodology
  pin_identification 2.1.4 3.1.2
lu_table_template_group 1.9.19
lut_group 2.1.4

M
map_only_attribute 2.1.2
map_to_logic_attribute 3.1.2
mapping_attribute 1.9.29 1.9.31
master_pin_attribute 2.1.4
master-slave
  clocks 2.1.4 2.1.4
  flip-flop 2.1.4
max_capacitance_attribute 3.1.2
max_cap_group 3.2.9
max_clock_tree_path, timing_type_value 3.2.15
max_count_attribute 1.9.27
max_fanout_attribute 3.1.2
max_input_noise_attribute 3.1.2
max_trans group 3.2.10
max_transition attribute 3.1.2
maxcap_lut_template group 1.9.20
maxtrans_lut_template group 1.9.21
member pins 2.1.4
members attribute 2.1.4
in bundle group 2.1.4
miller_cap_fall attribute 3.2.1
miller_cap_rise attribute 3.2.1
min_capacitance attribute 3.1.2
min_clock_tree_path, timing_type value 3.2.15
min_fanout attribute 3.1.2
min_input_noise attribute 3.1.2
min_period attribute 3.1.2
min_pulse_width_high attribute 3.1.2
min_pulse_width_low attribute 3.1.2
min_pulse_width group 3.2.11
constraint_high attribute 3.2.11
constraint_low attribute 3.2.11
sdf_cond attribute 3.2.11
when attribute 3.2.11
minimum_period, timing_type value 3.2.15
minimum_period group
constraint attribute 3.2.12
in pin group 3.2.12
sdf_cond attribute 3.2.12
when attribute 3.2.12
minimum_pulse_width, timing_type value 3.2.15
mode_definition group
syntax 2.1.4
mode attribute 3.2.15
model group
syntax 2.2
modeling
capacitance
total pin 1.6.21
total pin and wire 1.6.21
wire 1.6.21
multibit registers
flip-flop 2.1.4
multicell_pad_pin attribute 3.1.2
multiple paths 3.2.15
multiple power supply report
example 1.9.32
multiplied_by attribute 2.1.4
name mapping
  report_lib -vhdl_name 2.1.2

naming a technology library group 1.4

n-channel open drain 3.1.2

negative_unate value, of timing_sense 3.2.15

negative-edge-triggered devices, see falling-edge-triggered devices 2.1.4

negative timing constraints 2.1.2

next_state attribute
  in ff_bank group 2.1.4 2.1.4
  in ff group 2.1.4

nextstate_type attribute 3.1.2

noise_immunity_above_high group 3.2.15

noise_immunity_below_low group 3.2.15

noise_immunity_high group 3.2.15

noise_immunity_low group 3.2.15

noise_lut_template group 1.9.22

nom_calc_mode attribute 1.6.15

nom_temperature attribute 1.6.17

nom_voltage attribute 1.6.18

non_unate value, of timing_sense 3.2.15

num_blockrams attribute 1.9.27

num_cols attribute 1.9.27

num_ffs attribute 1.9.27

num_luts attribute 1.9.27

num_rows attribute 1.9.27

open_source pin 3.1.2

operating_conditions group 1.9.24
  effect on input voltage groups 1.9.16
  effect on output voltage groups 1.9.26

operators
  precedence of 2.1.4 3.1.2 3.2.8

operators, valid 2.1.4

orders attribute 1.9.29 3.2.13 3.2.13 3.2.15 3.2.15 3.2.15 3.2.15 3.2.15
  in lower group 3.2.13

out_put_rise group 3.2.1

output_current_fall group 3.2.15

output_current_rise group 3.2.15
output_current_template group 1.9.25
output_fall group 3.2.1
output_signal_level attribute 3.1.2
output_switching_condition attribute 2.1.4
output_threshold_pct_fall attribute 1.6.19
output_threshold_pct_rise attribute 1.6.20
output_voltage, variables 1.9.26
output_voltage group 1.9.26

