Timing Library Format Reference

Product Version 4.3
October 2000

© 2000 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 1-800-462-4522.

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

D
TLF Property Index

Preface

This manual contains reference information about the timing library format (TLF). This manual is intended for timing and power library developers. To use TLF you should be familiar with the concepts of cell modeling for both timing and power, circuit delays, and timing checks. You should also know how circuit delays and timing checks, and power dissipation are affected by circuit parameters.

The preface discusses the following:

Related Documents

TLF models can be used with these tools:

- Affirma™ Pearl® timing analyzer
- Envisia™ ultra placer
- Envisia clock tree generator
- Envisia Gate Ensemble® place and route
- Envisia Silicon Ensemble™ place and route
- Cadence® Preview floor planner

For a list of the documents describing these products, refer to the Alphabetic List of Products that you can access in the online documentation library.

- For information about the delay calculator algorithms that use TLF timing data, see the *Delay Calculation Algorithm Guide*.

Typographic and Syntax Conventions

The following syntax conventions are used to describe tool commands.

`literal`	Nonitalic words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names.
`argument`	Words in italics indicate user-defined arguments for which you must substitute a name or a value.
`|`	Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.
`[ ]`	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices from which you can choose one.
`{ }`	Braces are used with OR-bars and enclose a list of choices from which you must choose one.
`...`	Three dots (...) indicate that you can repeat the previous argument. If they are used with brackets, you can specify zero or more arguments. If they are used without brackets, you need to specify at least one argument, but you can specify more.

```
argument...: specify at least one, but more are possible
[argument]...: you can specify zero or more
```

Any characters not included in the list above are required by the language and must be entered literally.

1

Introducing the Timing Library Format

This chapter contains the definition and use of the timing library format (TLF) and shows an example.

Introduction

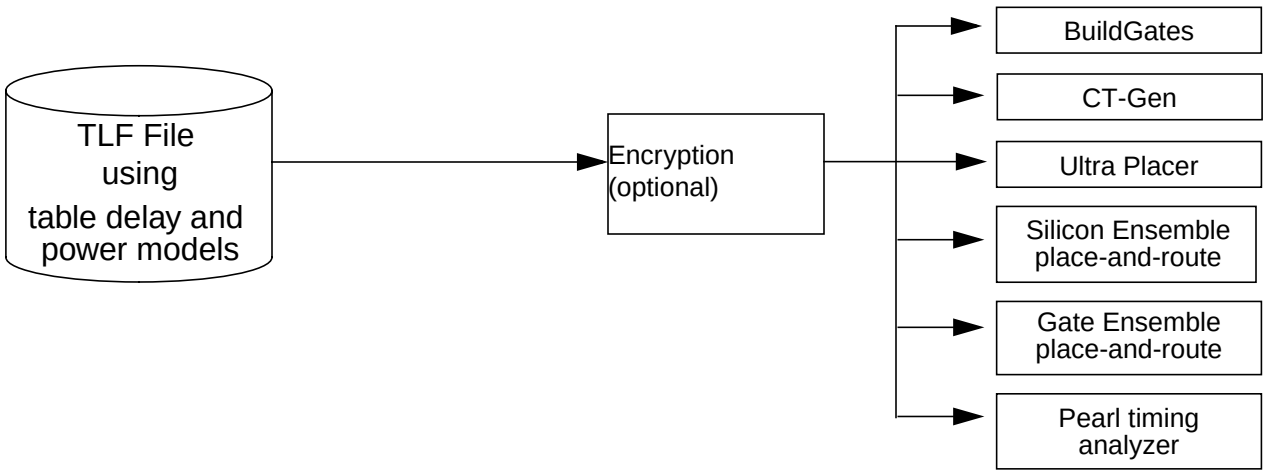
TLF is an ASCII representation of the timing and power parameters associated with any cell in a particular semiconductor technology. You can obtain the timing and power parameters of a particular cell by simulating the particular cell under a variety of conditions. Once you have obtained this data, you can then translate it into the TLF format for use in the Cadence timing environment.

A TLF library is used as input to all Cadence timing tools that can read a TLF library. The following tools are currently able to take advantage of the timing data included in TLF:

- Envisia™ ultra placer
- Envisia clock tree generator
- Envisia Gate Ensemble® place-and-route
- Envisia Silicon Ensemble™ place-and-route
- Affirma Pearl® timing analyzer
- Cadence® Preview floor planner

Some of these tools also read ASCII TLF. The timing algorithm that is common among all the timing tools in the Cadence flow is the Table delay algorithm. This algorithm is described in detail in the *Delay Calculation Algorithm Guide*.

Figure 1-1 TLF in the Timing Design Flow



Example TLF File

Figure 1-2 on page 16 shows a very simple TLF file example. It is divided into its basic scopes, ***Library*** and ***Cell***, and subdivided into *Headers*, *Models*, and *Timing Properties*.

The file describes a single cell (`inv`) with the minimum models needed to support it. The Table algorithm uses table delay and power models, also called “Spline.”

The file begins with library information common to the entire library of cells used in the design (only one cell, in this case). The library information includes a header section, models, and timing properties.

Each cell type in the design is contained within its scope of paired parentheses, and contains information for every model, pin, and path.

Figure 1-2 TLF File Example

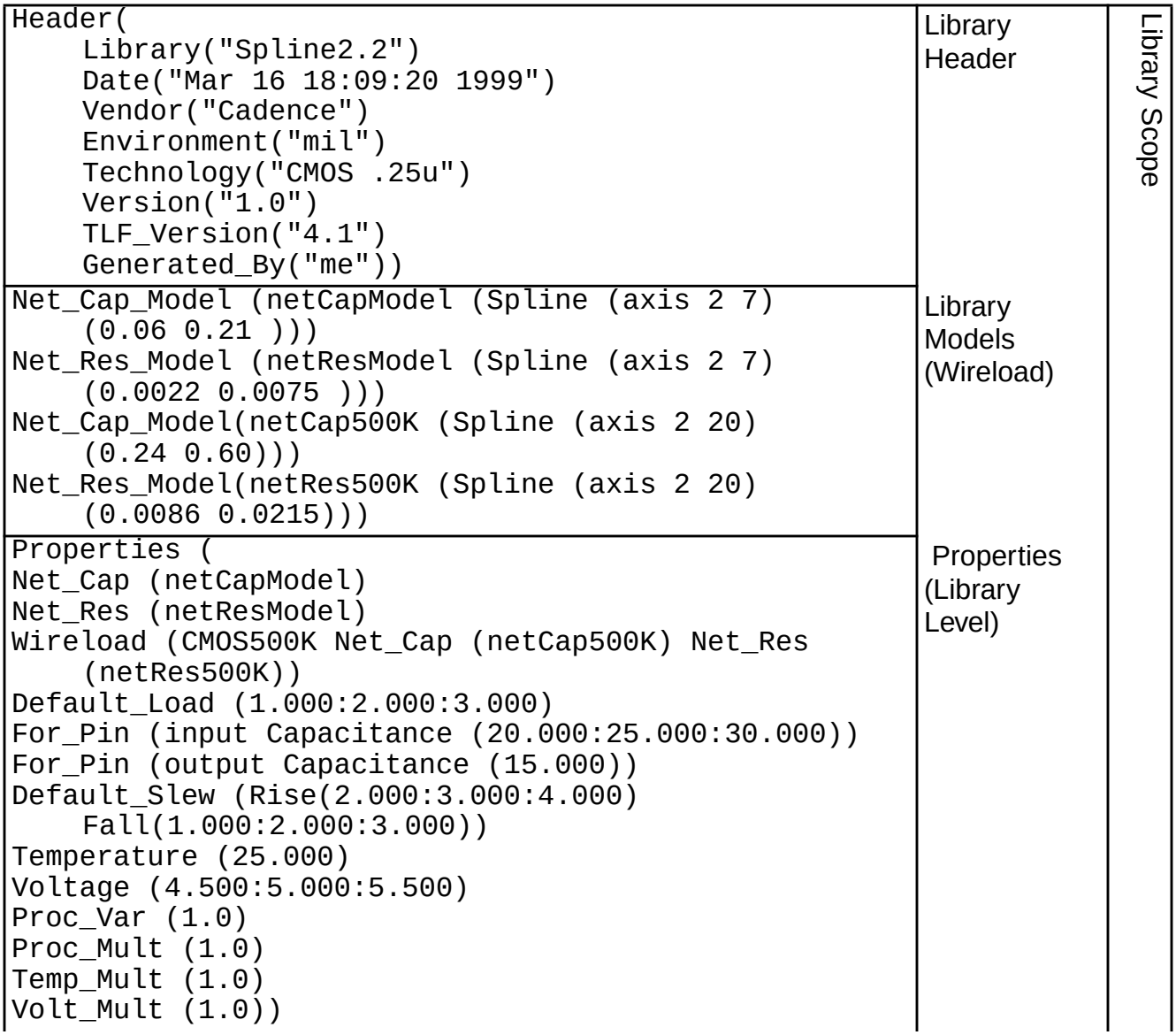


Figure 1-2 TLF File Example, *continued*

Cell(inv	Cell Header	Cell Scope
// Model Definition Timing_Model(td_A_to_Z_rise (Spline (load_axis 0.01 1.2) (input_slew_axis 1.0 2.0) ((0.1634 0.281) (0.5489 0.6679)))) Timing_Model(ts_A_to_Z_rise (Spline (load_axis 0.01 1.2) (input_slew_axis 1.0 2.0 4.0) ((0.106 0.1276 0.2394) (1.1775 1.1872 1.2531)))) Timing_Model(td_A_to_Z_fall (Spline (load_axis 0.01 1.2) (input_slew_axis 1.0 2.0) ((0.1549 0.4817) (0.7159 0.879)))) Timing_Model(ts_A_to_Z_fall (Spline (load_axis 0.01 1.2) (input_slew_axis 1.0 2.0 4.0) ((0.106 0.1276 0.2394) (1.1775 1.1872 1.2531))))	Cell Models	

Figure 1-2 TLF File Example, *continued*

```
Current_Model(2DTable1 output_by_cap_and_trans
    (Wavetable  (Load_Axis 0.000000 5.000000)
    (Input_Slew_Axis 0.000000 1.000000)
    data(
    ((-6.25 0)(-5.61 0)(3.99 0.000167772)
        (6.25 0.000239228)(22.9653 9.70859e-05)
        (40.2198 1.34283e-05)(62.8778 0))
    ((-5.61 0)(3.99 0.000167772)(6.25 0.00024718)
        (40.2393 0.000111914)(74.7681 1.69049e-05)
        (98.75 4.97199e-06)(128.75 0))
    ((-1.25 0)(3.99 0.000167772) (6.25 0.000239228)
        (22.9653 9.70859e-05)(40.2198 1.34283e-05)
        (62.8778 0)(128.75 0))
    ((-0.61 0)(3.99 0.000167772)(6.25 0.00024718)
        (40.2393 0.000111914)(74.7681 1.69049e-05)
        (98.75 4.97199e-06)(128.75 0))
    )
)
Current_Model(2DTable2 output_by_cap_and_trans
    (Wavetable  (Load_Axis 0.000000 5.000000)
    (Input_Slew_Axis 0.000000 1.000000)
    data(
    ((-8.25 0)(-5.61 0)(3.99 0.000167772)
        (6.25 0.000239228)(22.9653 9.70859e-05)
        (40.2198 1.34283e-05)(62.8778 0))
    ((-3.61 0)(3.99 0.000167772)(6.25 0.00024718)
        (40.2393 0.000111914)(74.7681 1.69049e-05)
        (98.75 4.97199e-06)(128.75 0))
    ((-0.25 0)(3.99 0.000167772) (6.25 0.000239228)
        (22.9653 9.70859e-05)(40.2198 1.34283e-05)
        (62.8778 0)(128.75 0))
    ((-0.1 0)(3.99 0.000167772)(6.25 0.00024718)
        (40.2393 0.000111914)(74.7681 1.69049e-05)
        (98.75 4.97199e-06)(128.75 0))
    )
)
```

Current Models — Cell Scope

```
        Volt_Mult_Propagation (Rise(1.1) Fall(1.2))
        Temp_Mult_Propagation (Rise(1.1) Fall(1.2))
        Volt_Mult_Transition (Rise(1.1) Fall(1.2))
        Temp_Mult_Transition (Rise(1.1) Fall(1.2))
```

Properties (Cell Level) — Cell Scope

```
    Pin(A Pintype(input) Capacitance(0.0267))
    Pin(Z Pintype(output) Pin_SPower(2.56))
```

Pins — Cell Scope

```
    Load_Limit (Warn (40) Error (50))
```

Properties (Pin Level) — Cell Scope

```
    Path(A => Z 10 01 Delay(td_A_to_Z_rise)
        Slew(ts_A_to_Z_rise)
        Supply_Current(2Dtable1)
        Ground_Current(2DTable2))
    Path(A => Z 01 10 Delay(td_A_to_Z_fall)
        Slew(ts_A_to_Z_fall)
```

Paths — Cell Scope

2

Delay Calculation Concepts

This chapter contains the following information:

Introduction

The information in a timing library format (TLF) file contains timing models and data to calculate I/O path delays, timing check values, and interconnect delays.

I/O path delays and timing check values are computed on a per-instance basis. “Cell-Based Delay Calculation” on page 21 describes some of the concepts used in these calculations.

Path delays in a circuit depend upon the electrical behavior of interconnects between cells. This parasitic information can be based on the layout of the design, but must be estimated when no layout information is available. “Interconnect Parasitic Estimation” on page 32 gives more details on what data to include in TLF for interconnect delay estimation.

Because actual operating conditions cannot be anticipated during characterization of delay data, derating models can be used to approximate the timing behavior of a particular cell at selected operating conditions. See “Modeling Process, Voltage, and Temperature Variations” on page 33 for more details on the TLF data that relate to PVT derating.

Parallel cells which are equivalent and share common input and output net connections can be treated as special when doing delay calculations. For more information about defining equivalent cells, see “Equivalent Cells” on page 33.

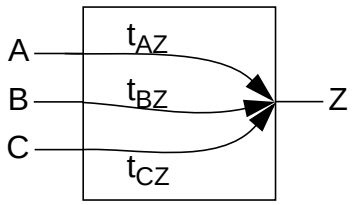
Cell-Based Delay Calculation

Cell-based delay calculation can be modeled accurately by characterizing the cell delay and the output transition time (output slew) as a function of the transition time of an input signal (input slew) and the capacitive load on the output of the cell. To efficiently calculate delays, the vendor must generate two models for each cell: the characterized cell delay model, and the characterized cell output slew model. (For definitions of cell delay and slew, refer to "Cell Delays and Signal Slews" on page 21.)

Similarly, timing checks are also functions of the input slew and output load. "Timing Checks" on page 22 defines all timing checks.

As cells are chained together, the output slew from a driving cell (driving stage) can be used to calculate the input slew of a receiving cell (receiving stage). The delay calculator can propagate slews from stage to stage using the characterized cell output slew model to compute the cell and interconnect delays.

Each cell has a specific number of input-to-output paths. For example:

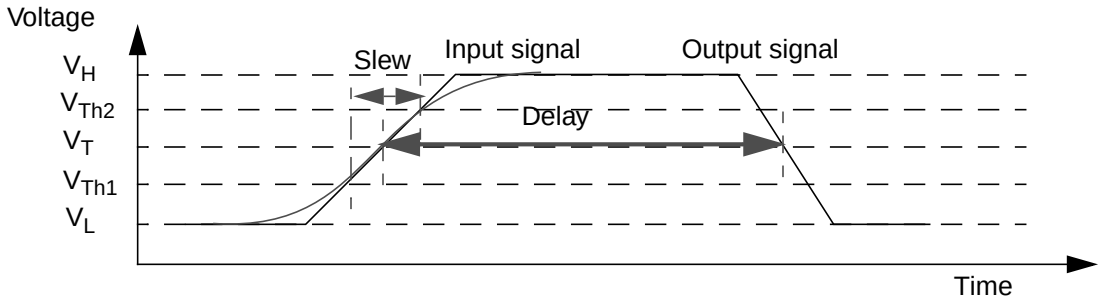


Path delays can be described for each input signal transition that affects an output signal, see the statement PATH on page 282. The path delay can also depend on signals at other inputs (see "State Dependencies" on page 28). In many sequential cells, the path delay from an input pin to an output pin can depend on the path delay from another output pin to this output pin (see "Output-to-Output Timing Paths" on page 31).

Cell Delays and Signal Slews

The transition time of a signal is an important factor in determining path delays. For delay calculation purposes, the waveform of a signal transition is modeled as a ramp, as shown below. Two threshold points (V_{Th1} and V_{Th2}) are chosen on the signal waveform and a straight line connecting these two points is extended to the on and off levels (V_H and V_L) of the signal. You can choose the threshold points at any voltage on the waveform. The time it takes for the signal to go from one threshold point to the other is referred to as the *slew.* If the data is characterized for use with the Table algorithm, the threshold points V_{Th1} and V_{Th2} can be

included in the TLF file through these properties: TABLE_TRANSITION_START on page 147 and TABLE_TRANSITION_END on page 149.



You can define a similar set of threshold points for the measurement of propagation delays. The propagation delay is then defined as the time difference between the input signal crossing a threshold voltage (V_{Tin}) and the output signal crossing its threshold voltage (V_{Tout}). In the figure above, the same threshold voltage (V_T) was chosen for the input and output threshold voltage. The V_{Tin} and V_{Tout} threshold points can be included in the TLF file through these properties: TABLE_INPUT_THRESHOLD on page 143 and TABLE_OUTPUT_THRESHOLD on page 145.

Library developers can choose the four threshold voltages (V_{Th1}, V_{Th2}, V_{Tin}, and V_{Tout}). For the path delay calculations to interact with the interconnect delay calculation, these threshold voltages must be known to the delay calculator.

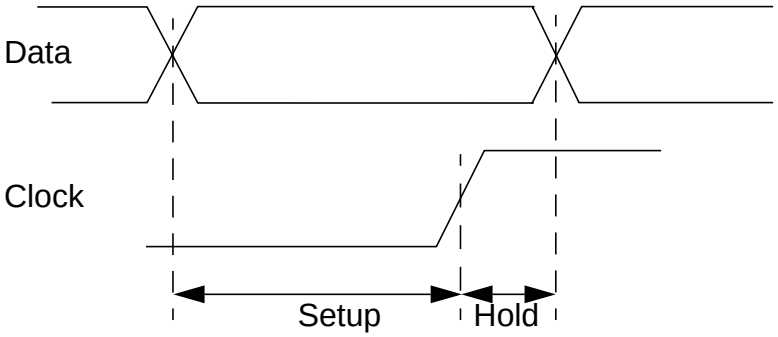
Timing Checks

In some cells input signals need to meet certain requirements or limits for the physical cell to operate correctly. These limits, which are often functions of design-dependent parameters like input slew or output load, are used during simulation to verify the operation of the cell. Models similar in concept to the delay or slew models are used to provide the data for computing timing checks. The following sections define the different timing checks.

Setup

The *setup* timing check specifies limit values for a setup time. In a flip-flop, the setup time is the time during which a data signal must remain stable before the clock edge. Any change to the data signal within this interval results in a timing violation. Figure 2-1 on page 23 shows a positive setup time—one occurring before the active edge of the clock. Figure 2-2 on page 24 shows the difference between a positive and negative setup time.

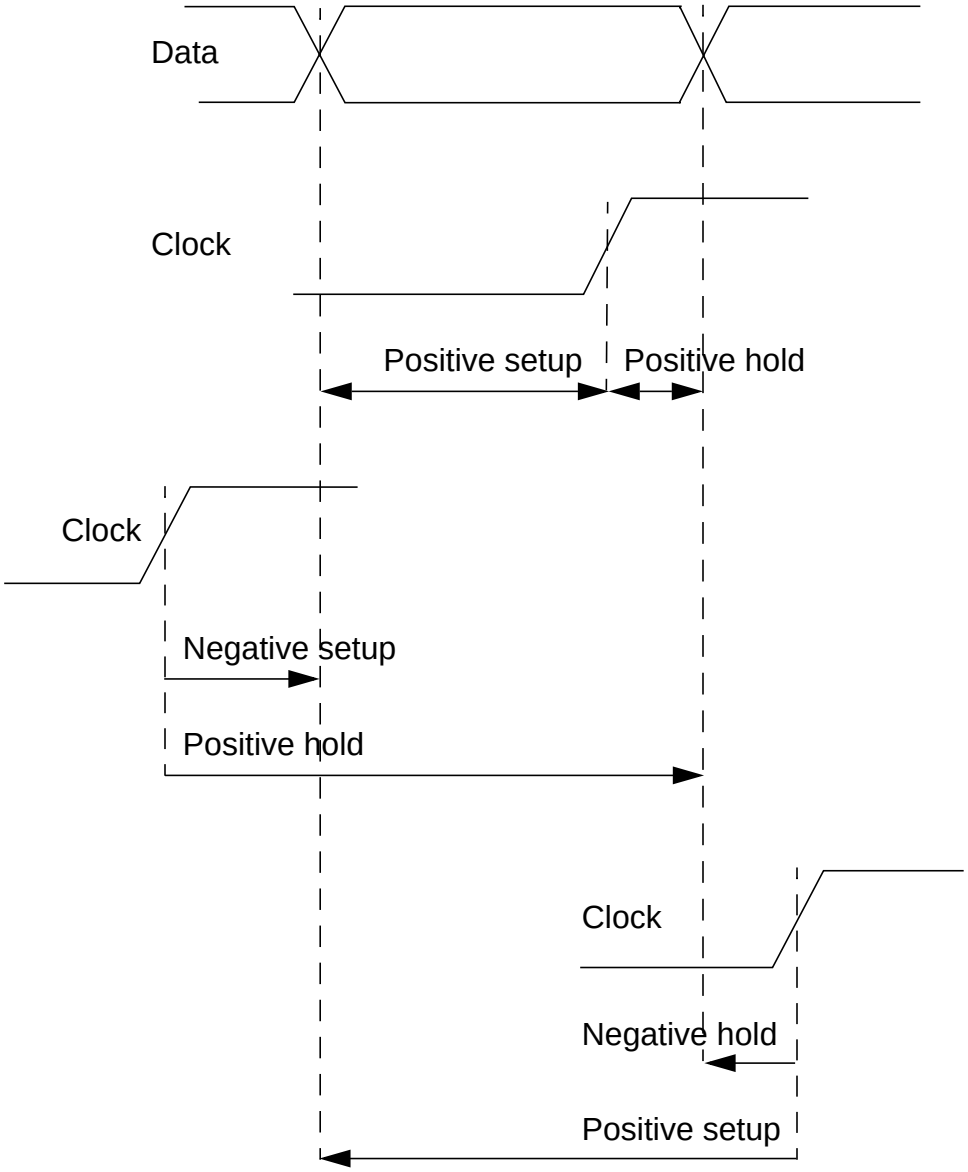
Figure 2-1 Positive Setup and Hold



Hold

The *hold* timing check specifies limit values for a hold time. In a flip-flop, the hold time is the time during which a data signal must remain stable after the clock edge. Any change to the data signal within this interval results in a timing violation. Figure 2-1 shows a positive hold time—one occurring after the clock edge. Other hold times are shown in Figure 2-2 on page 24.

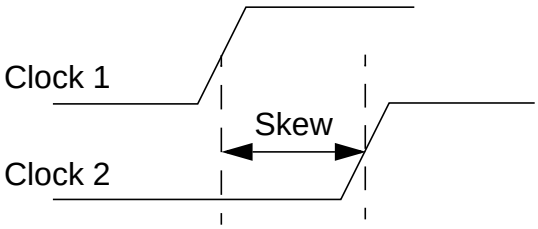
Figure 2-2 Positive and Negative Setup and Hold Times



Skew

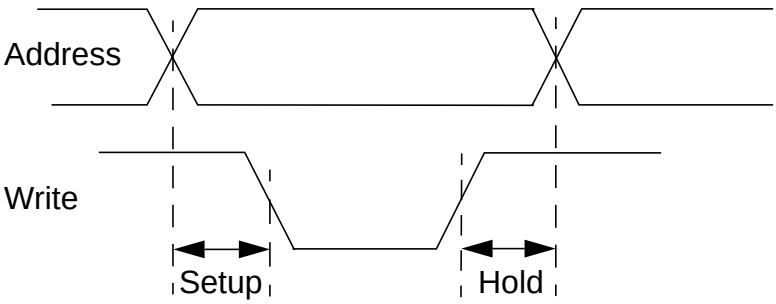
The *skew* timing check specifies the limit of the maximum allowable delay between two signals, which if exceeded causes devices to behave unreliably.

This timing check is often used in cells with multiple clocks.



No_Change

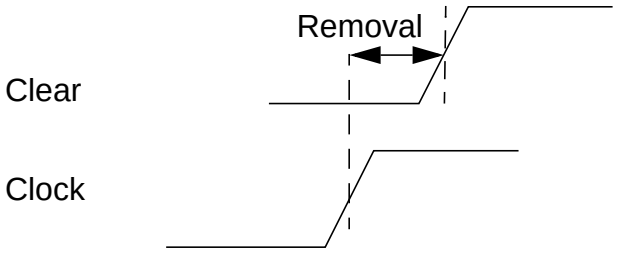
The *no_change* timing check is a signal check relative to the width of a control pulse. A “setup” period is established before the start of the control pulse and a “hold” period after the pulse. The signal checked against the control signal must remain stable during the setup period, the entire width of the pulse and the hold period. A *no_change* timing check is often used to model the timing of memory devices, when address lines must remain stable during a write pulse with margins both before and after the pulse.



Removal

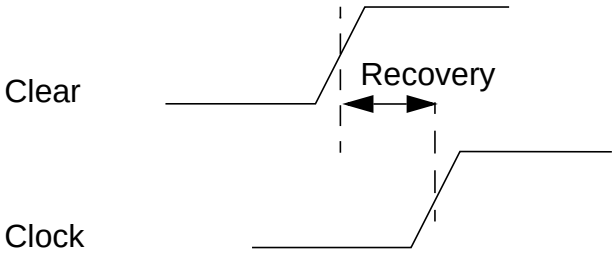
The *removal* timing check specifies a limit for the time allowed between an active clock edge and the release of an asynchronous control signal from the active state, for example, the time between the active edge of the clock and the release of the clear for a flip-flop.

If the release of the clear occurs too soon after the active clock edge, the state of the flip-flop becomes uncertain. The output can have the value set by the clear, or the value clocked into the flip-flop from the data input.



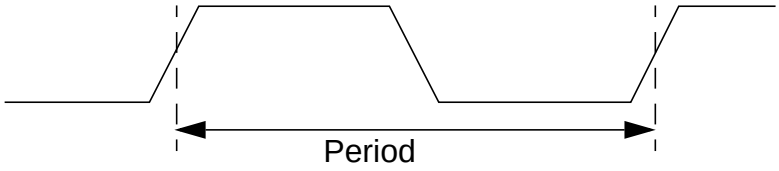
Recovery

The *recovery* timing check specifies a limit for the time allowed between the release of an asynchronous control signal from the active state and the next active clock edge, for example, a limit for the time between the release of the clear and the next active edge of the clock of a flip-flop. If the active clock edge occurs too soon after the release of the clear, the state of the flip-flop becomes uncertain. The output can have the value set by the clear, or the value clocked into the flip-flop from the data input.



Period

The *period* timing check specifies the minimum allowable time for one complete cycle (or period) of a signal.

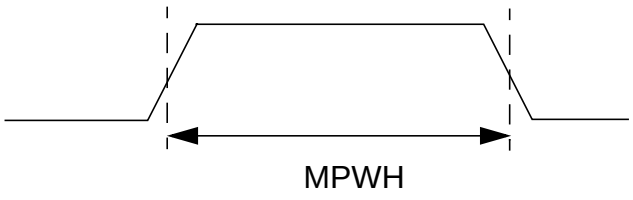


Minimum Pulse Width Low

The *MPWL* timing check specifies the minimum time a negative-going pulse must remain low. This timing check corresponds to the standard delay format (SDF) `WIDTH` timing check with a “negedge” specification.

Minimum Pulse Width High

The *MPWH* timing check specifies the minimum time a positive-going pulse must remain high. This timing check corresponds to the SDF `WIDTH` timing check with a “posedge” specification.



State Dependencies

In some cells, a path's output signal can depend not only on the signal at the path's input, but also on the existing logic states of other input pins of the cell. This state dependency adds another dimension to the delay definitions for a path; delay and slew models also need to be characterized for each set of states of other cell inputs which affect the output. This state dependency can be seen in an XOR gate.

Many timing checks compare the times of two different events. For example, a setup check compares a data edge to the clock edge. A conditional timing check can have separate conditions associated with each event that are evaluated at the time of the event. In TLF these conditions are known as the start and end conditions. You can use the `SDF_Cond_Start` and `Cond_Start` statements to specify the condition for the first event and the `SDF_Cond_End` and `Cond_End` statements to specify the condition for the second event in the timing check.

Verilog® simulator and SDF restrict the conditional expressions for timing checks to the value of a signal or its inverse, or to an equality test of the signal value. The following Verilog simulator setup check uses the `clr` signal as a condition for the clock edge.

```
$setup(data, posedge clk &&& clr, 10);
```

If the timing check condition depends on a more complicated expression, such as "`clr` or `set`," a natural way to write the Verilog simulator check would be:

```
$setup(data, posedge clk &&& (clr || set)), 10);
```

However, this expression is not a valid Verilog simulator expression. To get around this restriction, you can define an internal signal that computes the value of the expression in the Verilog simulator module for the primitive that contains the timing check. The following example shows a sample simulator model:

```
// internal signal clr_or_set defines
// condition outside specify block
or(clr_or_set, clr, set);
// timing check references internal signal
$setup(data, posedge clk &&& clr_or_set, 10);
```

The SDF file used to backannotate this conditional timing check must reference the internal signal `clr_or_set`.

```
(TIMINGCHECK
(SETUP (COND clr_or_set data) (posedge clk)     (0.1:0.1:0.1)))
```

To specify SDF condition expressions for timing checks, you can use the TLF statements: `SDF_COND`, `SDF_COND_START` and `SDF_COND_END`.

Note: The internal signal `clr_or_set` referenced by the SDF `COND` construct is visible only to tools that read the Verilog simulator cell definition. The TLF statements allow more general expressions than the Verilog simulator or SDF permit so that other tools can evaluate the condition expressions before evaluating the timing checks. See the `COND`, `COND_START`, and `COND_END` statements.

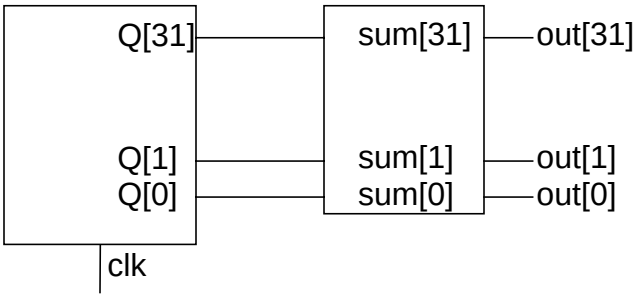
State-dependent path delays and timing checks can also depend on the internal state of registers inside a cell rather than the state of its pins. For example, a microprocessor core has an internal state that controls its path delays and timing checks. The processor core uses a bidirectional data bus to both read and write to the memory subsystem. During a read cycle the processor asserts the address and waits for the data to arrive. In the read mode a timing check on the data bus makes sure the data arrive on time. During a write cycle the processor sources the data bus, so there is a path from the clock to the data bus. The read/write mode is controlled by the internal state of the processor and not by any processor pins.

In TLF, you can define internal state variables as “internal” pins. Internal pins can be referenced by `COND`, `COND_START`, `COND_END`, `SDF_COND`, `SDF_COND_START`, and `SDF_COND_END` expressions.

Fast and Slow Paths

The number of state variables that control the delay between two pins in a complex block can be quite large. For example, the timing model for the circuit in Figure 2-3 has a path from `clk` to `out[31]`. The delay through this path depends on the state of register inputs that are clocked to the register outputs `Q[31:0]`. Because the internal state variables can have 2^{32} possible values, you would need 2^{32} different state-dependent delay paths to accurately describe the delay between `clk` and `out[31]`.

Figure 2-3 32-bit Register Driving Adder



Instead of listing all possible state-dependent paths between an input and output pin on the cell, you can use the FAST and SLOW options of the `Path` statement (see the `PATH` statement) to add the fastest and slowest paths between the pins to the timing model. You cannot use

fast and slow paths between a pair of pins with state-dependent timing paths between the same pins, or with a regular path (path without FAST and SLOW options) between the same pins. To model the paths between a pair of pins, you can use

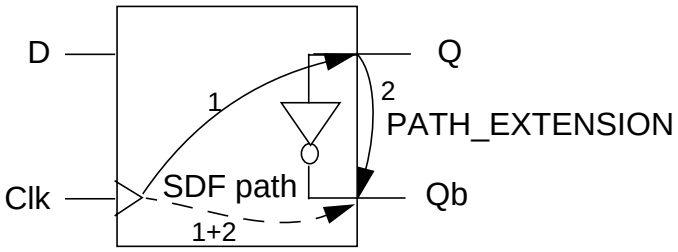
- One Path statement without any state dependencies
- Two or more Path statements containing COND or SDF_COND statements to express state dependencies
- One Path statement with a FAST option and one Path statement with the SLOW option

  The SDF generated uses the FAST path and the SLOW path for the minimum and maximum values in the delay triplet, respectively.

Output-to-Output Timing Paths

In many sequential cells (latches and flip-flops), the path delay from an input pin to an output pin depends on the path delay from another output pin to this output pin.

For example, consider the following D flip-flop with a dependent output Qb tied to the noninverted output Q through an inverter.



The path delay from Clk to Qb can be seen as the sum of the path delays from Clk to Q and from Q to Qb. The Clk to Q path is a normal delay path while the Q to Qb is an output-to-output path. The Verilog simulator does not allow output-to-output path statements, such as for Q to Qb, but instead requires both the Clk to Q and Clk to Qb Path statements.

To ensure that both paths Clk to Q, and Clk to Qb are included in the SDF file that is generated from TLF, you can describe the Clk to Q path using a Path statement and the Path_Extension statement to describe the Q to Qb output-to-output path. See the PATH_EXTENSION statement.

> **Note:** The Path_Extension statement is unnecessary for cells where both outputs are driven independently.

When using the Path_Extension capability, the application that reads TLF and generates the delays for this example cell assembles the Clk to Qb path from the Clk to Q path and the Q to Qb path. The application first recognizes that the Q to Qb path is a “path extension,” then it locates the path driving the source pin of the output-to-output path (driving the Q to Qb path); this path is the Clk to Q path. The delay of the two paths is calculated and then added; and the virtual path Clk to Qb is reported with the resulting delay.

Special Cases

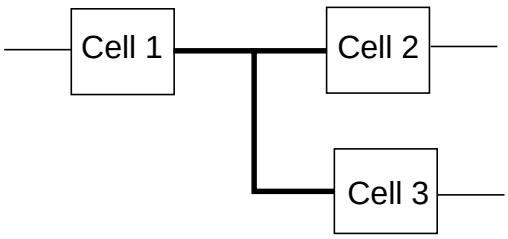
If an output-to-output path is driven from another output-to-output path, then the additional delay segment is added. Path extensions can be cascaded an arbitrary number of times.

If the source output pin of an output-to-output path is driven by more than one input pin, each resulting combination of paths is generated. To prevent this, you can include an OTHER_PINS statement in the Path_Extension statement to list the source pin (or pins) that drives the output-to-output path. See the OTHER_PINS statement.

Interconnect Parasitic Estimation

Accurate calculations of path delays in a circuit require information about the electrical behavior of interconnects between cells. The parasitic information is based on the connectivity and layout of the design and is generally generated in the form of a resistor/capacitor (RC) network for each interconnect.

In many cases delay calculation is done prior to the layout phase of the design, and the interconnect delays are based on the parasitic estimations of the delay calculator. The models used to estimate parasitics (also referred to as wireload models) are required by the delay calculator. In TLF syntax, these models are referred to as Net_Cap and Net_Res models (see the NET_CAP and NET_RES statements). These models provide values for the total capacitance and resistance of the interconnect as a function of the number of pins (including the driver pins), which are connected to the net. The following figure shows three cells with the interconnect highlighted.



TLF supports two syntaxes to describe a wireload model: the Wireload statement and the Wireload_By_xxx statement.

- The WIRELOAD statement specifies a named wireload model. A wireload model is known as a category (see WIRELOAD).
- The WIRELOAD_BY_XXX statements group wireloads models into a class where XXX is Area, Cell_Count, Gate_Count, or Transistor_Count (see WIRELOAD_BY_XXX).

  As design-related parameters like AREA, GATE_COUNT, and TRANSISTOR_COUNT vary, the interconnect lengths, and hence the delays, also vary. Pairs of the estimation models (one each for the capacitance and the resistance) can be specified for each particular value of these design-related parameters using a Wireload_By_xxx statement. Then,

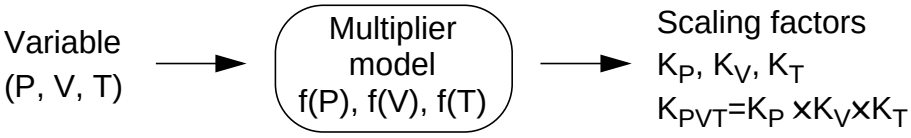
when calculating interconnect delays for a design, the delay calculator is flagged to use the appropriate pair of estimation models from the timing library. The wireload model can be referenced by either its category name or a parameter value (area, cell count, etc.). The parameter value is called a wireload category value.

Modeling Process, Voltage, and Temperature Variations

Process (P) conditions vary from one integrated circuit (IC) to another. During the operation of a particular IC, the voltage (V) and temperature (T) can vary slowly over time. At any instant in time, however, these variations are assumed to be small across a single IC.

Usually a timing library is characterized for a certain set of conditions: a particular process, voltage, and temperature. Based on the timing data in the timing library, the delay calculator reports pin-to-pin delays, interconnect delays, and timing check values. However, when the circuit operates under different conditions than those for which the library was characterized, the reported delay calculation values can differ from the actual values. To reflect the change in conditions, the delay calculator can scale the values.

TLF uses models to define scaling factors (or multipliers) for PVT variations. Each multiplier is determined using the model and the actual condition value. For example, the multiplier to account for voltage changes is calculated from the model `VOLT_MULT`, which is a function of the voltage. Similarly, the process and temperature multipliers are calculated from the models `PROC_MULT` and `TEMP_MULT`, which are functions of a process variable and the temperature, respectively. The three multipliers are then simultaneously used to derate the delays and timing checks.

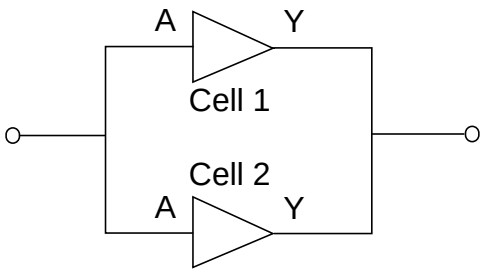


The P, V, and T variables can be used for best, typical, and worst case analysis and they can be specified in the form of triplets to reflect these cases. When the P, V, and T variables are in the form of triplets, the final derated delays are also in the form of triplets.

Equivalent Cells

In some designs, identical cells are connected in “parallel” to increase drive currents, as shown below. For cells to be considered in parallel, all the identical inputs and outputs must be tied together. Such configurations with identical cells can be recognized by the delay calculator so that they can be treated in a special way when doing delay calculations.

If cells are identical in behavior but not physically identical (for example, two buffers with different cells with different delay data), some delay calculators require the cells to be labeled as equivalent in order to recognize them as being in parallel. Only with such labelling can those delay calculators recognize these cells as being parallel and make the improvement in drive strength. Additionally, the corresponding pin names of the cells must match. That is, for two dissimilar buffers, pin names for both cells should be the same. In the example shown below, the input and output pins of both cell 1 and cell 2 are the same.



In TLF, you can declare cells to be equivalent using the statement EQ_CELLS on page 224.

3

Lexical Conventions and Data Types in TLF

This chapter contains the following information:

Lexical Conventions

A lexical unit (or token) is a sequence of characters in the text file that is treated as a single element, such as a number, a parenthesis, an expression operator, or a word. A single lexical element can contain up to 1000 characters. Comments, however, can be any length.

You must separate tokens of types identifier, number, Verilog® simulator integer constant, and transition using spacing, comments, strings, or any of the miscellaneous symbols (see “Miscellaneous Symbols” on page 40).

Spacing

Spaces, tabs, and the “newline” character split lexical units in timing library format (TLF) and are not syntactically significant. When used in quoted strings, however, these spacing characters are retained. For more information about strings, see “Strings” on page 36.

Comments

To include comments (text not interpreted by the TLF reader) you can

- Use a double slash (`//`) to start a comment that extends to the end of the same line.

```
// comment
```

- Use the symbols `/*` and `*/` to start and end a comment that can extend over several lines.

```
/* comment */
```

Within a comment, the only significant characters are those that form the corresponding comment terminator sequence:

- A *newline* character for "`//`" comments
- A "`*/`" for "`/*`" comments

All other characters within the comment text are ignored.

Note: Although the symbols "`//`" and "`/*`" can be included within either form of comment, they are interpreted only as part of the comment text; as a result, comments cannot be nested with the desired effect (the same applies to the comment terminator "`*/`" when contained in a "`//`"-style comment).

The TLF reader treats comments like a single space character. Comments separate other lexical units and, therefore, cannot be embedded inside any lexical unit.

Strings

A string is a sequence of characters surrounded by double-quotes (`"`). For example,

```
"tech05" "0132" "June96"
```

A double-quote character cannot be included within a string, because it would be interpreted as ending the string.

Note: TLF does not support the backslash (`\`) escape mechanism for strings.

Identifiers

Identifiers are sequences that can contain any of the following characters:

- A through Z (either uppercase or lowercase)
- 0 through 9
- Underscore (_)
- Dollar sign ($)
- Hierarchy delimiter (/)

In TLF, identifiers cannot begin with a digit, a dollar sign, or a hierarchy delimiter character. You can include other characters in an identifier by enclosing them in a quoted string. (Note that TLF does not have an escape mechanism, like the Verilog simulator, to include other characters in an identifier.)

Identifiers are used for naming many of the data units in TLF, such as cells and pins. Identifiers for one category of elements are considered separate from identifiers in other categories of elements; thus, a pin and a cell can have the same name.

User-defined identifiers are case sensitive: the cell names `AND3` and `and3` are different.

In TLF, numerous identifiers are used as syntactic keywords or property names. These identifiers are not case sensitive. The keyword identifiers `Cell`, `CELL` and `cell` are equivalent. Syntactic keywords are an identifier category of their own, as are property names. Consequently, you can use keywords and property names to identify cells, pins, or any other data units.

Transitions

Signal transitions refer to a changing signal value, a key component in timing descriptions. Following the Verilog simulator and standard delay format (SDF) conventions, the characters `0`, `1`, `Z`, and `X` indicate different signal values.

A transition in TLF is a combination of two different signal condition characters, as listed in the following transition symbols:

```
01    10    0Z    Z0    1Z    Z1
0X    X0    1X    X1    XZ    ZX
00    11    XX
```

Some transition symbols can be interpreted as either numbers, identifiers, or even invalid identifiers. After the extent of the lexical token is determined, the TLF reader uses context to

decide if the symbol should be interpreted as a transition or some other element. Therefore, the following sequence is always interpreted as an invalid three-character identifier—not as a transition followed by a single-character identifier.

```
0ZX
```

Numbers

The notation for a number in TLF follows the standard convention for “floating-point” numbers. TLF makes no distinction between integers and floating-point numbers: integers are always read as floating-point numbers.