P

pad_cell attribute 2.1.2
pad_driver_sites 1.8.4
pad_input_driver_sites 1.8.4
pad_output_driver_sites 1.8.4
pad_slots 1.8.4
pad_type attribute 2.1.2

pad cells
  pad_cell attribute 2.1.2
  pad_type attribute 2.1.2
  power_cell_type attribute 2.1.2

pads
  input voltage levels 1.9.16
  output voltage levels 1.9.26 1.9.26
  slew-rate control 3.1.2

pad slots, defining number of 1.8.4
parallel single-bit sequential cells 2.1.4

part group 1.9.27
path tracing
  defining multiple paths 3.2.15
p-channel open drain 3.1.2
pg_current_template group 1.9.28
pg_pin group 2.1.4
pg_type attribute 2.1.4
physical time unit in library 1.6.34

piece_define attribute 1.8.7
piece_type attribute 1.6.21
pin_capacitance group 3.2.13
pin_count attribute 1.9.27
pin_equal attribute 2.1.3
pin_func_type attribute 3.1.2
pin_opposite attribute 2.1.3
pin default attributes

pin group
  attribute defaults
  bit_width attribute
  capacitance attribute
  cell_degradation group
  clock_gate_clock_pin attribute
  clock_gate_enable_pin attribute
  clock_gate_obs_pin attribute
  clock_gate_out_pin attribute
  clock_gate_test_pin attribute
  clock attribute
  coefs attribute
  complementary_pin attribute
  connection_class attribute
  defaults
  direction attribute
  dont_fault attribute
  drive_current attribute
  driver_type attribute
  examples, CMOS
  fall capacitance attribute
  fall_current_slope_after_threshold attribute
  fall_current_slope_before_threshold attribute
  fall_time_after_threshold attribute
  fall_time_before_threshold attribute
  fanout_load attribute
  fault_model attribute
  function attribute
  hysteresis attribute
  in bundle group
  in bus group
  input_map attribute
  input_signal_level attribute
  input_voltage attribute
  internal_node attribute
  internal_power group
    equal_or_opposite_output attribute
    fan_power group
    falling_together_group attribute
    power_level attribute
    power group
    related_pin attribute
    rise_power attribute
    rising_together_group attribute
    switching_interval attribute
    switching_together_group attribute
    when attribute
  in test_cell group
  direction attribute
  inverted_output attribute
  is_pad attribute
  isolation_cell_enable_pin attribute
  level_shifter_enable_pin attribute
  map to logic attribute
  max capacitance attribute
  max_fanout attribute
  max_transition attribute
  min capacitance attribute
  min_fanout attribute
  min_period attribute
  min_pulse_width_high attribute
  min_pulse_width_low attribute
  min_pulse_width group
  minimum_period group
  multicell_pad_pin attribute
  nextstate_type attribute
  orders attribute
  output_signal_level attribute
  output_voltage attribute
pin_func_type attribute 3.1.2
prefer_tied attribute 3.1.2
primary_output attribute 3.1.2
pulling_current attribute 3.1.2
pulling_resistance attribute 3.1.2
pulse_clock attribute 3.1.2
related_bus_pins attribute 3.2.15
related_grond_pin attribute 3.1.2
related_output_pin attribute 3.2.15
related_pin attribute 3.2.15
related_power_pin attribute 3.1.2
rise_capacitance attribute 3.1.2
rise_current_slope_after_threshold attribute 3.1.2
rise_current_slope_before_threshold attribute 3.1.2
rise_time_after_threshold attribute 3.1.2
rise_time_before_threshold attribute 3.1.2
signal_type attribute 3.1.2
slew_control attribute 3.1.2
state_function attribute 3.1.2
test_output_only attribute 3.1.2
three_state attribute 3.1.2
timing group 3.2.15
tlatch group 3.2.16
vhdl_name attribute 3.1.2

pin names
  as function arguments 2.1.4
  starting with numerals 2.1.4 3.1.2

pins
  pin group

poly_template group 1.9.29

positive_unate value, of timing_sense 3.2.15

positive-edge-triggered devices, see rising-edge-triggered devices 2.1.4

power_cell_type attribute 2.1.2
power_gating_cell attribute 2.1.2
power_gating_pin attribute 3.1.3

power_level attribute
  in internal_power group 3.2.8
  in leakage_power group 2.1.4

power_lut_template group 1.9.30

power_lut_template report, example 1.9.30

power_poly_template group 1.9.31

power_rail attribute
  in operating_conditions group 1.9.24
  in power_supply group 1.9.32