In the following regular expression used to describe floating-point numbers the following conventions are used:

- Parentheses ( ) are used for precedence and grouping.
- A question mark (?) identifies an optional item.
- Square brackets and a dash ([-]) identify a range of valid characters.

  Characters listed between the square brackets are allowed. If the form [*x*-*y*] is used, all ASCII characters between *x* and *y* are allowed, *x* and *y* included.
- A vertical bar (|) identifies alternatives.
- An asterisk (*) identifies zero or more occurrences of the preceding element.

```
( + | - ) ?
( [0-9][0-9]* . ? | [0-9]* . [0-9][0-9]* )
( [eEdD] ( + | - ) ? [0-9][0-9]* ) ?
```

For example:

```
3.5E-2     3.567     1.111E-5    4     2e-8   12.000
```

Integer Numbers

Integer numbers represented in the Verilog simulator constant format are also recognized by TLF (however, they are accepted only within expressions). Verilog simulator format floating-point constants are ***not*** recognized.

The following expression describes a Verilog simulator integer number:

```
[0-9]* ' [bBoOdDhH] [0-9a-fA-FxXzZ][0-9a-fA-FxXzZ]*
```

See “Numbers” on page 38 for an explanation of the notation. The expression first defines a width (optional) in bits as a decimal number, then a base for the data bits with the single-quote mark and a base indicator character, and finally the actual digits for the data value. Note that, like the Verilog simualtor, the data digits can contain the letters x or z (upper- or lowercase).

The base indicator character defines the interpretation of the data digits, as follows:

```
Character    Base         Valid Digits (upper- or lowercase)
---------    ----    ----------------------------
b or B       binary       0, 1, x, z
o or O       octal        0-7, x, z
d or D       decimal      0-9
h or H       hexadecimal  0-9, a-f, x, z
```

For example,

```
4’b0100       4’Hc    6’B0x1zz1
```

The Verilog simulator also accepts simple decimal integers, which are also accepted in TLF. The following expression describes a simple decimal integer:

```
[0-9][0-9]*
```

See “Numbers” on page 38 for an explanation of the notation.

Note: The only integers that are valid in SDF expressions are the following one-bit binary numbers (or the alternatives using an uppercase 'b'):

```
0 1 'b0 'b1 1'b0 1'b1
```

Miscellaneous Symbols

Left and right facing parentheses, “(” and “)”, are used throughout TLF as basic structural indicators.

Square brackets, “[” and “]”, are used for indexing into, or indicating the size of, pin vectors (or “busses”).

The colon (:) is used with square brackets to separate range indices, as in “A[7:0]”, or to separate minimum-maximum pairs, or minimum-typical-maximum triplets, as in “1:2:3”. For more information, see “MTM Parameters and Values” on page 40.

The symbols made up of an equal sign or asterisk with a right angle bracket (=> and *>) are used to indicate either a one-to-one or an all-combinations relationship. These symbols are used in the PATH statement.

Within TLF data, the tilde (~) can be used to indicate an infinite value (or very large positive value). A tilde preceded by a minus sign (–~) likewise indicates a value of negative infinity (or a large negative value). However, neither of these interpretations is valid within expressions where the minus sign and tilde are interpreted as operators.

Expressions in TLF follow the Verilog syntax. The following is a list of operators or other such symbols:

```
+ - * / %
& | ^ ~
! && || == != === !==
<= >= < > << >>
? :
( )
{ , }
~& ~| ~^ ^~
```

Details on expression syntax and operator precedence can be found in “Expressions” on page 50.

Data Types

MTM Parameters and Values

The data values within a TLF file can represent behavior for one, two, or three different operating points, which correspond to minimum, typical, or maximum delay conditions.

Wherever you can specify minimum, typical, and maximum (MTM) values, you can also specify either a single floating-point value, or a minimum-maximum pair.

- If a TLF file contains data for one operating point, the values listed in the TLF file are simple floating-point numbers.
- If the TLF file describes two operating points, then the data values must be listed as two floating-point numbers separated by a colon (:). For example,

  ```
  4.5:5.5
  ```

  An optional syntax for data at two limit conditions is to separate the two data values with a double colon; this is similar to the SDF syntax for data at two limits. For example,

  ```
  4.5::5.5
  ```
- If data is provided for all three operating points, each data value is given as three separate floating-point numbers, separated by colons, as in:

  ```
  4.5:5.0:5.5
  ```
- Model and Table axis values cannot be MTM values. Only Model and Table data values can be in MTM format.

A parameter or value which can be defined as either a constant, a minimum-maximum pair, or minimum-typical-maximum triplet is referred to as an MTM parameter or value. An MTM constant contains all three values: minimum, typical and maximum. Sometimes both are referred to as simply an MTM.

In a TLF file you can mix single data values and MTM constants. However, the table data values should be either single data values or MTM constants.

If an infinite value is desired instead of some specific floating-point value, you can use a tilde (~). This is usually used for turning off error-check boundaries or for specifying an unlimited range for functions specified in a “slope-intercept” form (see Linear Segments on page 41). Negative infinite values are specified with a tilde preceded by a minus sign (-~). Internally the infinite value is represented as a floating-point number with a very large magnitude.

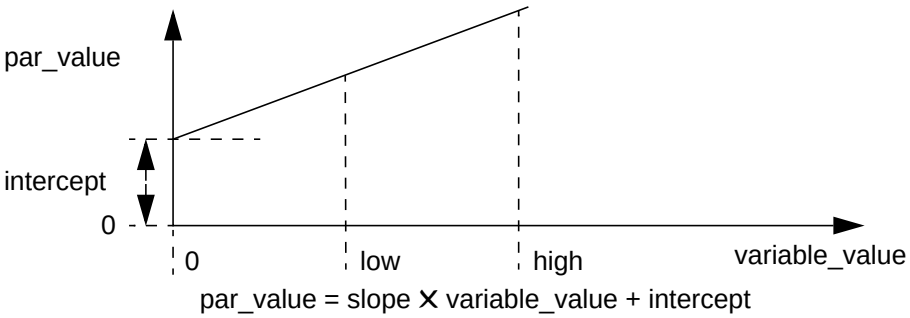
Linear Segments

Linear segments are used to define one-dimensional piecewise linear functions. Each segment defines a linear portion of a function over a part of the input variable range. One or more segments combined define a complete function.

A segment is a list of four values separated from each other by colons (floating-point numbers or the tilde (~) and negative tilde (-~) can be used for each of the values, but not MTM triplets). The four numbers are ordered as:

`low:high:intercept:slope`

`low, high`	Define the start and end points of the region defined by this segment.
`intercept`	Is the value for the linear function where the dependent variable is zero.
`slope`	Is the input variable multiplier.



Models

TLF models define functions that can be referenced by properties, delay paths, power and signal and design integrity constructs, or timing checks. For example, the delay from an input to an output is a function of the output load and the slew rate of the input signal. This example is a two-input or two-dimensional function.

A model is defined by

1. Selecting one of several built-in function forms (or “algorithms”)
2. Giving values for the coefficients of the form

The simplest algorithm is the constant function, which takes one coefficient, or parameter, that defines the function’s value regardless of any input variables. Other algorithms define functions that vary depending on the value of one, two, or three input variables (such as output load, input slew rate, and output load2 in the case of load-dependent outputs).

Models are defined either at the top level of the TLF file or within a cell definition. Models defined at the top level can be referenced anywhere within the TLF file after they have been defined. Models defined within a cell definition can be referenced only within that cell’s definition.

Syntax

For detailed information about model syntax, see usage_MODEL on page 348. The general syntax for defining a model is:

```
usage_MODEL(modelName [referenceModel]
    (algorithm
         { [parameter] (value) | (cond [parameter] (value))}...
    )
)
```

The following is an example of a simple constant delay model:

```
Timing_Model(Const_Delay (Const (1.3)))
```

In the model definition, *model_name* is a user-defined identifier that will be used later in the TLF file to reference the model.

default_model_name is an optional user-defined identifier that references a previously defined model. If a *default_model_name* is given, then the parameter list of the model referred to is used to define default values for the parameter list of the model being defined. Default parameter values are detailed below.

algorithm_name identifies the form of the function being defined; *algorithm_name* is one of the following:

`Const`	Constant function
`Linear`	One-dimensional piecewise linear function
`Spline`	One-, two-, or three-dimensional (linear) spline
`Table`	An alias for `Spline`

The `Const`, `Linear`, and `Spline` algorithms are described in "Constant Model" on page 45, "Linear Model" on page 45, and "Spline or Table Models" on page 46.

Usually the *parameter_list* in a model definition is a sequence of one or more parameter definitions. Normal parameter definitions have one of the following forms:

```
[parameter_name] ( value )
```

or

```
[parameter_name] ( axis_name value )
```

The second variation using *axis_name* is specific to the `Spline` or `Table` algorithm, and is explained below with details of the Spline algorithm.

The *parameter_name* in a parameter definition is optional. The parameter name is necessary to override values inherited from a default model, but can also be useful just for reference purposes.

The *value* for a model parameter can be an MTM value, a list of linear segments, a model reference, or a one-, two-, or three dimensional array of numbers (see "Spline or Table Models" on page 46).

Each algorithm uses a specific number of parameters. The order in which the parameters are listed is significant. For delay calculator-specific models, the number of parameters varies depending on the final usage of the model. (PWL models are not supported).

When inheriting parameters from a default model, the order of the parameters in the default model definition is always used regardless of the order in which they might be referenced in the inheriting model. All parameters of a default model are inherited, even parameters without a name. These parameter values can be over-written in the inheriting model, but a new parameter cannot be added to it. For example, the following model, `AxisBaseModel`, specifies only the axis values of a table model while it refers to the table model data by listing a parameter name of `data`.

```
Timing_Model(AxisBaseModel
    (Spline
        (load_axis 0.01 1.2 2.4 4.6 8.0 20.0)
        (input_slew_axis 1.0 2.0 4.0 8.0 16.0)
        data()
    )
    )
```

The following model, `DelayModel1`, references the default model, `AxisBaseModel`. It inherits the axis values from the default model and specifies the table model data.

```
Timing_Model(DelayModel1 AxisBaseModel
    (Spline
        data((0.3747 0.4818 0.6035 0.787  0.9605)
        (0.4562 0.5395 0.7685 0.949  1.1285)
        (1.143  1.1952 1.3034 1.6016 1.7826)
        (1.9346 2.0163 2.125  2.2559 2.4192)
        (3.4409 3.5225 3.7521 3.9239 4.1228)
        (8.1925 8.274  8.383  8.5036 8.8591))
    )
    )
```

A parameter list can also include parameter sublists to group the minimum, typical, and maximum values separately instead of using MTM values. In this form, a parameter list is:

`( mtm_keyword parameter_sublist )`

mtm_keyword is either min, typ, or max, indicating which subvalue is being defined for the subsequent parameters.

parameter_sublist is the same as the *parameter_list* (without the possibility of further nesting using the additional min, typ, max keywords). For example, the following two models are equivalent:

```
Timing_Model(const_model (Const val(1.0:2.0:3.0)))
```

```
Timing_Model(const_model
    (Const
        (min val(1.0))
        (typ val(2.0))
        (max val(3.0))
    )
    )
```

Note: The parameter within the min, typ, and max subclauses must be given a name. This form is primarily used in Spline models for defining separate minimum, typical, and maximum arrays, where using MTM values might cause confusion.

Constant Model

A Const model takes one parameter, a constant MTM value. This is the simplest possible model. For example,

```
Timing_Model(Const_Delay (Const (1.3)))
```

Linear Model

A Linear model is a one-dimensional piecewise linear function. The model takes a single parameter whose value is a list of one or more linear segments.

Note: All segments together make up the value for one parameter.

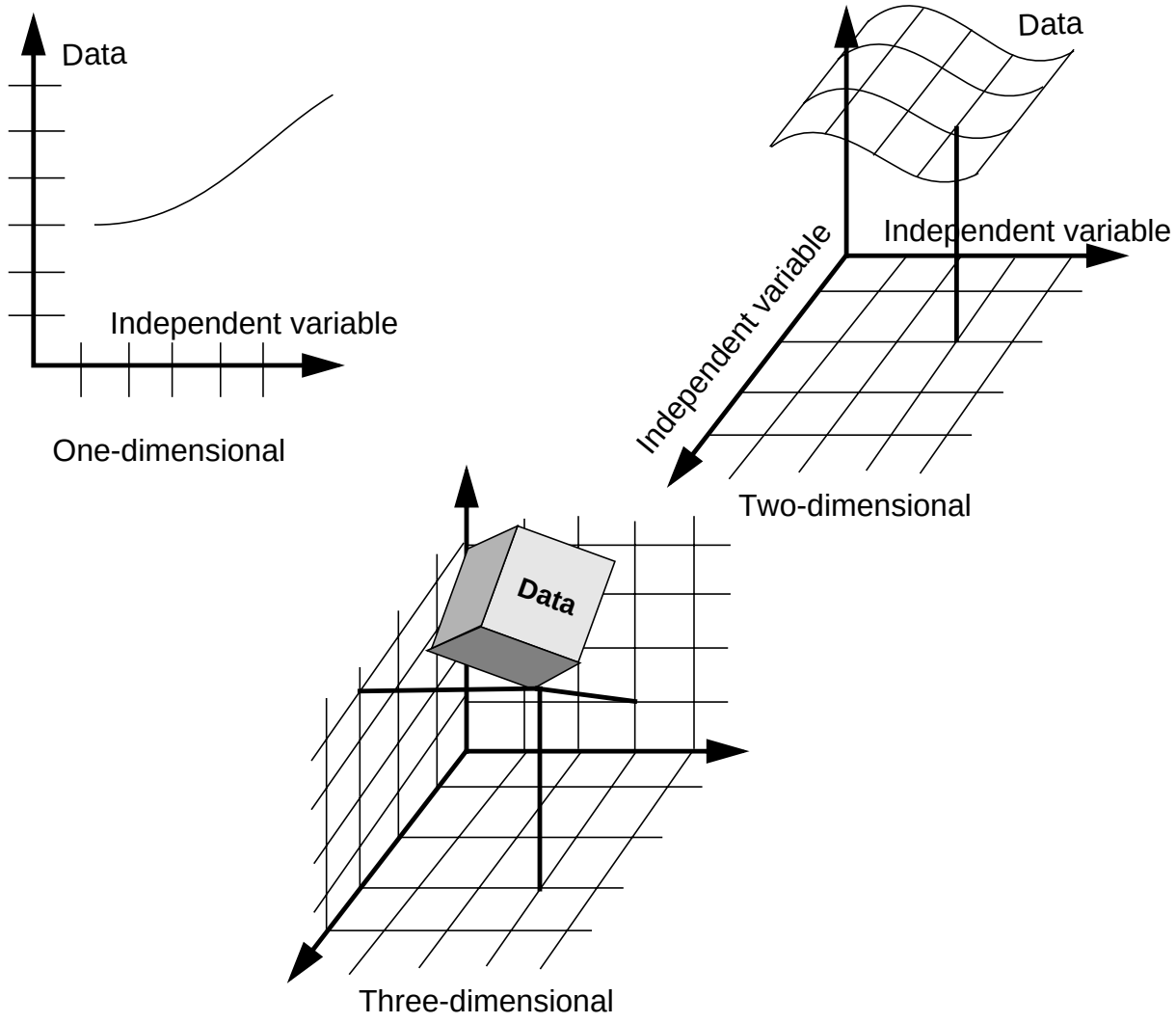
An example of a Linear model is:

```
Timing_Model(Linear_Delay
    (Linear
    (0:1.5:.03:1.0  1.5:10:0.33:0.8  10:~:7.33:0.1)
    )
    )
```

Endpoints of adjacent segments should define the same point on both the x and y axes.

Spline or Table Models

The `Spline` or `Table` models are one-, two-, or three dimensional functions given a set of sample points from the original function. Values between sample points are interpolated from the nearest sample values.



For the one-dimensional case, the algorithm takes two parameters, both an array of floating-point numbers. One array is the list of axis (input-variable) values at each sample point, and the other is the corresponding set of result (output-variable) values.

To distinguish the axis values from the data values, the modified parameter definition format is used:

`[`*`name`*`] ( `*`axis_name axis_values`*` )`

axis_name must be one of the following: axis, slew_axis, load_axis, input_slew_axis, or clock_slew_axis.

slew_axis and load_axis are used for defining delay, slew rate, or timing check functions, and are particularly useful for differentiating the two axes of a two-dimensional function.

input_slew_axis and clock_slew_axis are used for timing check functions.

axis_values list the floating-point numbers representing the input-variable values for each sample point. These values must be listed in increasing order. Note that the axis values do not have to be equally spaced.

The second parameter of a one-dimensional table consists of the output values for each sample point listed in the same order as the corresponding axis values.

Examples

An example of a one-dimensional Spline model is:

```
Timing_Model(spline_delay
    (Spline
        data(
            (axis 0.0 1.0 1.5 2.0 3.0 5.0)
            (     1.1 1.2 1.5 3.5 5.0 5.5))
    )
```

Two- and three-dimensional Spline models are very similar to the one-dimensional models. Additional axis parameters are necessary, and the output data values are represented as a two- or three-dimensional array of floating-point numbers (using additional sets of parentheses to indicate each row of data points in the array). The output data values correspond to the values of the original function on a grid defined by the axes.

An example of a two-dimensional Spline model is:

```
Timing_Model(spline_delay
    (Spline
        (load_axis 0.0  1.0  3.0  5.0)
        (slew_axis 0.0  0.25 0.50 1.00 2.0  5.0)
        data(
            ((1.1  1.2  1.5  3.5  5.0  5.5)
            (2.1  2.2  2.5  5.5  7.0  7.5)
            (5.1  5.5  5.9  7.5  9.0  9.5)
            (8.1  8.9  9.2  11.5 13.7 15.5))
        )
    )
)
```

An example of a three-dimensional Spline model is:

```
Timing_Model(TchkFallModel1
    (Spline
        (Clock_Slew_Axis 0.030000 0.810000)
        (Input_Slew_Axis 0.030000 0.810000)
        (Output_Load_Axis 0.00800 0.016000)
        data(
            ((0.059800, -0.179200) (0.0612, -0.1812))
            ((0.272700, 0.033700) (0.2936, 0.0412))

    )
)
```

Note: The order of the axis parameters determines the association of the axes to row or column. If the axes in the above example were listed in the opposite order, then the output data points would have to be transposed, as in the following example:

```
Timing_Model(exposed_spline_delay
    (Spline
        (slew_axis 0.0  0.25 0.5  1.00 2.0 5.0)
        (load_axis 0.0  1.0  3.0  5.0)
        data(
            ((1.1  2.1  5.1  8.1)
            (1.2  2.2  5.5  8.9)
            (1.5  2.5  5.9  9.2)
            (3.5  5.5  7.5  11.5)
            (5.0  7.0  9.0  13.7)
            (5.5  7.5  9.5  15.5))
        )
    )
)
```

Wavetable Model

Wavetable is a new algorithm type in TLF version 4.1. It appears in a *usage*_MODEL statement after model name and optional model reference.The wavetable algorithm is used to describe the current waveform as a set of current and time co-ordinates for each value of input slew and output loads. The wavetable algorithm describes the current waveform as a set of current and time co-ordinates for each value of input slew and output loads. It is used for current based dynamic power analysis.

Properties

A property is associated with a pin, cell, or library. The following is the syntax for properties at library level only:

```
Properties(
    property_name ( value )
    ...
)
```

Here is an example of a library level properties section:

```
PROPERTIES(
Default_Load(10.0) Slew_Limit(10.0)
    ...
)
```

All properties that are not at library level are specified by the property name itself. In the following cell level example, the values given for the cell overwrite the library level values of the previous example:

```
CELL(ABC
Default_Load(5.0) Slew_Limit(8.0)
    ...
)
```

In general, a property value can be an MTM value, a set of linear segments defining a piecewise linear function, or a table model reference. To check which type of value is valid for a specific property, refer to Chapter 5, “Properties” and Chapter 6, “TLF Statements.”

Note: If properties are specified multiple times at any of the library, cell, and pin levels, the property value specified last will overwrite all other property values specified for that property.

For slew-rate properties (that is, properties with the word `slew` in their name), the value can depend on the slew direction—whether it involves a rising or falling transition. In this case, the rising-transition value is listed within a `Rise` subclause and the corresponding falling-transition value is listed within a `Fall` subclause, as in this example:

```
Default_Slew(Rise(5.4) Fall(4.2))
```

Note: If the rising and falling slew values are the same, the `Rise` and `Fall` subclauses can be omitted and a single MTM value can be supplied.

The following example uses MTM triplets for the rising and falling slew values:

```
Default_Slew(Rise(4.1:5.4:6.7) Fall(3.8:4.2:4.9))
```

Limit properties (that is, properties with the word `limit` in their name) indicate values beyond which either warning or error messages are generated by the delay calculator. The tolerance

for warnings and errors can be defined separately with `Warn` and `Error` subclauses, similar to using `Rise` and `Fall` subclauses in slew properties. You can use MTM values.

```
Load_Limit(Warn(10) Error(20))
```

For the `Slew_Limit` property, which is both a limit and slew property, `Rise` and `Fall` subclauses may be nested within `Warn` and `Error` subclauses, as in:

```
Slew_Limit(Warn(Rise(9) Fall(12)) Error(Rise(10) Fall(13.5)))
```

Expressions

Symbolic expressions can appear in several places in a TLF file, most notably in describing the state for a delay or timing check. The syntax for expressions follows that for the Verilog simulator, or the restricted variation of the Verilog language format used in SDF.

Conditional Expressions

The following SDF conditional expressions are valid in TLF.

Conditions for Path Delays

The following definition applies to conditional expressions used in

- `SDF_Cond` and `Cond` statements used within `Path` statements
- `Cond`, `Cond_Start`, and `Cond_End` statements used within the timing check statements

```
conditional_expr
        ::= expression

expression
        ::= simple_expression
        ||= (expression)
        ||=  UNARY_OPERATOR (expression)
        ||= expression BINARY_OPERATOR expression

simple_expression
        ::= (simple_expression)
        ||= UNARY_OPERATOR (simple_expression)
        ||= port
        ||= UNARY_OPERATOR port
        ||= SCALAR_CONSTANT
        ||= UNARY_OPERATOR SCALAR_CONSTANT
        ||= simple_expression QM simple_expression
```

```
                        CLN simple_expression
            ||= {simple_expression concat_expression? }
            ||= {simple_expression {simple_expression
                        concat_expression? }}

    port
            ::= scalar_port
            ||= bus_port

    bus_port
            ::= IDENTIFIER[DNUMBER:DNUMBER]

    concat_expression
            ::= ,simple_expression

    QM
            ::= ?    // a literal question mark

    CLN
            ::= :    // a literal colon
```

SDF Conditions for Timing Checks

The following definition applies to conditional expressions used in `SDF_Cond`, `SDF_Cond_Start`, and `SDF_Cond_End` statements used within the timing check statements.

```
sdf_timing_check
        ::= scalar_node
        ||= INVERSION_OPERATOR scalar_node
        ||= scalar_node EQUALITY_OPERATOR SCALAR_CONSTANT

scalar_node
        ::= scalar_port
        ||= scalar_net

scalar_net
        ::= IDENTIFIER

scalar_port
        ::= IDENTIFIER
        ||= IDENTIFIER[DNUMBER]

DNUMBER represents a non-negative integer number.
```

Constants for Expressions

This section defines the logical constants used in TLF conditional path expressions and timing check conditions.

```
SCALAR_CONSTANT  ::= 1'b0   //logical zero
                 ||= 1'b1   //logical one
                 ||= 1'B0   //logical zero
                 ||= 1'B1   //logical one
                 ||= 'b0    //logical zero
                 ||= 'b1    //logical one
                 ||= 'B0    //logical zero
                 ||= 'B1    //logical one
                 ||= 0      //logical zero
                 ||= 1       //logical one
```

Operators for Expressions

This section defines the operators used in TLF conditional port expressions and timing check conditions.

```
UNARY_OPERATOR   ::= +  //arithmetic identity
                 ||= -    //arithmetic negation
                 ||= !    //logical negation
                 ||= ~    //bit-wise unary negation
                 ||= &    //reduction unary AND
                 ||= ~&    //reduction unary NAND
                 ||= |    //reduction unary OR
                 ||= ~|    //reduction unary NOR
                 ||= ^    //reduction unary XOR
                 ||= ^~    //reduction unary XNOR
                 ||= ~^    //reduction unary XNOR (alternative)

BINARY_OPERATOR  ::= +   //arithmetic sum
                 ||= -   //arithmetic difference
                 ||= *   //arithmetic product
                 ||= /   //arithmetic quotient
                 ||= %   //modulus
                 ||= ==  //logical equality
                 ||= !=  //logical inequality
                 ||= === //case equality
                 ||= !== //case inequality
                 ||= &&  //logical AND
                 ||= ||  //logical OR
                 ||= <   //relational
                 ||= <=  //relational
                 ||= >   //relational
                 ||= >=  //relational
                 ||= &   //bit-wise binary AND
                 ||= |   //bit-wise binary inclusive OR
                 ||= ^   //bit-wise binary exclusive OR
```

```
                    ||= ^~  //bit-wise binary equivalence
                    ||= ~^  //bit-wise binary equivalence (alternative)
                    ||= >>  //right shift
                    ||= <<  //left shift
INVERSION_OPERATOR ::= !  //logical negation
                     ||= ~  //bit-wise unary negation
EQUALITY_OPERATOR  ::= ==  //logical equality
                     ||= !=  //logical inequality
                     ||= === //case equality
                     ||= !== //case inequality
```

Operation of Equality Operators

This section defines the operation of equality operators used in TLF conditional port expressions and timing check conditions. These operators return a logical value representing the result of the comparison, which is `1` for `TRUE`, and `0` for `FALSE` but can also be `X`.

a `==` b (logical equality) will be `TRUE` (`1`) only if a and b are of known logical state (`0` or `1`) and equal, and `FALSE` (`0`) only if a and b are known and not equal. If either a or b is `X` or `Z`, the result is `X`.

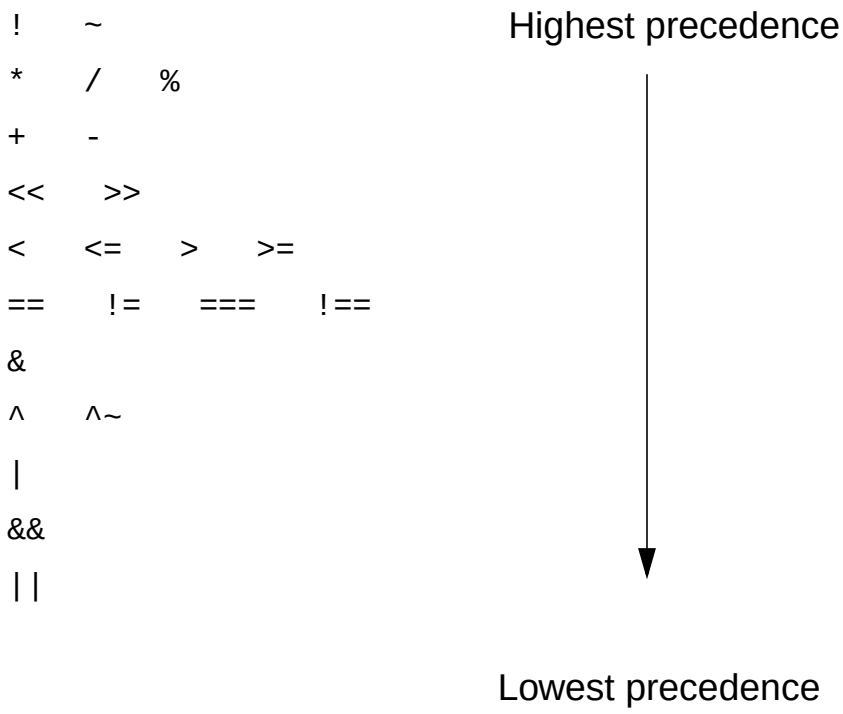
a `!=` b (logical inequality) will be `TRUE` (`1`) only if a and b are of known logical state (`0` or `1`) and not equal, and `FALSE` (`0`) only if a and b are known and equal. If either a or b is `X` or `Z`, the result is `X`.

a `===` b (case equality) will be `TRUE` (`1`) only if a and b are of the exact same logical state, including the `X` and `Z` states, and `FALSE` (`0`) otherwise.

a `!==` b (case inequality) will be `TRUE` (`1`) only if a and b are of different logical states, including the `X` and `Z` states, and `FALSE` (`0`) otherwise.

Precedence Rules of Operators

This section defines the precedence rules of the operators, in descending order.



4

TLF File Structure

This chapter contains the following information:

Introduction

A timing library format (TLF) file is organized into two major scopes: the library scope and the cell scope.

- Library-scope entries contain administrative information including vendor, technology used, as well as global models for timing, fluence, current data, and properties used to define net resistance and capacitance (wireloads), and more.
- Cell-scope entries define cells, such as inverters and flip-flops. Any general default values (for example, input pin capacitance) defined at the library scope can be redefined for the cell. The cell scope also contains the characteristics of every path through a cell, as well as pin information (including bit-specific information) for every pin of the particular cell being defined.

Global default values and specific characterization data can be specified in the TLF file using models and properties.

- Properties can be assigned to the library, cells, and pins.

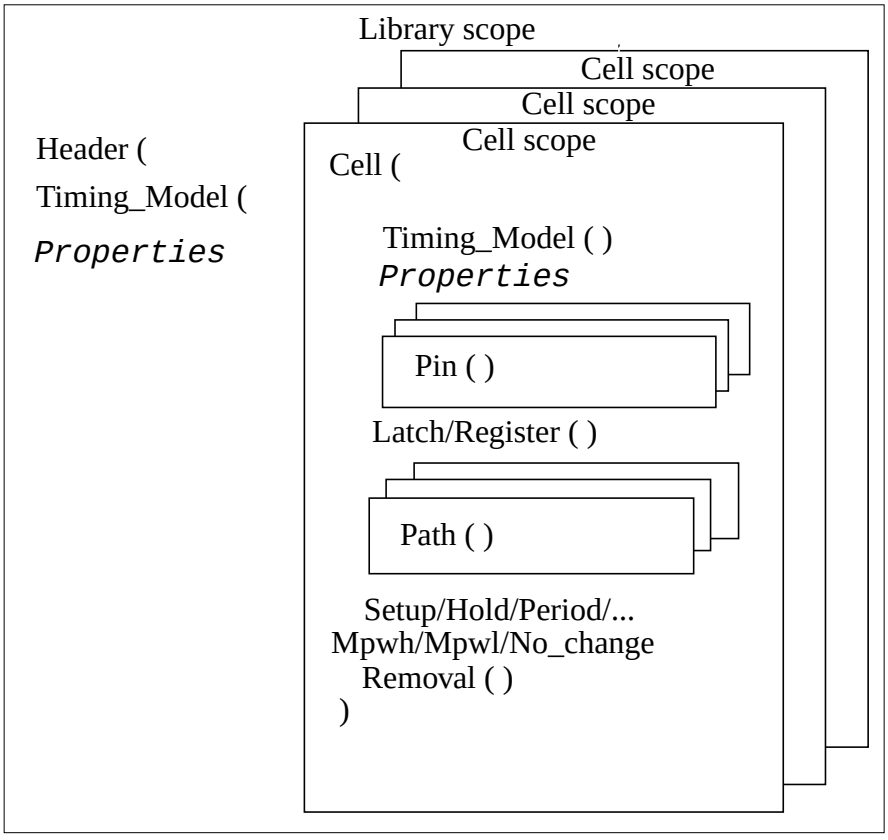
Library, cell, and pin are also referred to as the levels where properties (PROPERTIES) can be defined. For each property Chapter 5, “Properties,” and Chapter 6, “TLF Statements” list the lowest level at which the property can be specified.

In summary, the three levels where properties can be specified (library, cell, and pin) are contained within two scopes: library and cell.

- Models usually refer to timing data, and can also be a property value.

  Models can be specified at both the library scope and cell scope.

The information specified at the library scope (including models and properties) is global unless overwritten at the cell scope, which includes the cell and pin levels.



All three levels allow characterization through predefined properties in the TLF file. It is always possible to overwrite the properties of higher levels.

The vast majority of information is at the cell scope. A cell scope has several sections. One section defines the timing models used by that cell type for each internal cell path. Properties

read from the library level can be modified at the cell level, and overwritten again when defining pins at the pin level.

Figure 4-1 shows an example of a small TLF file.

Figure 4-1 TLF File Example

Header(Library("Spline2.2") Date("Mar 16 18:09:20 1999") Vendor("Cadence") Environment("mil") Technology("CMOS .25u") Version("1.0") TLF_Version("4.1") Generated_By("me"))	Library Header	Library Scope
Net_Cap_Model(netCapModel (Spline (axis 2 7) (0.06 0.21))) Net_Res_Model(netResModel (Spline (axis 2 7) (0.0022 0.0075))) Net_Cap_Model(netCap500K (Spline (axis 2 20) (0.24 0.60))) Net_Res_Model(netRes500K (Spline (axis 2 20) (0.0086 0.0215)))	**Library Models** (Wireload)	

Figure 4-1 TLF File Example, *continued*

Properties (
  Net_Cap (netCapModel)
  Net_Res (netResModel)
  Wireload (CMOS500K Net_Cap (netCap500K)
  Net_Res (netRes500K))
  Default_Load (1.000:2.000:3.000)
  For_Pin (input Capacitance
  (20.000:25.000:30.000))
  For_Pin (output Capacitance (15.000))
  Default_Slew (Rise(2.000:3.000:4.000)
  Fall(1.000:2.000:3.000))
  Temperature (25.000)
  Voltage (4.500:5.000:5.500)
  Proc_Var (1.0)
  Proc_Mult (1.0)
  Temp_Mult (1.0)
  Volt_Mult (1.0))
	Properties (Library Level)	Library Scope

Cell(inv	Cell Header	

// Model Definition
Timing_Model(td_A_to_Z_rise
  (Spline
  (load_axis 0.01 1.2)
  (input_slew_axis 1.0 2.0)
  ((0.1634 0.281)
  (0.5489 0.6679)))
)
	Cell Models	Cell Scope

Figure 4-1 TLF File Example, *continued*

`Timing_Model( ts_A_to_Z_rise`
`  (Spline`
`  (load_axis 0.01 1.2)`
`  (input_slew_axis 1.0 2.0 4.0)`
`  (( 0.106 0.1276 0.2394)`
`  ( 1.1775 1.1872 1.2531)))`
`  )`
`Timing_Model( td_A_to_Z_fall`
`  (Spline`
`  (load_axis 0.01 1.2)`
`  (input_slew_axis 1.0 2.0)`
`  (( 0.1549 0.4817)`
`  ( 0.7159 0.879)))`
`  )`
`Timing_Model( ts_A_to_Z_fall`
`  (Spline`
`  (load_axis 0.01 1.2)`
`  (input_slew_axis 1.0 2.0 4.0)`
`  (( 0.106 0.1276 0.2394)`
`  ( 1.1775 1.1872 1.2531)))`
`)
	Cell Models (continued)	Cell Scope

`Volt_Mult_Propagation (Rise(1.1) Fall(1.2))`
`Temp_Mult_Propagation (Rise(1.1) Fall(1.2))`
`Volt_Mult_Transition (Rise(1.1) Fall(1.2))`
`Temp_Mult_Transition (Rise(1.1) Fall(1.2))`
	Properties (Cell Level)	

`Pin(A Pintype(input) Capacitance(0.0267))` `Pin(Z Pintype(output))`	Pins	
`  Load_Limit (Warn (40) Error (50))`	**Properties** (Pin Level)	

`Path(A => Z 10 01 Delay(td_A_to_Z_rise)`
`  Slew(ts_A_to_Z_rise))`
`Path(A => Z 01 10 Delay(td_A_to_Z_fall)`
`  Slew(ts_A_to_Z_fall))`
`)`
	Paths	

Header

The `Header` statement at the library scope marks the beginning of the TLF file. The header section contains general library information. This information is specified in the form of: keyword (string).

The `Environment` statement indicates the type of applications for which the data is prederated or cornered.

The `Date`, `Vendor`, `Technology`, `Version`, and `Generated_By` statements document library-specific information. They have no impact on the data that is stored in the TLF library.

The `TLF_Version` statement specifies the version of the TLF language.

The TLF header information can be used to produce the standard delay format (SDF) header.

Library Models

The *usage*`_Model` keyword marks the beginning of the model information. When a model is referenced in the timing properties section at the library level, it must first be defined in this section. (Other models referenced at the cell level can also be defined here. Delay models and slew models are typical cell-level models.) A model can have a nested reference to another model.

Models described at the cell scope override models described at the library scope. Models that are not overridden are inherited from the library scope by the cell scope.

The wireload models (referenced by the `Net_Cap` and `Net_Res` timing properties) can only be specified at the library level. These models are used for interconnect estimation. These models combined with netlist, and RSPF data are used to produce the `INTERCONNECT` constructs in SDF.

The model names listed below are the names used in the example in Figure 4-1 on page 57. However, you can use any identifier for these models.

`netCapModel`	Specifies the model referenced by the `Net_Cap` timing property, which describes the general wire capacitance data for prelayout interconnect estimation.
`netResModel`	Specifies the model referenced by the `Net_Res` timing property, which describes the general wire resistance data for prelayout interconnect estimation.
`netCap500K`	Specifies the model referenced by the `Net_Cap` timing property in the `Wireload` statement, which describes specific wire capacitance data for prelayout interconnect delay estimation. This can be keyed by the size of the area, or cell count.

`netRes500K` Specifies the model referenced by the `Net_Res` timing property in the `Wireload` statement, which describes specific wire resistance data for prelayout interconnect delay estimation. This can be keyed by the size of the area, or cell count.

Properties

Properties described at the cell level override properties at the library level. Properties described at the pin level override properties at the cell or library level. Properties not overridden are inherited from the library by the cell level and then by the pin level. For example, for the *inv* cell in Figure 4-1 on page 57, the `Proc_Mult` property is not given at the cell level, so the value given at the library level is used.

Properties can be categorized. See Appendix D, “TLF Property Index” for a complete listing of all properties and categories.

The following properties are used by the delay calculator to derate the interconnect delays, pin-to-pin delays, and timing checks.

`Proc_Mult` Describes the process scaling factor of the PVT derating data. See PROC_MULT on page 120.

`Volt_Mult` Describes the voltage scaling factor of the PVT derating data. See VOLT_MULT on page 169.

`Temp_Mult` Describes the temperature scaling factor of the PVT derating data. See TEMP_MULT on page 152.

Cell Information

The `Cell` keyword marks the beginning of a cell scope. For the example in Figure 4-1 on page 57, cell-specific information includes:

`inv` Describes the name of the cell.

`usage``_Model` Describes the slew, delay, current, energy, and fluence characteristics of a given path through a given cell.

`Pin` Describes the name, type, and timing properties of this pin. For more information, see “Pin Information” on page 63.

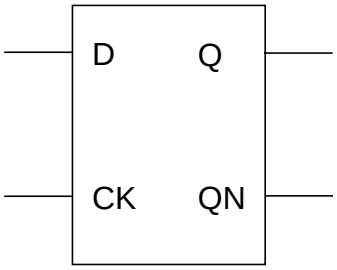
`Path` Describes the timing and power relationship between pins within a cell. For more information, see "Path Information" on page 64.

For a sequential cell, like the D flip-flop in Figure 4-2 on page 62, the following cell information can also be specified:

`Latch/Register` Tells whether a sequential device is a level-sensitive latch or an edge-triggered flip-flop. Provides the function designation of the pin. For more information, see "Latch/Register Information" on page 63

`timing_check` Describes the type, the path, and the constraint of a timing check. For more information, see "Timing Checks" on page 65.

Figure 4-2 D Flip-Flop



The cell scope information is similar to:

```
Cell (dff
     usage_Model(...)
     Pin(...)
     Register(...)
     Path(...)
     Setup(...)
)
```

In this example the first field, `dff`, is the cell name. This is the cell name used in the design. The second section contains a number of *usage*`_Model` statements. Models describe the timing relation of input/output pin pairs. The models can be referenced in `Path` statements and timing check statements. Models can also be referenced in cell or pin level properties. For more information, see usage_MODEL on page 348.

A `Pin` statement is required for each pin (`D`, `CK`, `Q`, and `QN` in the example).

The `Setup` statement is one of the timing check statements that can be specified for this sequential cell.

Pin Information

The `Pin` keyword marks the beginning of the information of a pin. For the example in Figure 4-1 on page 57, pin-specific information includes:

A	Describes the name of a pin.
`Pintype(data)`	Describes the pin category. Valid pin types are `input`, `output`, `bidir`, `ground`, and `supply`.

For the D flip-flop shown in Figure 4-2 on page 62, the information for pin D is:

```
    Pin(D Pintype(input) Capacitance(0.08))
```

The first field, `D`, is the pin name.

The `Pintype` statement indicates that pin `D` is an input pin.

The `Capacitance` statement defines the capacitance of the pin to be 0.08 where the units are set by the `UNITS` statement.

Latch/Register Information

The `Latch` and `Register` keywords mark the beginning of the sequential-logic behavior description of the cell. This information is included in the description of sequential cells for static timing analysis purposes.

For the edge-triggered `D` flip-flop shown in Figure 4-2 on page 62, the information might be similar to:

```
Register(
        Input(D)
        Clock(CLK)
        Output(Q)
        Inverted_Output(Q_)
        )
```

For a level-sensitive device the information might be similar to:

```
Latch(
    Input(D)
```

```
Clock(En)
Output(Q)
Inverted_Output(QB)
)
```

Note: If there are more than one `INPUT`, `CLOCK`, `OUTPUT`, or `INVERTED_OUTPUT` statements in a register or latch block, the latest definition overwrites all the others.

For more detailed syntax information, refer to the LATCH and the REGISTER statements on pages 6-253 and 6-301.

Path Information

The `Path` keyword marks the beginning of the information of a path. `Path` statements describe the pin-to-pin path, transition, and power, delay, and slew models of a path. The models depend on the type of delay algorithm used.

The TLF path information is used to produce the `IOPATH` construct in the SDF file. Each `Path` statement provides rise or fall transition data that corresponds to the rise/ fall data in the SDF file generated by the delay calculator. It takes two `Path` statements (in a TLF model) to completely describe one corresponding
I/O path delay in SDF.

Multiple states can be simplified to a `SLOW` and `FAST` path pair. TLF recognizes `0`, `1`, `X`, and `Z` states.

For the `D` flip-flop shown in Figure 4-2 on page 62, the path information from pin `CK` to pin `Q` is:

```
Path(CK => Q  01 01 Delay(ioDelRiseCKtoQ) Slew(slewRiseQ))
Path(CK => Q  01 10 Delay(ioDelFallCKtoQ) Slew(slewFallQ))
```

The first field, `CK => Q`, indicates a timing path is from input pin `CK` to output pin `Q`. (Other path types are “`D *> Z`.” Multiple bits of bus pin `D` can pass through output `Z`.)

`01 10` defines the input/output transition to which the path information applies. In this example, a `01` input transition causes a `01` or a `10` output transition.