color_supply group 1.9.32

color_supply group 1.9.32

color_supplies group 1.9.32

color_supplies group 1.9.32

power group 3.2.8

color_power_group

power lookup table template
  example 1.9.30

color_power_group

power units 1.6.14

prefer_tied attribute 3.1.2

preferred_input_pad_voltage attribute 1.6.24
preferred_output_pad_slew_rate_control attribute 1.6.22 1.6.23
preferred_output_pad_voltage attribute 1.6.25
preferred attribute 2.1.2
preset, timing_type value 3.2.15
preset attribute
  in ff_bank group 2.1.4
  in ff group 2.1.4
  in latch_bank group 2.1.4
  in latch group 2.1.4
primary_output attribute 3.1.2
process attribute 1.9.24
propagated_lut_template group 1.9.33
propagated_noise_height_above_high group 3.2.15
propagated_noise_height_below_low group 3.2.15
propagated_noise_height_high group 3.2.15
propagated_noise_height_low group 3.2.15
propagated_noise_peak_time_ratio_above_high group 3.2.15
propagated_noise_peak_time_ratio_below_low group 3.2.15
propagated_noise_peak_time_ratio_high group 3.2.15
propagated_noise_peak_time_ratio_low group 3.2.15
propagated_noise_width_above_high group 3.2.15
propagated_noise_width_below_low group 3.2.15
propagated_noise_width_high group 3.2.15
propagated_noise_width_low group 3.2.15
propogated_noise_high group 3.2.1
propogated_noise_low group 3.2.1
pulling_current attribute 3.1.2
pulling_resistance_unit attribute 1.6.26
pulling_resistance attribute 3.1.2
pulse_clock attribute 3.1.2

R
rail_connection attribute 2.1.3
range of bus members
  in related_pin 3.2.15
receiver_capacitance1_fall group
  in receiver_capacitance group 3.2.14
  in timing group 3.2.15
receiver_capacitance1_rise group
  in receiver_capacitance group 3.2.14
  in timing group 3.2.15
receiver_capacitance2_fall group
  in receiver_capacitance group 3.2.14
  in timing group 3.2.15

receiver_capacitance2_rise group
  in receiver_capacitance group 3.2.14
  in timing group 3.2.15

receiver_capacitance group  3.2.14

recovery_falling, timing_type value 3.2.15
recovery_rising, timing_type value 3.2.15

reducing file size 1.3

reference_time attribute 2.1.4 2.1.4 3.2.15

registers 2.1.4

related_bus_pins attribute 3.2.3
  in pin group 3.2.15

related_ground_pin attribute 3.1.2

related_inputs attribute 2.1.4

related_outputs attribute 2.1.4

related_output_pin attribute
  in pin group 3.2.15

related_outputs attribute 2.1.4

related_pg_pin attribute 2.1.4

related_pin attribute 3.2.3 3.2.8
  timing group 3.2.15

related_power_pin attribute 3.1.2

removal_falling, timing_type value 3.2.15
removal_rising, timing_type value 3.2.15

report_lib command
  in-place swap mode information 1.6.11
  routing layer information 1.8.8

report_lib -power command
  multiple power supply report 1.9.32

report_lib -vhdl_name command 2.1.2

resistance attribute 1.9.45 1.9.45

resource_usage attribute 2.1.3

retain_fall_slew group 3.2.15

retain_rise_slew group 3.2.15

retaining_fall group 3.2.15

retaining_rise group 3.2.15

revision attribute 1.6.27

rise_capacitance_range attribute 3.1.3

rise_capacitance_range group 3.2.13
  in pin_capacitance group 3.2.13

rise_capacitance attribute 3.1.2
rise_capacitance group 3.2.13
rise_constraint group 3.2.15
rise_current_slope_after_threshold attribute 3.1.2
rise_current_slope_before_threshold attribute 3.1.2
rise_delay_intercept attribute
timing group
CMOS libraries 3.2.15
rise_net_delay group 1.9.34
rise_pin_resistance attribute
timing group
CMOS libraries 3.2.15
rise_power group 3.2.8
rise_propagation group 3.2.15 3.2.15
rise_resistance attribute
timing group 3.2.15
rise_time_after_threshold attribute 3.1.2
rise_time_before_threshold attribute 3.1.2
rise_transition_degradation group 1.9.35
rise_transition group 3.2.15
rising_edge, timing_type value 3.2.15
rising_together_group attribute 3.2.8
rising-edge-triggered-devices 2.1.4
routing_layers attribute 1.8.8
routing_track group 2.1.4
total_track_area attribute 2.1.4
tracks attribute 2.1.4