The `Delay` statement contains the delay information for the path. `ioDelRiseCKtoQ` refers to the delay model name.

The `Slew` statement contains the slew information for the path. `slewRiseQ` refers to the slew model name.

Timing Checks

The following keywords mark the beginning of a timing check: `Setup`, `Hold`, `Period`, `MPWL`, `MPWH`, `Skew`, `Recovery`, `Removal`, and `No_Change`. Timing checks can be specified using the following syntax:

```
check_type (path [triggering_edge] model)
```

The delay calculator generates the timing check values in the corresponding SDF timing check statements based on the dynamic load and slew conditions of a design. The actual timing check is done by either a timing simulator or timing analyzer.

For example, consider the `D` flip-flop example in Figure 4-2 on page 62.

The setup timing check for the D flip-flop is similar to:

```
Setup (D => CK posEdge setUpModel)
```

The `Setup` keyword marks the beginning of the timing check. Timing checks are mostly used in sequential cells, but they can also be used in combinational cells: for example, a gated clock can use the `No_Change` timing check.

The timing check path is from input pin `D` to reference pin `CK`, which is an input clock pin in this case.

`posEdge` indicates that the positive edge of the reference pin `CK` is used to determine the setup check.

`setUpModel` refers to the model name for the setup timing check.

For more information about this timing check, refer to the SETUP statement on page 6-322.

5

Properties

This chapter contains the following information:

Note: In the timing library format (TLF) version 3.1, the properties were known as timing properties because they were mainly concerned with timing analysis. In TLF version 4.1, the TIMING_PROPS statement at library level has been replaced by the PROPERTIES statement to reflect the enhancements for signal and design integrity and power analysis. Properties at levels below library level are specified by their keyword.

The syntax for PROPERTIES is the same as that of TIMING_PROPS; however, the context has changed. While the TIMING_PROPS statement could appear at the Library, CELL, and PIN levels, PROPERTIES can only appear at library level. Properties at levels lower than library level must be specified directly by the property name (not within a PROPERTIES statement).

For an index of all properties and statements described in this manual, see Appendix D, "TLF Property Index."

Introduction

Properties can be associated with a library, cell, or pin. They can define defaults and limits for load and slew values, process, voltage and temperature (PVT) characterization values, wire load model information, crosstalk tolerance, estimated power, and more.

Using Properties

When using properties, remember:

- When specified at library level, the properties must be contained within the PROPERTIES section.
- At lower than library level, properties can be seen as keywords, because they introduce values, clauses, or reference models. When specifying a property at CELL, PIN, PATH, BUS or any level below library level, the keyword is used directly (not within a PROPERTIES section).

  Note: This is a change from earlier versions of TLF that required properties to be within a TIMING_PROPS statement at every level.
- TLF is not case sensitive. User specific data like the cell name and pin name, however, are case sensitive.
- You can disable checking for a particular parameter, such as a limit, by using a tilde (~) for the property value. For example:

```
Slew_Limit (Warn(~) Error(0.222))
```

Properties can be loosely grouped by purpose or usage. For example, any property name beginning with `Proc_` is a process variable designed to allow for inevitable differences between chip batches. The four properties beginning with `Table_` are used only by the Table algorithm for converting measurement points to the standard for the delay calculator.

For an index of all properties and keywords in alphabetical order or grouped by the purpose, see Appendix D, “TLF Property Index.”

Properties Description

This section alphabetically lists the properties used in the TLF file. For each property, you are given

- A syntax description and the possible values

  For a description of the syntax conventions, see Typographic and Syntax Conventions on page 12.

- A context section that describes
  - At which level the property can appear. At library level the property must be contained within the `PROPERTIES` section. At lower levels the property is applied directly without a `PROPERTIES` statement.
  - If the property is required
  - Any restrictions regarding algorithms
- A short description
- A description of the arguments or possible values for variables
- An example
- (optional) Related information

  If extensive details are presented in another part of this document, you are referred there. In many cases where properties work together in complex ways, such as with the environmental (PVT) variables, you are referred to the appropriate section of the *Delay Calculation Algorithm Guide* for more details.

AREA

Syntax

AREA(*value*)

Context

The AREA property can appear at both the library level within the PROPERTIES statement and at the CELL level. When specified at the library level, the area applies to all cells in the library.

Description

Assigns an area cost function to a cell, in square microns.

AREA can be used for two purposes:

- As a cost function during logic synthesis
- To compute a total area value during delay calculation that can then be used to select a net category

Arguments

value

Provides a floating number for the area in units specified by AREA_UNIT.

Example

```
Properties (...
    Area(2.5)
    )
Cell (...
    Area(3.0)
    )
```

Related Information

AREA_UNIT, WIRELOAD_BY_XXX

AREA_UNIT

Syntax

AREA_UNIT(*areaUnit*)

Context

An AREA_UNIT statement can appear within the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the unit of area in square microns. The default is 1squ.

Arguments

areaUnit

1squ | 10squ | 100squ

Example

```
AREA_UNIT(10squ)
```

Related Information

AREA, UNIT

CAPACITANCE

Syntax

CAPACITANCE(*value*)

Context

A CAPACITANCE statement can appear at the library level within the PROPERTIES statement and at the CELL and PIN levels. This statement can also appear within a BUS statement and applies to all pins in the bus.

The CAPACITANCE property is required by all algorithms.

Purpose

Specifies the capacitance of a pin.

Description

You can attach this property to an input, output, or bidirectional pin. For an output of a bidirectional pin, the value is the capacitance seen from outside the pin when the pin is not the driver. Therefore, you should attach this property to output pins only if they can be tristated.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Provides the capacitance of the pin(s) in units specified by the CAP_UNIT statement.

Examples

```
CAPACITANCE(0.0544)
CAPACITANCE(0.024:0.053)
CAPACITANCE(0.024:0.038:0.053)
```

Related Information

BUS, CAP_UNIT, PIN

CAP_UNIT

Syntax

CAP_UNIT(*capUnit*)

Context

A CAP_UNIT statement can appear in the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for capacitance. The default is 1pF.

Arguments

capUnit

1pF | 10pF | 100pF | 1fF | 10fF | 100fF | *float*pF

Examples

```
CAP_UNIT(10pF)
CAP_UNIT(0.0012pF)
```

Related Information

CAPACITANCE, RES_UNIT, UNIT

CELL_SPOWER

Syntax

CELL_SPOWER(*value*)

or

CELL_SPOWER(*value* COND(*cond*))

Note: The COND option can apply only at cell level.

Context

A CELL_SPOWER statement can appear at the library level within the PROPERTIES statement and at the CELL level. Library level specification is used for all cells having no CELL_SPOWER specified at cell level.

Purpose

Specifies the static power of the cell. Static power is consumed by the cell when none of the pins are switching. The value is the summation of static power consumed by all the pins of the cell. Static power at the pin level is modeled using PIN_SPOWER.

Description

Static power results from source to drain subthreshold leakage. This leakage is due to reduced threshold voltages that prevent the gate from turning off completely. Static power can also result from other conditions such as current leaks between the diffusion layers and substrate. Normally, static power is a small component of total power. It is important to model static power drain for the circuits which remain in idle state most of the time. Static power consumption of a cell varies with changes in operating conditions.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Specifies the value of the cell static power consumption (summation of static power consumed by all the pins of the cell). Units are specified by the POWER_UNIT statement.

cond

Specifies an expression that conditionalizes the CELL_SPOWER statement. Power specified is valid only if the condition is true.

Example

```
CELL(xyzBlock1
    PIN(...)
    PIN_SPOWER(1.56)
    ....)
CELL_SPOWER(2.45 COND(!A & B))
```

Related Information

PIN_SPOWER, POWER_UNIT

CONDUCTANCE_UNIT

Syntax

```
CONDUCTANCE_UNIT(condctUnit)
```

Context

A CONDUCTANCE_UNIT statement can appear in the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for conductance. The default is 1mS.

Arguments

condctUnit

1mS | 10 mS | 100mS

Example

```
CONDUCTANCE_UNIT(100mS)
```

Related Information

UNIT

CT_RES_LOW

Syntax

```
CT_RES_LOW {(value) | ( POSITIVE(value) NEGATIVE(value)) }
    value : float | min::max | min:typ:max
```

Context

A CT_RES_LOW statement can appear at the lIBRARY, CELL, or PIN level. Library specification will apply to all output pins without cell or pin specifications. Cell specification will apply to all output pins without pin specification for that cell. Pin level value overrides cell level value and cell level value overrides library level value. CT_RES_LOW applies to output (and bidirectional) pins. RES_UNIT value is assumed to be unit for CT_RES_LOW. Please note that both CT_RES_LOW and CT_RES_HIGH values need to be specified for a given output.

Description

A CT_RES_LOW statement specifies the crosstalk (static) resistance of the output pin with low state.

Example

```
Header(abc
        ...
        )
Properties(
        CT_RES_LOW(Positive(0.172) Negative(0.141))
        CT_RES_HIGH(Positive(0.182) Negative(0.151))
        ...
        )

Cell(xyz
    CT_RES_LOW(0.16)
    // both positive and negative values are 0.16
    CT_RES_HIGH(0.18)
    // both positive and negative values are 0.18

    ...
    Pin(Z Pintype(Output)
        CT_RES_LOW(Positive(0.15)Negative(0.13))
        CT_RES_HIGH(Positive(0.18) Negative(0.14) ...)
        Pin(Y Pintype(Bidir)
```

```
        CT_RES_LOW(0.14) CT_RES_HIGH(0.18) ...)
        Pin(X Pintype(Input) ... )
        )
```

Related Information

CT_RES_HIGH, WAVEFORM_TAIL_RES

CT_RES_HIGH

Syntax

```
CT_RES_HIGH {(value) | ( POSITIVE(value) NEGATIVE(value)) }
    value : float | min::max | min:typ:max
```

Context

A CT_RES_HIGH property can appear at the LIBRARY, CELL, or PIN level. Library specification will apply to all output pins without cell or pin specifications. Cell specification will apply to all output pins without pin specification for that cell. Pin level value overrides cell level value and cell level value overrides library level value. CT_RES_HIGH applies to output (and bidirectional) pins. RES_UNIT value is assumed to be unit for CT_RES_HIGH. Please note that both CT_RES_LOW and CT_RES_HIGH values need to be specified for a given output.

Description

A CT_RES_HIGH statement specifies the crosstalk (static) resistance of the output pin with high state.

Example

```
Header(abc
        ...
        )
Properties(
        CT_RES_LOW(Positive(0.172) Negative(0.141))
        CT_RES_HIGH(Positive(0.182) Negative(0.151))
        ...
        )

Cell(xyz
    CT_RES_LOW(0.16)
    // both positive and negative values are 0.16
    CT_RES_HIGH(0.18)
    // both positive and negative values are 0.18

    ...
    Pin(Z Pintype(Output)
        CT_RES_LOW(Positive(0.15)Negative(0.13))
        CT_RES_HIGH(Positive(0.18) Negative(0.14) ...)
```

```
        Pin(Y Pintype(Bidir)
        CT_RES_LOW(0.14) CT_RES_HIGH(0.18) ...)
        Pin(X Pintype(Input) ... )
        )
```

Related Information

CT_RES_LOW, WAVEFORM_TAIL_RES

CT_TOLERANCE

Syntax

```
CT_TOLERANCE( {value | [POSITIVE(value) NEGATIVE(value)]} )
```

Context

A CT_TOLERANCE statement can appear at the library level within the PROPERTIES statement and at the CELL and PIN levels. It can also appear within a BUS statement.

Library specification applies by default to all input pins without cell, bus or pin CT_TOLERANCE specifications. Cell specification applies by default to all input pins without bus or pin CT_TOLERANCE specification for that cell. Pin level value overrides cell level value and cell level value overrides library level value. CT_TOLERANCE applies to input (and bidirectional) pins.

Description

Specifies the crosstalk, in volts, that an input pin can tolerate. A single value will be used for both positive and negative tolerance. The POSITIVE and NEGATIVE keywords can be used to specify different values for positive and negative tolerance.

Arguments

value
: *float* | *min*::*max* | *min*:*typ*:*max*

POSITIVE
: Specifies that the value is for positive tolerance.

NEGATIVE
: Specifies that the value is for negative tolerance.

Example

```
Library(abc
        ...
        Properties(
        CT_TOLERANCE(POSITIVE(2.0) NEGATIVE(-0.5))
```

```
        ...
        )
Cell(xyz
    CT_TOLERANCE(1.2)  // both positive and negative tolerance is 1.2
    ...
    Pin(X Pintype(Input)
    CT_TOLERANCE(POSITIVE(1.5) NEGATIVE(0.1))
    ...
    )
Pin(Y Pintype(Bidir) CT_TOLERANCE(1.5) ...)
Pin(Z Pintype(Input) ... )
    )
)
```

CURRENT_UNIT

Syntax

CURRENT_UNIT(*currentUnit*)

Context

A CURRENT_UNIT statement can appear in the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for current. The default is 1mA.

Arguments

currentUnit

1mA | 10mA | 100mA | 1 uA | 10 uA | 100uA | 1A

Example

```
CURRENT_UNIT(10mA)
```

Related Information

UNIT

DEFAULT_LOAD

Syntax

DEFAULT_LOAD(*value*)

Context

The Default_Load property can appear at the library level within the PROPERTIES statement, and at the CELL and PIN levels.

Description

Specifies the default capacitance load for unconnected output pins or connected output pins that have no source.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Provides the value of the default load in units specified by the CAP_UNIT statement. The default unit is picofarads.

Example

```
Default_Load(10.000:20.000:30.000)
```

Related Information

CAP_UNIT

DEFAULT_PVT_COND

Syntax

DEFAULT_PVT_COND(*OpCondName*)

Context

A DEFAULT_PVT_COND statement can only appear at the library level within the PROPERTIES statement.

Purpose

Defines a default operating condition for the library. An operating condition contains process, voltage, temperature (PVT) values along with the definition for the environment interconnect model (optional).

Description

Values in the library are interpreted with respect to the operating condition which has been defined using DEFAULT_PVT_COND (unless the default is overridden).

PVT values defined using PROC_VAR, TEMPERATURE and VOLTAGE take higher precedence over the one defined using DEFAULT_PVT_COND.

Arguments

OpCondName

Specifies the default operating condition as defined earlier using the PVT_CONDS statement.

Example

```
DEFAULT_PVT_COND(Mil_70)
```

Related Information

PVT_CONDS

DEFAULT_SLEW

Syntax

DEFAULT_SLEW{(*value*)|(RISE(*value*) FALL(*value*))}

Context

The Default_Slew property can appear at the library level within the PROPERTIES statement, and at the CELL and PIN levels.

Description

The Default_Slew property specifies an input slew rate that is applied to unconnected input pins or pins that have no driver. You can specify separate values for rising and falling signals using the RISE and FALL keywords.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Provides the value of input slew time in units specified by TIME_UNIT. The default unit is nanoseconds.

Example

```
Default_Slew(Rise(2.100:3.100:4.100)
             Fall(1.200:2.200:3.200)
            )
```

Related Information

TIME_UNIT

DEFAULT_WIRELOAD_GROUP

Syntax

DEFAULT_WIRELOAD_GROUP(*WireLoadGroupName*)

Context

A DEFAULT_WIRELOAD_GROUP statement can appear at the library level within the PROPERTIES statement.

Purpose

Specifies the default WIRELOAD_BY_*XXX* group to be used for a design. In a library which contains multiple WIRELOAD_BY_XXX statements (used to group wire loads based on a parameter), this statement can be used to specify the default group.

Arguments

WireLoadGroupName

Specifies the WIRELOAD_BY_*XXX* group name as previously defined with the WIRELOAD_BY_XXX statement.

Example

```
DEFAULT_WIRELOAD_GROUP(metal_2)
```

See also

WIRELOAD_BY_XXX

DEFAULT_WIRELOAD_MODE

Syntax

DEFAULT_WIRELOAD_MODE(*ModeValue*)

Context

A DEFAULT_WIRELOAD_MODE statement can appear at the library level within the PROPERTIES statement.

Purpose

Used to specify how wire capacitance is computed for nets in a design.

Arguments

ModeValue
top | enclosed | segmented

top
For this mode, all nets at all hierarchical levels use the wire load model defined for the top level in the design.

enclosed
In this case, the wire load model of the smallest design that fully encloses the net is used. If the design enclosing the net has no wire load model, the tools search upward through the design hierarchy until a wire load model is found.

segmented
In this case, the nets that cross hierarchical boundaries are divided into segments and for each net segment, the wire load model of the design containing the segment is used. For the remaining nets, the enclosed mode is used.

Example

```
DEFAULT_WIRELOAD_MODE(enclosed)
```

Related Information

WIRELOAD, WIRELOAD_BY_XXX

FANOUT_LIMIT

Syntax

FANOUT_LIMIT(*value*)

or

FANOUT_LIMIT(WARN(*value*) ERROR(*value*))

Context

A FANOUT_LIMIT statement can appear at the library level within the PROPERTIES statement. It can also appear within the CELL, BUS, and PIN statements. If a FANOUT_LIMIT statement appears at the cell level, the value applies to all the pins in the cell. If it appears within the PROPERTIES statement, the value applies to all the pins in all the cells.

Purpose

Specifies the maximum fanout for an output pin.

Description

To perform this check, fanloads (specified using INPUT_FANLOAD) for all the input pins connected to this output are summed. TLF provides the values of the construct for warning and error message when you use the WARN and ERROR statements.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Example

```
FANOUT_LIMIT(5)
```

Related Information

FANOUT_MIN

FANOUT_MIN

Syntax

FANOUT_MIN(*value*)

or

FANOUT_MIN(WARN(*value*) ERROR(*value*))

Context

A FANOUT_MIN statement can appear at the library level within the PROPERTIES statement. It can also appear within the CELL, BUS, and PIN statements. If A FANOUT_MIN statement appears at the cell level, the value applies to all the pins in the cell. If it appears within the PROPERTIES statement, the value applies to all the pins in all the cells.

Purpose

Specifies the minimum fanout for an output pin.

Description

To perform this check, fanloads (specified using INPUT_FANLOAD) for all the input pins connected to this output are summed. TLF provides the values of the construct for warning and error message when you use the WARN and ERROR statements.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Example

FANOUT_MIN(1)

Related Information

FANOUT_LIMIT

FOR_PIN

Syntax

FOR_PIN(*pinType propertyName*(*value*) [*propertyName*(*value*)]...)

Context

A FOR_PIN statement can appear at the library level within the PROPERTIES statement and at the CELL level.

Description

Assigns properties to all pins of the given type.

Arguments

pinType

input | output| bidir | ground | supply | internal

Describes the type of the pins to which listed properties apply.

pinType	Use for . . .
input	Input pins
output	Output pins
bidir	Bidirectional pins
ground	Ground or VSS pins
supply	Power or VDD pins
internal	Internal nodes

propertyName

Specifies a default property for the specified pin type. If this property is set within a specific PIN statement, that value overrides this default. You can specify any of the properties listed in Table 5-1 on page 5-3 whose lowest level is *pin*.

value

Specifies the value of the default pin property.

Example

```
Properties (
        For_Pin(output Slew_Limit(Warn(~) Error(~)))
        For_Pin(input Capacitance (20.000:25.000:30.000))
        For_Pin(output Capacitance (15.000))
        )
```

Related Information

CELL

GATE_COUNT

Syntax

GATE_COUNT(*value*)

Context

The GATE_COUNT property can appear at the library level within the PROPERTIES statement and at the CELL level.

Description

The GATE_COUNT property assigns a gate count cost function to a cell.

GATE_COUNT represents the number of logic gates within the cell, and can be used for two purposes:

- As a cost function during logic synthesis (number of gates allowed)
- To compute a total area value during delay calculation that can then be used to select a net category

Arguments

value

Specifies the number of gates in the cell. Must be an integer.

Example

```
Gate_Count(1)
```

Related Information

WIRELOAD_BY_XXX

INDUCTANCE_UNIT

Syntax

```
INDUCTANCE_UNIT(inductanceUnit)
```

Context

A INDUCTANCE_UNIT statement can appear in the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for inductance. The default is pico henry (1pH.

Arguments

inductanceUnit

1pH | 10pH | 100pH

Example

```
INDUCTANCE_UNIT(10pH)
```

Related Information

UNIT

INPUT_FANLOAD

Syntax

INPUT_FANLOAD(*value*)

Context

An INPUT_FANLOAD statement can appear at library level within the PROPERTIES statement. It can also appear within CELL, BUS, and PIN statements. If an INPUT_FANLOAD statement appears at cell level, the value applies to all the pins in the cell. If it appears within the PROPERTIES statement, the value applies to all the pins in all the cells.

Purpose

Defines fanout load at input pin. It denotes load of input pin in terms of standard load. Fanout loads for all the input pins connected to an output pin are summed and the sum is checked against the maximum and minimum fanout values of the output pin for design rule violations.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Example

```
INPUT_FANLOAD(3.0)
```

Related Information

FANOUT_LIMIT, FANOUT_MIN

INPUT_THRESHOLD_PCT

Syntax

```
INPUT_THRESHOLD_PCT{(value)|(RISE(value)FALL(value))}
value: float
```

Context

The INPUT_THRESHOLD_PCT property can appear at the library level within the PROPERTIES statement and at the CELL levels.

Description

The INPUT_THRESHOLD_PCT property specifies the point where the delay measurement begins.This property is similar to the TABLE_INPUT_THRESHOLD property except that it is represented in percentage, and for both rising & falling signals the values are with respect to zero level.

Arguments

value

A floating number.

Example

```
Header(abc
    ...
        )
Properties(
        INPUT_THRESHOLD_PCT(42)
        // Both RISE and FALL values are assumed to be 42
        ...
        )
Cell(xyz
    INPUT_THRESHOLD_PCT(RISE(42)FALL(42))
    ...
    )
```

Related Information

OUTPUT_THRESHOLD_PCT, SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_OUTPUT_THRESHOLD

INPUT_VOLTAGE

Syntax

```
INPUT_VOLTAGE(
     groupName
     [VOLT_LOW_THRESHOLD(...)]
     [VOLT_HIGH_THRESHOLD(...)]
     [VOLT_MIN(...)]
     [VOLT_MAX(...)]
     )
```

At pin level, the following syntax is applicable:

`INPUT_VOLTAGE(`*`groupName`*`)`

Context

An `INPUT_VOLTAGE` statement can appear at library level within the `PROPERTIES` statement and at `PIN` level. This statement is optional and can appear multiple times in a library provided each statement specifies a unique *`groupName`*. Redefinition of an existing *`groupName`* is not permitted.

For more information, see the description of `PAD PROPS`

Purpose

Specifies the voltage levels for an input pad.

Arguments

`groupName`

Specifies the name of the `INPUT_VOLTAGE` group if it has been defined at library level. Within a pad cell, an `INPUT_VOLTAGE` defined at library level can be referenced by this name.

`VOLT_LOW_THRESHOLD`

Specifies the low threshold voltage level. Maximum input voltage for which input to core is guaranteed to be at logic 0.

VOLT_HIGH_THRESHOLD

Specifies the high threshold voltage level. Beyond this voltage signal value is considered to be at logic 1.

VOLT_MIN

Specifies the minimum acceptable input voltage.

VOLT_MAX

Specifies the maximum acceptable input voltage

Examples

```
INPUT_VOLTAGE(
    pad_bg
    VOLT_LOW_THRESHOLD(.3)
    VOLT_HIGH_THRESHOLD(4.3)
    VOLT_MIN(.1)
    VOLT_MAX(4.9)
    )
INPUT_VOLTAGE(pad_bg)
```

Also see the example of PAD_PROPS for more information.

Related Information

PAD_PROPS, VOLT_HIGH_THRESHOLD, VOLT_LOW_THRESHOLD, VOLT_MAX, VOLT_MIN

LOAD_LIMIT

Syntax

LOAD_LIMIT{(*value*)

or

LOAD_LIMIT{(WARN(*value*) ERROR(*value*))}

Context

The LOAD_LIMIT property can appear at the library level within the PROPERTIES statement and at the CELL and PIN levels.

Description

Specifies warning and error limits for the load on an output pin. Units are specified by CAP_UNIT. TLF provides the values of the construct for warning and error message when you use the WARN and ERROR statements.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Examples

```
Load_Limit(Warn(40) Error(50))
Load_Limit(Warn(30) Error(~))
```

The tilde (~) disables error checking.

LOAD_MIN

Syntax

LOAD_MIN(*value*)

or

LOAD_MIN(WARN(*value*) ERROR(*value*))

Context

A LOAD_MIN statement can appear at library level within the PROPERTIES statement. It can also appear within the CELL, BUS, and PIN statements.

Note: The tilde character (~) is internally represented as infinity. Warning and error checking is disabled if you use a statement like ERROR(~).

Purpose

Specifies the minimum load for an output pin. If it is defined at cell level, the value applies to all the pins in the cell. If it is defined at library level, the value applies to all the pins in all the cells. TLF provides the values of the construct for warning and error message when you use the WARN and ERROR statements.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Examples

```
LOAD_MIN(2.56)
LOAD_MIN(WARN(2.1) ERROR(1.9))
```

Related Information

LOAD_LIMIT

MIN_POROSITY

Syntax

MIN_POROSITY(*value*)

Context

A MIN_POROSITY statement can appear at the library level within the PROPERTIES statement.

Purpose

Specifies the minimum constraint on porosity. Porosity is the percentage of the total cell area that is routable; in other words, the total feedthrough area available divided by cell area. This constraint is used in routability optimization.

Arguments

value

Specifies a floating number greater than zero.

Example

```
MIN_POROSITY(15.0)
```

Also see example for Routing on page 393.

Related Information

ROUTING_LAYER, ROUTING_PROPS

NET_CAP

Syntax

NET_CAP(*value*)

Context

The NET_CAP property can appear at the library level within a PROPERTIES statement. The Net_Cap statement can also appear within the WIRELOAD and WIRELOAD_BY_XXX statements.

The NET_CAP property is required if the TLF file is used for prelayout interconnect estimation. It can be optional if the TLF file is used in postlayout delay calculations, where extracted interconnect parasitics are supplied.

Description

Specifies a model to estimate the total interconnect capacitance as a function of the number of pin connections on a net.

Arguments

value
: *constantValue* | *linearValue* | *linearModel* | *tableModel*

constantValue
: Specifies a constant floating point number.

linearValue
: Specifies a linear value expressed by a scale factor.

linearModel
: Specifies the name of a linear model as defined by the MODEL statement.

tableModel
: Specifies the name of a table model as defined by the MODEL statement.

Examples

```
Net_Cap(netcapModel)
Net_Cap(Spline(axis 2 7) (0.06 0.21))
```

Related Information

“Interconnect Parasitic Estimation” on page 2-17

NET_RES

Syntax

NET_RES(*value*)

Context

The NET_RES property can appear at the library level within the PROPERTIES statement and within the WIRELOAD and WIRELOAD_BY_XXX statements.

The NET_RES property is required if the TLF file is used for prelayout interconnect estimation. It can be optional if the TLF file is used in postlayout delay calculations, where extracted interconnect parasitics are supplied.

Description

The Net_Res property specifies a model to estimate the total interconnect resistance as a function of the number of pin connections on a net.

Arguments

value
: *constantValue* | *linearValue* | *linearModel* | *tableModel*

constantValue
: Specifies a constant floating point number.

linearValue
: Specifies a linear value expressed by a scale factor.

linearModel
: Specifies the name of a linear model as defined by the MODEL statement.

tableModel
: Specifies the name of a table model as defined by the MODEL statement.

Example

```
Net_Res(netresModel)
Net_Res(Spline(axis 2 6) (0.6 1.21))
```

Related Information

“Interconnect Parasitic Estimation” on page 2-17

OUTPUT_THRESHOLD_PCT

Syntax

OUTPUT_THRESHOLD_PCT{(*value*)|(RISE(value)FALL(value))}

value: float

Context

The OUTPUT_THRESHOLD_PCT property can appear at the library level within the PROPERTIES statement and at the CELL levels.

Description

The OUTPUT_THRESHOLD_PCT property specifies the point where the delay measurement ends.This property is similar to the TABLE_OUTPUT_THRESHOLD property except that it is represented in percentage, and for both rising & falling signals the values are with respect to zero level.

Arguments

value

A floating number.

Example

```
Header(abc
    ...
        )
Properties(
    OUTPUT_THRESHOLD_PCT(42)
    // Both RISE and FALL values are assumed to be 42
    ...
     )
Cell(xyz
    OUTPUT_THRESHOLD_PCT(RISE(42)FALL(42))
    ...
    )
```

Related Information

INTPUT_THRESHOLD_PCT, SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_OUTPUT_THRESHOLD

OUTPUT_VOLTAGE

Syntax

For library level:

```
OUTPUT_VOLTAGE(
     groupName
     [VOLT_LOW_THRESHOLD(...)]
     [VOLT_HIGH_THRESHOLD(...)]
     [VOLT_MIN(...)]
     [VOLT_MAX(...)]
     )
```

For pin level:

```
OUTPUT_VOLTAGE(groupName)
```

Context

An OUTPUT_VOLTAGE statement can appear at the library level within the PROPERTIES statement and at the PIN level. This statement is optional and can appear multiple times. At library level, multiple statements for the same group name are not allowed.

Purpose

Specifies the voltage levels for an output pad.

Arguments

groupName

At library level this defines a set of output voltage characteristics as a group and assigns a name to the group. At pin level within a pad cell, an OUTPUT_VOLTAGE defined at library level can be referenced by this name.

VOLT_LOW_THRESHOLD

Specifies the low threshold voltage level. It is the maximum output voltage for which the output is guaranteed to be at logic 0.

VOLT_HIGH_THRESHOLD

Specifies the high threshold voltage level. It is the minimum output voltage for which the output is guaranteed to be at logic 1.

VOLT_MIN

Specifies the minimum output voltage which the pad can generate.

VOLT_MAX

Specifies the maximum output voltage which the pad can generate.

Description

See the description of PAD_PROPS.

Example

```
OUTPUT_VOLTAGE(
    pad_bg
    VOLT_LOW_THRESHOLD(.3)
    VOLT_HIGH_THRESHOLD(4.3)
    VOLT_MIN(.1)
    VOLT_MAX(4.9)
    )
OUTPUT_VOLTAGE(pad_bg)
```

Also see the example of PAD_PROPS for more information.

Related Information

PAD_PROPS, VOLT_HIGH_THRESHOLD, VOLT_LOW_THRESHOLD, VOLT_MAX, VOLT_MIN

PIN_SPOWER

Syntax

PIN_SPOWER(*value*)

or

PIN_SPOWER(*value* COND(*cond*))

Note: The COND option can apply only at the cell, bus, or pin level.

Context

A PIN_SPOWER statement can appear at the library level within the PROPERTIES statement. It can also appear within the CELL, BUS, and PIN statements. Cell level specification is used for all pins having no PIN_SPOWER specified at pin level. Library level specification is used for all pins having no PIN_SPOWER specified at pin and cell level.

Purpose

Specifies the static power for a pin. Static power is the power consumed when there is no activity on the pin.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Provides the value of the static power in units specified by the POWER_UNIT statement.

cond

Specifies an expression that conditionalizes the PIN_SPOWER statement. Power specified is valid only if the condition is true. Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Example

```
CELL(
    xyzBlock1
    PIN_SPOWER(2.56)//for pins having no PIN_SPOWER at pin level
```

```
PIN(...
    PIN_SPOWER(2.50)
)
PIN(...
    PIN_SPOWER(2.45 COND(!A & B))
)
....
)
```

Related Information

CELL_SPOWER

POWER_ESTIMATE

Syntax

POWER_ESTIMATE(*value*)

Context

The Power_Estimate property can appear at both the library level within the PROPERTIES statement and at the CELL level.

Description

The Power_Estimate property is used for approximating power utilization.

Power_Estimate can be used as a cost function during logic synthesis. Because the value is used only in a relative sense to other cell power estimates, no units are assumed for this property.

Arguments

value
float

Example

```
Power_Estimate(1.0)
```

POWER_UNIT

Syntax

POWER_UNIT(*powerUnit*)

Context

A POWER_UNIT statement can appear in the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for power. The default is 1mW.

Arguments

powerUnit

1pW | 10pW | 100pW | 1nW | 10nW | 100nW | 1uW | 10uW | 100uW | 1mW

Example

```
POWER_UNIT(10uW)
```

Related Information

UNIT

PROC_MULT

Syntax

PROC_MULT (*value*)

or

PROC_MULT_*modeltype* {(*value*) |(RISE(*value*) FALL(*value*))}

Context

The PROC_MULT properties can appear at the LIBRARY level within the PROPERTIES statement and at the CELL level.

Description

Specifies a multiplier or model that can be used to scale timing parameters to reflect changes due to process variations.

If the value is not a constant, the PROC_MULT_*modeltype* value is a function of the PROC_VAR property which specifies the process used during a particular analysis with a timing tool. Table 5-1 on page 121 shows the PROC_MULT properties for the different timing parameters. It also shows the default value or multiplier that is used if a particular property is not specified.

With PROC_MULT_*modeltype*, a single value will be used for both rising and falling signals. The RISE and FALL keywords can be used to specify different values for rising and falling signals.

Arguments

modeltype

```
CAPACITANCE | CSPOWER | DCURRENT | GNDC | HOLD
| IENERGY | MPWH | MPWL | INSERTION_DELAY |
NET_CAP | NET_RES |
NO_CHANGE | PERIOD | PROPAGATION | PSPOWER|
RECOVERY | REMOVAL | SENERGY | SETUP |
SKEW | SUBSTRC | SUPPC | TENERGY |
TRANSITION | WAVEFORM_TAIL_RES
```

value

float | *min*::*max* | *min*:*typ*:*max* | *linear* | *modelName*

linear

Specifies a linear model. See Linear Model on page 45. Proc_Mult_modeltype statements can use linear models, but not table models.

modelName

Specifies the name of a table model. See Spline or Table Models on page 46. Only the PROC_MULT statement can use a table model.

Examples

```
Proc_Mult_Propagation(Rise(1.0) Fall(0.9))
Proc_Mult_Transition(Rise(procRiseMod) Fall(procFallMod))
Proc_Mult(procMultModel)
```

Table 5-1 Proc_Mult Properties

Property	Multiplier Usage	Default
Proc_Mult	Global or default multiplier	1
Proc_Mult_Capacitance	Pin capacitance	Proc_Mult
Proc_Mult_CSPower	Scales pin static power	Proc_Mult
Proc_Mult_DCurrent	Scales drive current of the pad pin	Proc_Mult
Proc_Mult_GNDC	Scales the parameter for ground current	Proc_Mult
Proc_Mult_Hold	Hold timing check margins	Proc_Mult
Proc_Mult_IENERGY	Scales the parameter for internal energy	Proc_Mult

Table 5-1 Proc_Mult Properties, *continued*

Property	Multiplier Usage	Default
Proc_Mult_Insertion_Delay	Scales the values for insertion_delay_min & insertion_delay_max properties at pin level	Proc_Mult
Proc_Mult_MPWH	Minimum pulse width high timing check margins	Proc_Mult
Proc_Mult_MPWL	Minimum pulse width low timing check margins	Proc_Mult
Proc_Mult_Net_Cap	Interconnect capacitance	Proc_Mult
Proc_Mult_Net_Res	Interconnect resistance	Proc_Mult
Proc_Mult_No_Change	Multiplier/Model to scale setup and hold data within NO_CHANGE timing checks.	Proc_Mult
Proc_Mult_Period	Period timing check margins	Proc_Mult
Proc_Mult_Propagation	Cell path delays	Proc_Mult
Proc_Mult_PSPower	Scales cell static power	Proc_Mult
Proc_Mult_Recovery	Recovery timing check margins	Proc_Mult
Proc_Mult_Removal	Removal timing check margins	Proc_Mult
Proc_Mult_SENERGY	Scales the parameter for temporary short circuit energy	Proc_Mult
Proc_Mult_Setup	Setup timing check margins	Proc_Mult
Proc_Mult_Skew	Skew timing check margins	Proc_Mult
Proc_Mult_Substrc	Scales substrate current	Proc_Mult
Proc_Mult_SUPPC	Scales the parameter for supply current	Proc_Mult
Proc_Mult_TENERGYP	Scales the parameters for total energy	Proc_Mult
Proc_Mult_Transition	Cell output transitions	Proc_Mult

Table 5-1 Proc_Mult Properties, *continued*

Property	Multiplier Usage	Default
Proc_Mult_waveform_Tail_Res	Multiplier/Model to scale transient resistance.	Proc_Mult

Related Information

For more information on the usage of this property, see the “PVT Derating” chapter of the *Delay Calculation Algorithm Guide*.

PROC_VAR

Syntax

PROC_VAR(*value*)

Context

The PROC_VAR property can appear at the library level within the PROPERTIES statement.

Description

Specifies the reference points for process variation used for the library characterization.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Example

```
Proc_Var(0.9:1.0:1.1)
```

PROPERTIES

Syntax

```
PROPERTIES(
    { propertyName (propertyValue)
    | FOR_PIN(...)
    | WIRELOAD(...)
    | WIRELOAD_BY_AREA(...)
    | WIRELOAD_BY_CELL_COUNT(...)
    | WIRELOAD_BY_TRANSISTOR_COUNT(...)
    | WIRELOAD_BY_GATE_COUNT(...) }...
    )
```

Context

A PROPERTIES statement can appear at the library level only and must follow the usage MODEL statements.

Purpose

The PROPERTIES statement defines properties specific for the library. Each lower level property overrides any identical property specified at a higher level.

The properties presented in this chapter can be specified at the library level within the PROPERTIES statement. Most of the properties can also be specified at lower levels as indicated in the **Context** for the property. When you specify the property within a cell, path, pin, or bus definition, use the property name without a PROPERTIES statement as shown in the **Examples**.

Arguments

propertyName

Identifies a property whose value you want to specify. For an index of all properties, see Appendix D, “TLF Property Index.”

propertyValue

Specifies the property value. The value depends on the property.

FOR_PIN

Assigns common properties to all pins of a given pin type.

WIRELOAD

Assigns interconnect properties to a particular design group.

WIRELOAD_BY_XXX

Assigns interconnect properties to a particular design group.

Examples

The following PROPERTIES section specifies

- Interconnect properties for default use and for a special interconnect type
- Different warning and error values for slew limits for input and output pins
- Default input slew rates

```
Header (Library("TableTlf"))
Model(gateCap       (Spline (axis 2 7) (0.06 0.21)))
Model(gateRes       (Spline (axis 2 7) (0.0022 0.0075)))
Model(gateCap100    (Spline (axis 2 20) (0.24 0.60)))
Model(gateRes100    (Spline (axis 2 20) (0.0086 0.0215)))
Properties(
    Net_Cap(gateCap)
    Net_Res(gateRes)
    Wireload("a100" Net_Cap(gateCap100) Net_Res(gateRes100))
    Slew_Limit(Warn(~) Error(~))
    For_Pin(output Slew_Limit(Warn(~) Error(~)))
    Default_Slew (Rise(2.000:3.000:4.000) Fall(1.000:2.000:3.000))
    )
```

Properties, such as slew limits set at the library level, can be overwritten at the cell and pin levels. The next cell level section specifies

- A process multiplier for the cell path delays
- A warning and error value for the load limit
- Cell-specific values for the warning and error values for slew limits

  Because the cell limit differs from the library limits, the cell-level value is used.

```
Cell(And2
    Proc_Mult_Propagation(1.5)
    For_Pin(input Slew_Limit(Warn(1.2) Error(2.4)))
    Load_Limit(Warn(40) Error(50))
    Pin(...)
)
```

At the pin level the input slew warning and error limits are further specified. The values specified at the pin level are used, if given. Otherwise, values are inherited from the next higher level.

```
Pin(A
    Pintype(input)
    Capacitance (0.0267)
    Slew_Limit(Warn(1.1) Error(2.3))
    )
```

Related Information

FOR PIN, WIRELOAD, WIRELOAD BY XXX

PVT_CONDS

Syntax

```
PVT_CONDS(
     opCondName
     VOLTAGE(...)
     PROC_VAR(...)
     TEMPERATURE(...)
     [TREE_TYPE(...)]
)
```

Context

A PVT_CONDS statement can appear at library level within the PROPERTIES statement. The PVT_CONDS statement is optional and more than one statement can appear in the library.

Purpose

Defines an operating condition and gives it a name.

Description

An operating condition defines a process, voltage, temperature (PVT) and an optional interconnect model environment. Multiple operating conditions can exist in the library and each one must have a unique name to identify it. If you define multiple operating conditions, you must use the DEFAULT_PVT_COND statement to specify which one to use as the default.

Application

A named operating condition statement allows the tools to access the PVT and the environment interconnect model by giving a single name.

Arguments

opCondName
: Specifies a unique identifier for the operating condition.

PROC_VAR
: Specifies the process variation multiplier.

VOLTAGE

Specifies the operating voltage.

TEMPERATURE

Specifies the temperature of the environment in which the circuit operates.

TREE_TYPE

Specifies an interconnect model.

Example

```
PVT_CONDS(
    mil_70
    PROC_VAR(2.3)
    VOLTAGE(5)
    TEMPERATURE(43)
    TREE_TYPE(best_case_tree)
    )
```

Related Information

DEFAULT_PVT_COND, PROC_VAR, TEMPERATURE, TREE_TYPE, VOLTAGE

RES_UNIT

Syntax

RES_UNIT(*resistanceUnit*)

Context

A RES_UNIT statement can appear in the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for resistance. The default is 1kohm.

Arguments

resistanceUnit

1ohm | 10ohm | 100ohm | 1kohm | 10 kohm | 100 kohm | 1mohm | *float*kohm

Examples

```
RES_UNIT(10kohm)
RES_UNIT(1024.5 kohm)
```

Related Information

UNIT

ROUTING_LAYER

Syntax

ROUTING_LAYER(*layerName* ...)

Context

A ROUTING_LAYER statement can appear at library level within the PROPERTIES statement. This statement is optional and can appear more than once at library level provided that each statement specifies a unique layer name. Repetition of layer names is not permitted.

Purpose

Specifies the names of routing layers. These layer names are used to specify routing properties at cell level.

Arguments

layerName

Specifies the unique name of a layer. More than one layer can be specified in a space-separated list.

Example

```
ROUTING_LAYER(metal_1 metal_2)
```

Related Information

MIN_POROSITY, ROUTING_PROPS

SLEW_DEGRADATION

Syntax

SLEW_DEGRADATION(*modelName*)

or

SLEW_DEGRADATION(RISE(*modelName*) FALL(*modelName*))

Context

A SLEW_DEGRADATION statement can appear at library level within the PROPERTIES statement. This statement is optional and can appear more than once at library level. Redefinition is allowed and the value is overwritten by the latest definition.

Purpose

Specifies the degradation in the slew of a signal due to wire resistance and capacitance from the output of a cell to the input of next cell. Separate degradation can be specified for rise and fall transitions. Table axis values are the slew at the output pin and delay across the wire.

Arguments

modelName

Specifies the name of a table model. See Spline or Table Models on page 46.

Example

```
SLEW_DEGRADATION(degWireTab)
```

Also see example for Slew Degradation on page 401.

SLEW_LIMIT

Syntax

```
SLEW_LIMIT{(value)
     |(RISE(value) FALL(value))
     |(WARN(value) ERROR(value))}
```

or

```
SLEW_LIMIT(
     WARN(RISE(value) FALL(value))
     ERROR(RISE(value) FALL(value))
     )
```

Context

The `Slew_Limit` property can appear at the library level within the `PROPERTIES` statement and at the `CELL` and `PIN` levels. Each lower level definition overwrites the value specified at the higher level.