S
scaled_cell group 1.9.36
scaling_factors attribute 2.1.2
scaling_factors group 1.9.40
scan-in pin
in pin group
in test cells 2.1.4
scan-in pin, inverted
in pin group
in test cells 2.1.4
scan input on JK flip-flop 2.1.4
scan-out pin 2.1.4
in pin group
in test cells 2.1.4
scan-out pin, inverted
in pin group
in test cells 2.1.4
sdf_cond_end attribute 3.2.15
sdf_cond_start attribute 3.2.15
sdf_cond attribute
  in min_pulse_width group 3.2.11
  in minimum_period group 3.2.12
  in mode_definition group 2.1.4
timing group 3.2.15
sdf_edges attribute
timing group 3.2.15
sequential cells, complex
  statetable group 2.1.4
set_dont_touch command 2.1.2 2.1.2
set_map_only command 2.1.2
setup_falling, timing_type value 3.2.15
setup_rising, timing_type value 3.2.15
shifts attribute
  in generated_clock group 2.1.4
short attribute
  in model group 2.2.1
signal_type attribute 2.1.4 3.1.2
  in pin group
    in test cells 2.1.4
test_clock 2.1.4
test_scan_clock 2.1.4
test_scan_clock_a 2.1.4
test_scan_clock_b 2.1.4
test_scan_enable 2.1.4
test_scan_enable_inverted 2.1.4
test_scan_in 2.1.4
test_scan_in_inverted 2.1.4
test_scan_out 2.1.4
test_scan_out_inverted 2.1.4
test pin types 2.1.4 3.1.2
simulation attribute 1.6.28
simulation library files
generating 1.6.28
single_bit_degenerate attribute 2.1.2
single-latch LSSD methodology
  pin identification 2.1.4 3.1.2
skew_falling, timing_type value 3.2.15
skew_rising, timing_type value 3.2.15
slave clock 2.1.4 2.1.4
slew_control attribute 3.1.2
slew_derate_from_library attribute 1.6.29
slew_lower_threshold_pct_fall attribute 1.6.30 3.1.2
slew_lower_threshold_pct_rise attribute 1.6.31 3.1.2
slew_type attribute 2.1.2
slew_upper_threshold_pct_fall attribute 1.6.32 3.1.2
slew_upper_threshold_pct_rise attribute 1.6.33 3.1.2
slew attribute 1.9.17
slew-rate control 3.1.2
slope_fall attribute 3.2.15
slope_rise attribute 3.2.15
slowest_factor attribute 1.9.42
speed_grade group 1.9.27
SR latch 2.1.4
stage_type attribute 3.2.1 3.2.1
state_function attribute 3.1.2
state declaration
clocked_on_also attribute 2.1.4 2.1.4
clocked_on attribute 2.1.4
state-dependent timing
in timing group
when attribute 3.2.15
statetable cells
inverted_output attribute 3.1.2
statetable group 2.1.4
state variables
for flip-flops 2.1.4
with function attribute 2.1.4
steady_state_current_high group 3.2.15
steady_state_current_low group 3.2.15
steady_state_current_tristate group 3.2.15
step_level attribute 1.9.27
switching_group group 2.1.4 2.1.4
switching_interval attribute 3.2.8
switching_together_group attribute 3.2.8

T
table attribute
statetable group 2.1.4
tables
lu_table_template group 1.9.19
tdisable attribute 3.2.16
technology attribute 1.8.9
technology library
(see attributes, technology library)
temperature attribute 1.9.24
test_cell group 2.1.4
ff_bank group 2.1.4
ff group 2.1.4
latch group 2.1.4 2.1.4
test_output_only attribute 2.1.4 3.1.2
test pin attributes
function 2.1.4
signal_type 2.1.4

test pins, naming 2.1.4

three_state attribute 3.1.2
    in multibit latch registers 2.1.4

three-state cell
    timing_sense attribute 3.2.15

threshold_voltage_group attribute 2.1.2

time_unit attribute 1.6.34

timing_model_type attribute 2.1.2

timing_range group 1.9.42
    timing_sense attribute 3.2.15
        values 3.2.15 3.2.15 3.2.15
    timing_type attribute 3.2.15