Description

The `Slew_Limit` property specifies the limits of an input slew that can be applied to an input pin. You can specify separate values for rising and falling signals using the `RISE` and `FALL` keywords to separate limits beyond which an error or warning message should be generated. TLF provides the values of the construct for warning and error message when you use the `WARN` and `ERROR` statements.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Example

```
Slew_Limit(Warn(1.2) Error(2.5))
```

```
Slew_Limit(
     Warn(Rise(3.5) Fall(3.0))
     Error(Rise(6.5) Fall(5.8))
     )
```

SLEW_LOWER_THRESHOLD_PCT

Syntax

```
SLEW_LOWER_THRESHOLD_PCT{(value)|(RISE(value)FALL(value))}
value: float
```

Context

The SLEW_LOWER_THRESHOLD_PCT property can appear at the library level within the PROPERTIES statement and at the CELL levels.

Description

The SLEW_LOWER_THRESHOLD_PCT property defines the interpretation of the transition time values in the library. This property is similar to the TABLE_TRANSITION_START construct except that the values can be specified in percentage. The RISE and FALL constructs can be used to specify different values for rising and falling signals. If single value is specified then both RISE and FALL values are assumed to be same

Arguments

value

A floating number.

Example

```
Header(abc
        ...
        )
Properties(
        SLEW_LOWER_THRESHOLD_PCT(33)
        // Both RISE and FALL values are assumed to be 33
        SLEW_UPPER_THRESHOLD_PCT(66)
        // Both RISE and FALL values are assumed to be 66
        ...
        )
Cell(xyz
    SLEW_LOWER_THRESHOLD_PCT(RISE(42) FALL(42))
    SLEW_UPPER_THRESHOLD_PCT(RISE(58) FALL(58))
```

```
    ...
    )
```

Related Information

INTPUT_THRESHOLD_PCT, OUTPUT_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_OUTPUT_THRESHOLD

SLEW_MEASURE_LOWER_THRESHOLD_PCT

Syntax

```
SLEW_MEASURE_LOWER_THRESHOLD_PCT{(value)|(RISE(value)FALL(value))}
value: float
```

Context

The SLEW_MEASURE_LOWER_THRESHOLD_PCT property can appear at the library level within the PROPERTIES statement and at the CELL level.

Description

The SLEW_MEASURE_LOWER_THRESHOLD_PCT property defines how the library data was measured during SPICE characterization. This property is similar to the SLEW_LOWER_THRESHOLD_PCT property except that the values of the property represent the measurement point during SPICE characterization. If this construct is not specified in the library, the measurement point during SPICE characterization is assumed to be 40. If single value is specified then both RISE and FALL values are assumed to be same

Arguments

value

A floating number.

Example

```
Header(abc
        ...
        )
Properties(
        SLEW_MEASURE_LOWER_THRESHOLD_PCT(33)
        // Both RISE and FALL values are assumed to be 33
        SLEW_MEASURE_UPPER_THRESHOLD_PCT(66)
        // Both RISE and FALL values are assumed to be 66
        ...
        )
Cell(xyz
    SLEW_MEASURE_LOWER_THRESHOLD_PCT(RISE(42) FALL(42))
    SLEW_MEASURE_UPPER_THRESHOLD_PCT(RISE(58) FALL(58))
```

```
    ...
    )
```

Related Information

INTPUT_THRESHOLD_PCT, OUTPUT_THRESHOLD_PCT,
SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_OUTPUT_THRESHOLD

SLEW_MEASURE_UPPER_THRESHOLD_PCT

Syntax

SLEW_MEASURE_UPPER_THRESHOLD_PCT{(*value*)|(RISE(value)FALL(value))}

value: float

Context

The SLEW_MEASURE_UPPER_THRESHOLD_PCT property can appear at the library level within the PROPERTIES statement and at the CELL level.

Description

The SLEW_MEASURE_UPPER_THRESHOLD_PCT property defines how the library data was measured during SPICE characterization. This property is similar to the SLEW_UPPER_THRESHOLD_PCT property except that the values of the property represent the measurement point during SPICE characterization. If this construct is not specified in the library, the measurement point during SPICE characterization is assumed to be 60. If single value is specified then both RISE and FALL values are assumed to be same

Arguments

value

A floating number.

Example

```
Header(abc
        ...
        )
Properties(
        SLEW_MEASURE_LOWER_THRESHOLD_PCT(33)
        // Both RISE and FALL values are assumed to be 33
        SLEW_MEASURE_UPPER_THRESHOLD_PCT(66)
        // Both RISE and FALL values are assumed to be 66
        ...
        )
Cell(xyz
    SLEW_MEASURE_LOWER_THRESHOLD_PCT(RISE(42) FALL(42))
    SLEW_MEASURE_UPPER_THRESHOLD_PCT(RISE(58) FALL(58))
```

```
    ...
    )
```

Related Information

INTPUT_THRESHOLD_PCT, OUTPUT_THRESHOLD_PCT,
SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_OUTPUT_THRESHOLD

SLEW_MIN

Syntax

```
SLEW_MIN{(value)
     | (RISE(value) FALL(value))
     | (WARN(value) ERROR(value))
     }
```

Context

A SLEW_MIN statement can appear at the library level within the PROPERTIES statement and at the CELL, BUS and PIN levels.

Description

Specifies the minimum input slew for a pin. If it is defined at cell level, the value applies to all the pins in the cell. If it is defined at library level, the value applies to all the pins in all the cells.

You can specify separate values for rising and falling signals using the RISE and FALL keywords to separate limits beyond which an error or warning message should be generated. TLF provides the values of the construct for warning and error messages when you use the WARN and ERROR statements.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Example

```
SLEW_MIN(2.56)
SLEW_MIN(WARN(2.1) ERROR(1.9))
```

Related Information

SLEW_LIMIT

SLEW_UPPER_THRESHOLD_PCT

Syntax

```
SLEW_UPPER_THRESHOLD_PCT{(value)|(RISE(value)FALL(value))}
value: float
```

Context

The SLEW_UPPER_THRESHOLD_PCT property can appear at the library level within the PROPERTIES statement and at the CELL levels.

Description

The SLEW_UPPER_THRESHOLD_PCT property defines the interpretation of the transition time values in the library. This property is similar to the TABLE_TRANSITION_END construct except that the values can be specified in percentage. The RISE and FALL constructs can be used to specify different values for rising and falling signals. If single value is specified then both RISE and FALL values are assumed to be same

Arguments

value

A floating number.

Example

```
Header(abc
        ...
        )
Properties(
        SLEW_LOWER_THRESHOLD_PCT(33)
        // Both RISE and FALL values are assumed to be 33
        SLEW_UPPER_THRESHOLD_PCT(66)
        // Both RISE and FALL values are assumed to be 66
        ...
    )
Cell(xyz
    SLEW_LOWER_THRESHOLD_PCT(RISE(42) FALL(42))
    SLEW_UPPER_THRESHOLD_PCT(RISE(58) FALL(58))
```

```
    ...
    )
```

Related Information

INTPUT_THRESHOLD_PCT, OUTPUT_THRESHOLD_PCT,
SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_OUTPUT_THRESHOLD

TABLE_INPUT_THRESHOLD

Syntax

TABLE_INPUT_THRESHOLD(*value*)

or

TABLE_INPUT_THRESHOLD(RISE(*value*) FALL(*value*))

Context

The TABLE_INPUT_THRESHOLD property can appear at the library level within the PROPERTIES statement and at the CELL level.

Description

The TABLE_INPUT_THRESHOLD property specifies the point—as a fraction of the total voltage swing between the starting and ending values—where delay measurement begins. The RISE and FALL keywords can be used to specify different values for rising and falling signals. If single value is specified then both RISE and FALL values are assumed to be same. Users are encouraged to use newly introduced threshold parameters.

Arguments

value

A floating number. The default is 0.5.

Example

```
Header(abc
        ...
        )
Properties(
        TABLE_INPUT_THRESHOLD(0.5)
        //Both RISE and FALL values are assumed to be 0.5
        ...
        )
Cell(xyz
    TABLE_INPUT_THRESHOLD(RISE(0.42)FALL(0.58))
    ...
    )
```

Also see example for Mixed Threshold Setting on page 395.

Related Information

INPUT_THRESHOLD_PCT, OUTPUT_THRESHOLD_PCT,
SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_OUTPUT_THRESHOLD,
TABLE_TRANSITION_START, TABLE_TRANSITION_END

For more information on the usage of this property, see the “Cell Delays” chapter of the *Delay Calculation Algorithm Guide*.

TABLE_OUTPUT_THRESHOLD

Syntax

TABLE_OUTPUT_THRESHOLD(*value*)

or

TABLE_OUTPUT_THRESHOLD(RISE(*value*) FALL(*value*))

Context

The TABLE_OUTPUT_THRESHOLD property can appear at the library level within the PROPERTIES statement and at the CELL level.

Description

Specifies the point—as a fraction of the total voltage swing between the starting and ending values—where delay measurement ends.The RISE and FALL constructs can be used to specify different values for rising and falling signals. If single value is specified then both RISE and FALL values are assumed to be same. Users are encouraged to use newly introduced threshold parameters.

Arguments

value

A floating number. The default is 0.5.

Example

```
Header(abc
        ...
        )
Properties(
        Table_Output_Threshold(0.5000)
        // Both RISE and FALL values are assumed to be 0.5
        ...
        )
Cell(xyz
    TABLE_OUTPUT_THRESHOLD(RISE(0.42)FALL(0.58))
    ...
    )
```

Also see example for Mixed Threshold Setting on page 395.

Related Information

INPUT_THRESHOLD_PCT, OUTPUT_THRESHOLD_PCT,
SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_TRANSITION_START, TABLE_TRANSITION_END

For more information on the usage of this property, see the “Cell Delays” chapter of the *Delay Calculation Algorithm Guide*.

TABLE_TRANSITION_START

Syntax

TABLE_TRANSITION_START(*value*)

or

TABLE_TRANSITION_START{(*value*) |(RISE(*value*) FALL(*value*))}

value: float

Context

The TABLE_TRANSITION_START property can appear at the library level within the PROPERTIES statement and at the CELL levels.

Description

Specifies the point—as a fraction of the total voltage swing between the starting and ending values—where slew measurement begins. The RISE and FALL constructs can be used to specify different values for rising and falling signals. If single value is specified then both RISE and FALL values are assumed to be same.

Arguments

value

A floating number. The default is 0.1.

Example

```
TABLE_TRANSITION_START(0.2)
```

Related Information

INPUT_THRESHOLD_PCT, OUTPUT_THRESHOLD_PCT,
SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_OUTPUT_THRESHOLD, TABLE_TRANSITION_END

For more information on the usage of this property, see the “Cell Delays” chapter of the *Delay Calculation Algorithm Guide*.

TABLE_TRANSITION_END

Syntax

TABLE_TRANSITION_END(*value*)

or

TABLE_TRANSITION_END{(*value*) |(RISE(*value*) FALL(*value*))}

Context

The TABLE_TRANSITION_END property can appear at the library and CELL levels.

Description

Specifies the point—as a fraction of the total voltage swing between the starting and ending values—where slew measurement ends. The RISE and FALL constructs can be used to specify different values for rising and falling signals. If single value is specified then both RISE and FALL values are assumed to be same.

Arguments

value

A floating number. The default is 0.9.

Example

```
TABLE_TRANSITION_END(0.2)
```

Related Information

INPUT_THRESHOLD_PCT, OUTPUT_THRESHOLD_PCT,
SLEW_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_LOWER_THRESHOLD_PCT,
SLEW_MEASURE_UPPER_THRESHOLD_PCT,
SLEW_UPPER_THRESHOLD_PCT, TABLE_INPUT_THRESHOLD,
TABLE_OUTPUT_THRESHOLD, TABLE_TRANSITION_START

For more information on the usage of this property, see the “Cell Delays” chapter of the *Delay Calculation Algorithm Guide*.

TEMPERATURE

Syntax

TEMPERATURE(*value*)

Context

The TEMPERATURE property can appear at the library level within the PROPERTIES statement.

Description

Specifies the reference temperature conditions used for the library characterization. Units are specified by the TEMPERATURE_UNIT statement.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Example

```
Temperature(-10:15:40)
```

TEMPERATURE_UNIT

Syntax

TEMPERATURE_UNIT(*tempUnit)*

Context

A TEMPERATURE_UNIT statement can appear in the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for temperature using Celsius (1C) or Kelvin (1K). The default is 1C.

Arguments

tempUnit
1C | 1K

Example

```
TEMPERATURE_UNIT(1C)
```

Related Information

UNIT

TEMP_MULT

Syntax

TEMP_MULT (*value*)

or

TEMP_MULT_*modeltype* {(*value*) |(RISE(*value*) FALL(*value*))}

Context

The TEMP_MULT properties can appear at both the library level within the PROPERTIES statement and at the CELL level.

Description

The Temp_Mult property specifies a multiplier or model that can be used to scale timing parameters to reflect changes due to temperature variations.

If the value is not a constant, the Temp_Mult_*modeltype* value is a function of the TEMPERATURE property which specifies the temperature used during a particular analysis with a timing tool. Table 5-2 on page 153 shows the TEMP_MULT properties for the different timing parameters. The table also shows the default value or multiplier that is used if a particular property is not specified.

With TEMP_MULT_*modeltype*, a single value will be used for both rising and falling signals. The RISE and FALL keywords can be used to specify different values for rising and falling signals.

Arguments

modeltype

```
CAPACITANCE | CSPOWER | DCURRENT | GNDC |
HOLD | IENERGY | MPWH | MPWL | INSERTION_DELAY
| NET_CAP | NET_RES |
NO_CHANGE | PERIOD | PROPAGATION | PSPOWER |
RECOVERY | REMOVAL | SENERGY | SETUP |
SKEW | SUBSTRC | SUPPC | TENERGY |
TRANSITION | WAVEFORM_TAIL_RES
```

value

float | *min*::*max* | *min*:*typ*:*max* | Linear | *modelName*

Linear

Specifies a linear model. See Linear Model on page 3-18.

modelName

Specifies the name of a table model. See Spline or Table Models on page 3-19.

Examples

```
Temp_Mult_Transition(Rise(0.9) Fall(1.1))
Temp_Mult(tempMultModel)
```

Table 5-2 Temp_Mult Properties

Property	Multiplier Usage	Default
Temp_Mult	Global or default multiplier	1
Temp_Mult_Capacitance	Pin capacitance	Temp_Mult
Temp_Mult_CSPower	Scales pin static power	Temp_Mult
Temp_Mult_DCurrent	Scales drive current of the pad pin	Temp_Mult
Temp_Mult_GNDC	Scales the parameter for ground current	Temp_Mult
Temp_Mult_Hold	Hold timing check margins	Temp_Mult
Temp_Mult_IENERGY	Scales the parameter for internal energy	Temp_Mult
Temp_Mult_MPWH	Minimum pulse width high timing check margins	Temp_Mult
Temp_mult_MPWL	Minimum pulse width low timing check margins	Temp_Mult
Temp_Mult_Insertion_Delay	Scales the values for insertion_delay_min & insertion_delay_max properties at pin level	Temp_Mult

Table 5-2 Temp_Mult Properties, *continued*

Property	Multiplier Usage	Default
Temp_Mult_Net_Cap	Interconnect capacitance	Temp_Mult
Temp_Mult_Net_Res	Interconnect resistance	Temp_Mult
Temp_Mult_No_Change	Multiplier/Model to scale setup and hold data within NO_CHANGE timing checks.	Temp_Mult
Temp_Mult_Period	Period timing check margins	Temp_Mult
Temp_Mult_Propagation	Cell path delays	Temp_Mult
Temp_Mult_PSPower	Scales cell static power	Temp_Mult
Temp_Mult_Recovery	Recovery timing check margins	Temp_Mult
Temp_Mult_Removal	Removal timing check margins	Temp_Mult
Temp_Mult_SENERGY	Scales the parameter for temporary short circuit energy	Temp_Mult
Temp_Mult_Setup	Setup timing check margins	Temp_Mult
Temp_Mult_Skew	Skew timing check margins	Temp_Mult
Temp_Mult_Substrc	Scales substrate current	Temp_Mult
Temp_Mult_SUPPC	Scales the parameter for supply current	Temp_Mult
Temp_Mult_TENERGY	Scales the parameters for total energy	Temp_Mult
Temp_Mult_Transition	Cell output transitions	Temp_Mult
Temp_Mult_Waveform_Tail_Res	Multiplier/Model to scale transient resistance.	Temp_Mult

Related Information

For more information on the usage of this property, see the “PVT Derating” chapter of the *Delay Calculation Algorithm Guide*.

THRESHOLD_LIMIT

Syntax

```
THRESHOLD_LIMIT(value)
```

Context

A THRESHOLD_LIMIT statement can appear within the CELL, BUS and PIN statements.

Purpose

Specifies the limit on the threshold shift at an output pin.

Description

Over a period of operation, the cell performance degrades due to the hot electron effect which results in a threshold shift. The hot electron effect has been captured through the FLUENCE statement. Fluence can also be mapped to threshold shift and mobility degradation. The THRESHOLD_LIMIT statement specifies the limit on threshold shift while the MOBILITY_LIMIT statement specifies the limit on mobility degradation.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Example

```
THRESHOLD_LIMIT(0.25)
```

Related Information

FLUENCE

TIME_UNIT

Syntax

TIME_UNIT(*timeUnit*)

Context

A TIME_UNIT statement can appear within the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for time. The default is ns.

Arguments

timeUnit

1ns | 10ns | 1ps | 10ps ⊥ 100ps

Example

```
TIME_UNIT(10ps)
```

Related Information

UNIT

TRANSISTOR_COUNT

Syntax

TRANSISTOR_COUNT(*value*)

Context

The TRANSISTOR_COUNT property can appear at both the library level within the PROPERTIES statement and at CELL level.

Description

Assigns a transistor count to a cell. TRANSISTOR_COUNT can be used for two purposes:

- As a cost function during logic synthesis
- To compute a total area value during delay calculation that can then be used to select a net category

Arguments

value
integer

Example

```
Transistor_Count(4)
```

Related Information

WIRELOAD_BY_XXX

TREE_TYPE

Syntax

TREE_TYPE(*interconnectModelType*)

Context

A TREE_TYPE statement can appear as an option within a PVT_CONDS statement. Different PVT_CONDS statements can refer to different estimated interconnect topology. A PVT_CONDS statement can appear at library level within the PROPERTIES statement.

Purpose

Defines the environment interconnect model.

Description

Provides estimated interconnect topology information to synthesis and timing tools for the interconnect delay calculation. Three types of interconnect models are supported.

Arguments

interconnectModelType
: best_case_tree | worst_case_tree | balanced_tree

best_case_tree
: Models the load pin as physically adjacent to the driver. In this case, all the wire capacitance is incurred, but none of the wire resistance must be overcome.

worst_case_tree
: Models the load pin at the extreme end of the wire. In this case, each load pin incurs both the full wire capacitance and the full wire resistance.

balanced_tree
: Models all the load pins on separate, equal branches of the

interconnect wire. In this case, each load pin incurs an equal portion of the wire capacitance and resistance.

Example

```
TREE_TYPE(balanced_tree)
```

Related Information

DEFAULT_PVT_COND, PVT_CONDS

UNIT

Syntax

```
UNIT(
     AREA_UNIT(...)
     CAP_UNIT(...)
     CONDUCTANCE_UNIT(...)
     CURRENT_UNIT(...)
     INDUCTANCE_UNIT(...)
     RES_UNIT(...)
     TIME_UNIT(...)
     TEMPERATURE_UNIT(...)
     VOLT_UNIT(...)
     POWER_UNIT(...)
)
```

Context

A `UNIT` statement can appear at the library level within a `PROPERTIES` statement. This statement is optional and can appear only once at library level.

Description

Specifies unit multipliers for various parameters. If you do not explicitly define the units, you must use the default units when entering values in the TLF file. The default units are shown in Table 5-3 on page 160.

Arguments

The following table shows the default units for each of the arguments.

Table 5-3 Default Units

Argument	Parameter	Unit	Default
AREA_UNIT	area	square micrometer	1squ
CAP_UNIT	capacitance	picofarad	1pF
CONDUCTANCE_UNIT	conductance	millisiemen	1mS
CURRENT_UNIT	current	milliampere	1mA

Table 5-3 Default Units

Argument	Parameter	Unit	Default
INDUCTANCE_UNIT	inductance	picohenry	1pH
POWER_UNIT	power	milliwatt	1mW
RES_UNIT	resistance	kiloOhm	1kohm
TEMPERATURE_UNIT	temperature	degree Celsius	1C
TIME_UNIT	time	nanosecond	1ns
VOLT_UNIT	voltage	volt	1V

Derived Units

For data values other than mentioned above, units are derived using the above unit types. For example, the unit for energy is derived by multiplying units of power and time. The unit for fluence is derived by multiplying units of current and time. You do not need to enter a unit when specifying a parameter with derived units. However, you need to know the derived unit in order to enter the correct magnitude for the value.

Example

```
UNIT(
    AREA_UNIT(10squ)
    TIME_UNIT(1ps)
    )
```

Also see example for Units and Pad Modeling on page 390.

VDROP_LIMIT

Syntax

VDROP_LIMIT(*value*)

or

VDROP_LIMIT(WARN(*value*) ERROR(*value*))

Context

A VDROP_LIMIT statement can appear at the library level within the PROPERTIES statement, within a CELL statement, or within a PIN statement with PINTYPE set to SUPPLY or GROUND.

Description

Specifies the worst case limit on the voltage drop (IR drop) at a power pin. TLF provides the values of the construct for warning and error message when you use the WARN and ERROR statements.