timing arc
    defining identical 3.2.15
        half-unate 3.2.15

timing constraints
    negative 2.1.2

timing group 1.9.41
    attribute defaults 1.7 1.7 1.7
        clock_gating_flag attribute 3.2.15
        default_timing attribute 3.2.15
        defaults 1.7 1.7 1.7
        fall_delay_intercept attribute 3.2.15
        fall_pin_resistance attribute 3.2.15
        fall_resistance attribute 3.2.15
        in pin group 3.2.15
        intrinsic_rise attribute 3.2.15
        related_bus_pins attribute 3.2.15
        related_output_pin attribute 3.2.15
        related_pin attribute 3.2.15
        retaining_fail group 3.2.15
        retaining_rise group 3.2.15
        rise_delay_intercept 3.2.15
        rise_pin_resistance attribute 3.2.15
        rise_resistance attribute 3.2.15
        sdf_cond_end attribute 3.2.15
        sdf_cond_start attribute 3.2.15
        sdf_cond attribute 3.2.15
        sdf_edges attribute 3.2.15
        slope_fall attribute 3.2.15
        slope_rise attribute 3.2.15
        steady_state_resistance_above_high attribute 3.2.15
        steady_state_resistance_below_low attribute 3.2.15
        steady_state_resistance_high attribute 3.2.15
        steady_state_resistance_low attribute 3.2.15
        tied_off attribute 3.2.15
        timing_sense attribute 3.2.15
        timing_type attribute 3.2.15
        when_end attribute 3.2.15
        when_start attribute 3.2.15
        when attribute 3.2.15
            conditional timing check 3.2.15 3.2.15
                state-dependent timing 3.2.15

tlatch group
    edge_type attribute 3.2.16
        in pin group 3.2.16
        tdisable attribute 3.2.16
total_track_area attribute 2.1.4

tracks attribute 2.1.4

transition time
   max_transition attribute 3.1.2

tree_type attribute 1.9.24

type group 1.9.43 2.1.4
   base_type attribute 1.9.43 2.1.4
   bit_from attribute 1.9.43 2.1.4
   bit_to attribute 1.9.43 2.1.4
   bit_width attribute 1.9.43 2.1.4
   data_type attribute 1.9.43 2.1.4
   downto attribute 1.9.43 2.1.4
   example 1.9.43 2.1.4
   typical_capacitances attribute 2.1.4

U

unate, definition of 3.2.15

units
   capacitive load 1.8.1
   voltage 1.6.35

upper group 3.2.13

use_for_size_only attribute 2.1.2

user_parameters group 1.9.44

user-defined
   attributes 1.8.3
   groups 1.8.5

V

valid_speed_grade attribute 1.9.27

valid_step_levels attribute 1.9.27

value attribute 2.1.4 2.1.4 2.1.4

variable_1 attribute
   power_lut_template group 1.9.30

variable_2 attribute
   power_lut_template group 1.9.30

variable_3 attribute
   power_lut_template group 1.9.30

variable_n_range attribute 1.9.29 1.9.31 3.2.13 3.2.13

variables attribute 1.9.29 1.9.29 1.9.31 1.9.31

VDD, voltage levels
   output_voltage group 1.9.26

vector group 2.1.4 3.2.15

vhdl_name attribute 2.1.2 3.1.2

VHDL models
   timing group
      when_end attribute 3.2.15
when_start attribute 3.2.15

vih voltage range 1.9.16
vil voltage range 1.9.16
vil voltage ratings 3.1.2
vmin voltage range 1.9.16
vimax voltage range 1.9.16
vimin voltage range 1.9.16

VITAL models
  conditional timing check 3.2.15
    handle_negative_constraint attribute 2.1.2

voltage
  input_voltage group 1.9.16
  output_voltage group 1.9.26

voltage_map attribute 1.8.10

voltage_name attribute 2.1.4

voltage_unit attribute 1.6.35

voltage attribute 1.9.24

voltage ranges
  input_voltage group 1.9.16
  output_voltage group 1.9.26

vol voltage ratings 3.1.2

VSS, voltage levels
  output_voltage group 1.9.26


W

when_end attribute
  in timing group
    in VHDL models 3.2.15

when_start attribute
  in timing group
    in VHDL models 3.2.15

when attribute 3.2.15
  conditional timing check
    in VITAL models 3.2.15
    in dynamic_current group 2.1.4
    in electromigration group
      in pin group 3.2.3
    in internal_power group
      in pin group 3.2.8
    in intrinsic_parasitic group 2.1.4
    in leakage_current group 2.1.4
    in leakage_power group 2.1.4
    in min_pulse_width group
      in pin group 3.2.11
      in minimum_period group
        in pin group 3.2.12
        in mode_definition group 2.1.4
        in timing group 3.2.15

width_coefficient attribute 3.2.4

wire_load_selection group 1.9.46

wire_load_table group 1.9.47

wire_load group 1.9.45
x

\texttt{x\_function\ attribute\ 3.1.2}