Arguments

value
float | *min*::*max* | *min*:*typ*:*max*
Specifies the value of the voltage drop limit in units specified by VOLT_UNIT.

Example

```
CELL(...
    VDROP_LIMIT(0.25)
    )
CELL(...
    PIN(VDD
        PINTYPE(SUPPLY)
        VDROP_LIMIT(0.15)
        ...
        )
    PIN(VSS
        PINTYPE(GROUND)
        VDROP_LIMIT(0.12)
        ...
```

```
        )
    . . .
    )
```

VOLTAGE

Syntax

VOLTAGE(*value*)

Context

The VOLTAGE property can appear at the library level within a PROPERTIES statement.

Description

Specifies the reference voltage conditions used for the library characterization.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*

Provides the value of the operating voltage in units specified by VOLT_UNIT.

Example

```
Voltage(5.4:5.0:4.6)
```

VOLT_HIGH_THRESHOLD

Syntax

VOLT_HIGH_THRESHOLD(*voltThresholdValue*)

Context

A VOLT_HIGH_THRESHOLD statement can appear within the INPUT_VOLTAGE and OUTPUT_VOLTAGE statements. The INPUT_VOLTAGE and OUTPUT_VOLTAGE statements can appear at library level within the PROPERTIES statement and at PIN level.

Purpose

Specifies the high input or output threshold voltage for a pad.

Description

See the description of PAD_PROPS.

Arguments

voltThresholdValue
float | *min*::*max* | *min*:*typ*:*max*

Example

```
VOLT_HIGH_THRESHOLD(1.2)
```

Also see the example of PAD_PROPS.

Related Information

PAD_PROPS, VOLT_LOW_THRESHOLD, VOLT_MIN, VOLT_MAX

VOLT_LOW_THRESHOLD

Syntax

VOLT_LOW_THRESHOLD(*voltThresholdValue*)

Context

A VOLT_LOW_THRESHOLD statement can appear within the INPUT_VOLTAGE and OUTPUT_VOLTAGE statements. The INPUT_VOLTAGE and OUTPUT_VOLTAGE statements can appear at library level within the PROPERTIES statement and at PIN level.

Purpose

Specifies the low input or output threshold voltage for a pad.

Description

See the description of PAD_PROPS.

Arguments

voltThresholdValue
float | *min*::*max* | *min*:*typ*:*max*

Example

```
VOLT_LOW_THRESHOLD(1.2)
```

Also see the example of PAD_PROPS.

Related Information

PAD_PROPS, VOLT_HIGH_THRESHOLD, VOLT_MIN, VOLT_MAX

VOLT_MAX

Syntax

VOLT_MAX(*maxVoltage*)

Context

A VOLT_MAX statement can appear within the INPUT_VOLTAGE and OUTPUT_VOLTAGE statements.

Purpose

Specifies the maximum input or output threshold voltage for a pad.

Description

See the description of PAD_PROPS.

Arguments

maxVoltage

float | *min*::*max* | *min*:*typ*:*max*

Example

```
VOLT_MAX(1.2)
```

Also see the example of PAD_PROPS.

VOLT_MIN

Syntax

VOLT_MIN(*minVolt)*

Context

A VOLT_MIN statement can appear within the INPUT_VOLTAGE and OUTPUT_VOLTAGE statements. The INPUT_VOLTAGE and OUTPUT_VOLTAGE statements can appear at library level within the PROPERTIES statement and at PIN level.

Purpose

Specifies the minimum input or output threshold voltage for a pad.

Description

See the description of PAD_PROPS.

Arguments

minVolt

float | *min*::*max* | *min*:*typ*:*max*

Example

```
VOLT_MIN(1.2)
```

Also see the example of PAD_PROPS.

Related Information

PAD_PROPS, VOLT_HIGH_THRESHOLD, VOLT_LOW_THRESHOLD, VOLT_MAX

VOLT_MULT

Syntax

VOLT_MULT (*value*)

or

VOLT_MULT_*modeltype* {(*value*)|(RISE(*value*) FALL(*value*))}

Context

The VOLT_MULT properties can appear at the library level within the PROPERTIES statement and at the CELL level.

Description

Specifies a multiplier or model that can be used to scale timing parameters to reflect changes due to voltage variations.

If the value is not a constant, then the VOLT_MULT_*modeltype* value is a function of the Voltage property which specifies the voltage used during a particular analysis with a timing tool. Table 5-4 on page 170 shows the Volt_Mult properties for the different timing parameters. The table also shows the default value or multiplier that is used if a particular property is not specified.

With VOLT_MULT_*modeltype*, a single value will be used for both rising and falling signals. The RISE and FALL keywords can be used to specify different values for rising and falling signals.

Arguments

modeltype

```
CAPACITANCE | CSPOWER | DCURRENT | GNDC |
HOLD | IENERGY | MPWH | MPWL | INSERTION_DELAY
| NET_CAP | NET_RES |
NO_CHANGE | PERIOD | PROPAGATION | PSPOWER |
RECOVERY | REMOVAL | SENERGY | SETUP |
SKEW | SUBSTRC | SUPPC | TENERGY |
TRANSITION | WAVEFORM_TAIL_RES
```

value

float | *min*::*max* | *min*:*typ*:*max* | Linear | *modelName*

Linear

Specifies a linear model. See Linear Model on page 45. Proc_Mult_model statements can use linear models, but not table models.

modelName

Specifies the name of a table model. See Spline or Table Models on page 46. Only the Proc_Mult statement can use a table model.

Examples

```
Volt_Mult_Recovery(Rise(.9) Fall(1.0))
Volt_Mult(voltMultModel)
```

Table 5-4 Volt_Mult Properties

Property	Multiplier Usage	Default
Volt_Mult	Global or default multiplier	1
Volt_Mult_Capacitance	Pin capacitance	Volt_Mult
Volt_Mult_CSPower	Scales pin static power	Volt_Mult
Volt_Mult_DCurrent	Scales drive current of the pad pin	Volt_Mult
Volt_Mult_GNDC	Scales the parameter for ground current	Volt_Mult
Volt_Mult_Hold	Hold timing check margins	Volt_Mult
Volt_Mult_IENERGY	Scales the parameter for internal energy	Volt_Mult
Volt_Mult_MPWH	Minimum pulse width high timing check margins	Volt_Mult
Volt_Mult_MPWL	Minimum pulse width low timing check margins	Volt_Mult

Table 5-4 Volt_Mult Properties, *continued*

Property	Multiplier Usage	Default
Volt_Mult_Insertion_Delay	Scales the values for insertion_delay_min & insertion_delay_max properties at pin level	Volt_Mult
Volt_Mult_Net_Cap	Interconnect capacitance	Volt_Mult
Volt_Mult_Net_Res	Interconnect resistance	Volt_Mult
Volt_Mult_No_Change	Multiplier/Model to scale setup and hold data within NO_CHANGE timing checks.	Volt_Mult
Volt_Mult_Period	Period timing check margins	Volt_Mult
Volt_Mult_Propagation	Cell path delays	Volt_Mult
Volt_Mult_PSPower	Scales cell static power	Volt_Mult
Volt_Mult_Recovery	Recovery timing check margins	Volt_Mult
Volt_Mult_Removal	Removal timing check margins	Volt_Mult
Volt_Mult_SENERGY	Scales the parameter for temporary short circuit energy	Volt_Mult
Volt_Mult_Setup	Setup timing check margins	Volt_Mult
Volt_Mult_Skew	Skew timing check margins	Volt_Mult
Volt_Mult_Substrc	Scales substrate current	Volt_Mult
Volt_Mult_SUPPC	Scales the parameter for supply current	Volt_Mult
Volt_Mult_TENERGY	Scales the parameters for total energy	Volt_Mult
Volt_Mult_Transition	Cell output transitions	Volt_Mult
Volt_Mult_Waveform_Tail_Res	Multiplier/Model to scale transient resistance.	Volt_Mult

Related Information

For more information on the usage of this property, see the “PVT Derating” chapter of the *Delay Calculation Algorithm Guide*.

VOLT_UNIT

Syntax

VOLT_UNIT(*voltUnit*)

Context

A VOLT_UNIT statement can appear in the UNIT statement. The UNIT statement can appear at the library level within the PROPERTIES statement.

Description

Specifies the units for voltage. The default is 1V.

Arguments

voltUnit

1mV | 10mV | 100mV | 1V

Example

```
VOLT_UNIT(10mV)
```

Related Information

UNIT

WAVEFORM_TAIL_RES

Syntax

```
WAVEFORM_TAIL_RES {(value) | ( RISE(value) FALL(value)) }

value : Model
```

Note: Only `Const` Model and `TRANSIENT_RES_Model` are supported.

Context

A `WAVEFORM_TAIL_RES` property can appear within the `CELL`, output `PIN`, or `PATH` statements. The `PATH` and `PIN` level specification overrides `CELL` level specification. If the `WAVEFORM_TAIL_RES` statement appears within `CELL` statement, the value is used for all paths and output pins that have missing path or pin level specification. `RES_UNIT` value is the unit for `WAVEFORM_TAIL_RES`.

Description

A `WAVEFORM_TAIL_RES` property specifies the transient driver resistance used by the delay calculation tools.

Example

```
Cell(xyz
    // Cell level specification with single resistance value for both
    Rise and fall
    WAVEFORM_TAIL_RES(Const(0.16:0.18:0.2))
    TRANSIENT_RES_MODEL(model1
    (Const( 0.2)
    )
    )


TRANSIENT_RES_Model (model2
                     (Const( 0.4)
                     )
                     )
...
// Cell level specification for Rise and Fall
Waveform_Tail_Res (Rise(model1) Fall(model2))
...
```

```
Pin(Z Pintype(Output) ...)
Pin(A Pintype(Input) ... )
// Note that as output transition is specified in the PATH statement,
there is no need to use Rise/Fall construct
PATH(A => Z 10 01 DELAY(...) SLEW(...) WAVEFORM_TAIL_RES(model1) )
PATH(A => Z 01 10 DELAY(...) SLEW(...) WAVEFORM_TAIL_RES(model2) )
...
)
```

Related Information

CT_RES_HIGH, CT_RES_LOW, usage_MODEL

WIRE_DELAY

Syntax

```
WIRE_DELAY {(value) | ( RISE(value) FALL(value)) }
value : Timing_Model
```

Context

The WIRE_DELAY property can appear at the LIBRARY level within the PROPERTIES and WIRELOAD statements.

Description

A WIRE_DELAY property is used to model 2-D wire delay. *Timing_Model* are 2D tables supporting output_slew_axis and rc_product_axis axis types.

Example

```
Header (
        Library(“library_using_2D_wire_delay_tables”)
        TLF_Version(“4.3”)
        )
Timing_Model(wire100_rise
            (Spline
                (output_slew_axis 0.0 5.0 20.0)
                (rc_product_axis 0.2 0.4 1.2)
            data(
                (0.2 0.4 1.2)
                (2.1 2.8 3.2)
                (3.4 3.9 4.5)
                )
            )
            )
Timing_Model(wire100_fall
            (Spline
                (output_slew_axis 0.0 5.0 20.0)
                (rc_product_axis 0.2 0.4 1.2)
                data(
                    (0.2 0.4 1.2)
                    (2.2 2.8 3.0)
                    (3.0 3.9 4.0)
                    )
```

```
            )
            )
Properties(
          WireLoad( Rise(wire100_rise) Fall(wire100_fall) )
          )
```

Related Statements

usage Model

WIRELOAD

Syntax

```
WIRELOAD(wireload_name
    NET_CAP(model_or_value)
    NET_RES(model_or_value)
    )
```

Context

A WIRELOAD statement can appear at library level within the PROPERTIES statement.

Description

The WIRELOAD statement specifies the wireload model to which the specified interconnect properties apply.

wireload_name
: Specifies the wireload model name. Use an identifier or string.

NET_CAP
: Specifies the interconnect capacitance.

NET_RES
: Specifies the interconnect resistance.

Example

```
Model(netCap      (Spline (axis 2 7) (0.06 0.21)))
Model(netRes      (Spline (axis 2 7) (0.0022 0.0075)))
Model(netCap500K  (Spline (axis 2 20) (0.24 0.60)))
Model(netRes500K  (Spline (axis 2 20) (0.0086 0.0215)))
Properties(
    Net_Cap(netCap) Net_Res(netRes) // default
    Wireload(CMOS500K    Net_Cap(netCap500K)
                         Net_Res(netRes500K)
    )
    )
```

Related Information

WIRELOAD_BY_XXX

WIRELOAD_BY_*XXX*

Syntax

```
WIRELOAD_BY_XXX(
    wireload_group_name
    (
        [wireload_name] limit
        NET_CAP(model_or_value)
        NET_RES(model_or_value)
    )...
    )
```

where *XXX* is either AREA, CELL_COUNT, GATE_COUNT, or TRANSISTOR_COUNT.

Context

The WIRELOAD_BY_AREA, WIRELOAD_BY_CELL_COUNT, WIRELOAD_BY_GATE_COUNT, and WIRELOAD_BY_TRANSISTOR_COUNT statements can appear at the library level within the PROPERTIES statement.

Description

These statements group wireload models. This grouping permits wireload selection based on an area parameter. You can parameterize a wireload model in terms of

- Square microns
- Transistor count
- (Logic) gate count
- Cell count

The wireload model maps the number of connections (fanout + 1 for a net with a single driver) on a net to the resistance and capacitance for that wire.

wireload_group_name
: Specifies the name of a group of wireload models.

wireload_name
: Specifies an optional wireload model name. Use a string, that is, a sequence of characters surrounded by double-quotes (").

Wireload model names are used to support applications that do not support parameterized wireload models.

limit

Specifies an area parameter limit. Use a floating number. The value indicates the maximum parameter value for which the wireload model is valid. The wireload model selected will be the smallest one with a limit that is greater than the block under consideration. If the block is larger than the largest parameter value, then the wireload model with the largest parameter value in the group is used.

NET_CAP

Specifies the interconnect capacitance.

NET_RES

Specifies the interconnect resistance.

Example

```
Wireload_By_Area("cmos5"
    (100 Net_Res(...) Net_Cap(...))
    (200 Net_Res(...) Net_Cap(...))
    (400 Net_Res(...) Net_Cap(...))
    (1000 Net_Res(...) Net_Cap(...))
    (10000 Net_Res(...) Net_Cap(...))
)
```

Also see the example for Wireload and Synthesis Constructs on page 407.

Related Information

WIRELOAD

6

TLF Statements

For an index of all properties and statements described in this manual, see Appendix D, “TLF Property Index.”

This chapter contains the following information:

Introduction

This chapter lists alphabetically the statements used in a Cadence® timing library format (TLF) file, except those representing properties which are described in Chapter 5, "Properties." A statement begins with a keyword followed by additional information within parentheses. Keywords, like the TLF language, are not case sensitive.

For a complete list of the statements described in this manual, see TLF Property Index on page 448.

For each statement, you are given

- A syntax description

  For a description of the syntax conventions, see the Typographic and Syntax Conventions (see Preface-12). Refer also to the special conventions listed below.

- A context section that describes where the statement can appear in other statements

  Most statements can appear in multiple places in a TLF file. For example, `SLEW_LIMIT` can appear at the library scope within a `PROPERTIES` statement and as a keyword within `CELL` and `PIN` statements.

- A short description
- A description of the arguments or possible values for variables
- Checks (optional)
- An example
- Related statements (optional)

Special Conventions

Within syntax descriptions, three dots inside parentheses (…) indicate that detailed information was omitted because it is described elsewhere. For example, the contents of the `EQ_PINS` option in the `CELL` statement (illustrated below) is not shown in the description of

CELL on page 194, because it is described in EQ_PINS on page 225. Hypertext links in the description allow you to navigate to the related statements.

```
CELL
    [EQ_PINS(...)]
```

In examples, three dots inside parentheses (`...`) or three dots preceded by a blank indicate that data or values were omitted because they are not relevant to the example. In the example for the `Setup` statement below, the model information following `posEdge` is left out, because it is not relevant to this example.

```
Setup   (D => CLK SDF_Cond_End(clr_or_set)
        Cond_End(CLR || SET) 01 posEdge ...)
```

ADDRESS_BUS

Syntax

ADDRESS_BUS(*adrBusName*)

Context

An ADDRESS_BUS statement can appear only within the MEMORY_BUS statement. The MEMORY_BUS statement can appear at the CELL level.

Description

Specifies the name of the address bus to use for the operation on the data bus which has been specified in the same BUS statement.

Arguments

adrBusName

Provides the address bus name as specified by the BUS statement at CELL level.

Checks

See the checks for MEMORY_BUS.

Example

```
ADDRESS_BUS(read_adr)
```

Related Statements

BUS, MEMORY_BUS

ADDRESS_WIDTH

Syntax

ADDRESS_WIDTH(*value*)

Context

An ADDRESS_WIDTH statement can appear only within the MEMORY_PROPS statement. The MEMORY_PROPS statement can appear at the CELL level.

Description

Defines the width of an address bus within a memory cell. See the description for MEMORY_PROPS.

Arguments

value

Specifies the number of bits in the address bus. Must be an integer.

Checks

See the checks for MEMORY_PROPS.

Example

```
ADDRESS_WIDTH(8)
```

Related Statements

BUS, MEMORY_BUS, MEMORY_PROPS

AVAILABLE_TRACK

Syntax

AVAILABLE_TRACK(*value*)

Context

An AVAILABLE_TRACK statement can appear only within the ROUTING_PROPS statement. The ROUTING_PROPS statement can appear within the CELL level.

Description

Specifies the tracks available for routing on a layer and for a cell.

Arguments

value

Specifies the number of tracks available for routing. The value must be zero or a positive integer.

Example

```
AVAILABLE_TRACK(5)
```

Also see example for Routing.

Related Statements

MIN_POROSITY, ROUTING_LAYER, ROUTING_PROPS

BUS

Syntax

```
BUS (busName[from:to]
    BUSTYPE(input|output|bidir|internal)
    pinStmt
    PIN(...)
    )
```

Context

A BUS statement can appear at the CELL level.

Description

Defines a bus. A bus is a logical collection of pins. Individual pins of the bus can be referenced by indexing. A collection of pins can also be defined using the PIN statement, but it is not treated as a bus in the design.

Arguments

busName
: Specifies the bus name.

from
: Provides an integer for the starting bit (LSB) of the bit range.

to
: Provides an integer for the ending bit (MSB) of the bit range.

BUSTYPE input | output | bidir | internal
: Specifies the type of bus as either `input`, `output`, `bidirectional` or a collection of `internal` nodes.

pinStmt
: Refers to all the statements that can appear at the PIN level except PINTYPE.

`PIN`

Overrides properties that were defined for the bus for a single pin or group of pins. Groups of pins are specified by a bit range.

Example

```
BUS (adrBus[7:0]
    BUSTYPE(input)
    CAPACITANCE(0.06)
    PIN(adrBus[3] CAPACITANCE(0.07))
    PIN(adrBus[4:5] CAPACITANCE(0.065))
    )
```

BUSMODE

Syntax

```
BUSMODE(READ | WRITE | READWRITE)
```

Context

A BUSMODE statement can appear within the MEMORY_BUS statement. The MEMORY_BUS statement can appear at the CELL level.

Description

Describes the type of data bus. See the description of MEMORY_BUS.

Arguments

READ | Write | READWRITE
: Specifies a data bus that is either readable or writeable or specifies a bidirectional data bus that is readable and writeable depending on the operation.

Checks

See the checks for MEMORY_BUS.

Example

```
BUSMODE(READ)
```

Related Statements

MEMORY_BUS, MEMORY_PROPS

BUSTYPE

Syntax

```
BUSTYPE(input | output | bidir | internal)
```

Context

A BUSTYPE statement can appear within a BUS statement. A BUS statement can appear at the CELL level.

Description

Specifies the direction or type of bus.

Arguments

`input | output | bidir | internal`
: Specifies the type of bus as either `input`, `output`, `bidirectional` or a collection of `internal` nodes.

Example

```
BUSTYPE(bidir)
```

Related Statement

BUS

CELL

Syntax

```
CELL(cellName
    {PAD_CELL | IGNORE_CELL}
    [usage_MODEL(...)]...
    [Properties]
    PIN(...)...
    [EQ_PINS(...)]
    [REGISTER(...) | LATCH(...) | TEST_LATCH(...)]
    [PATH(...)]...
    [PATH_EXTENSION(...)]...
    [timing_check]...
    )
```

Context

A CELL statement can appear after the library level PROPERTIES statement or after another CELL statement.

Description

Contains all of the information specific to the cell being described. Specify one CELL statement for each cell in a library.

Arguments

cellName

Specifies the name of the cell that you are describing. Use an identifier.

PAD_CELL

Defines the cell as an I/O cell.

usage_MODEL

Defines the internal timing and power behavior of one or more cells or the timing behavior of an interconnect.

Properties

Specifies the properties specific to the cell being described.

For more information on cell level properties, refer to Chapter 5, "Properties."

PIN

Describes a cell pin.

EQ_PINS

Lists all pins that are considered equivalent in this cell.

REGISTER, LATCH

Describe the behavior of sequential cells.

PATH, PATH_EXTENSION

Contain the information needed to calculate the delay between two specified pins of the cell.

timing_check SETUP | HOLD | NO_CHANGE | RECOVERY | REMOVAL | MPWH | MPWL | PERIOD | SKEW

Contains the information needed to calculate the timing check.

Example

```
Cell(Xor
    Model(A_X_B_01 (...))
    Model(A_X_B_10 (...))
    Model(A_X_notB_01 (...))
    Model(A_X_notB_10 (...))
    Model(...)...
    (...) //cell properties
    Pin (A Pintype(input) (Capacitance(0.1346)))
    Pin(B Pintype(input) (Capacitance(0.1346)))
    Pin(X Pintype(output) Function(A ^ B))
    Path(A => X Cond(!B) SDF_Cond(B==0) 01 01 Delay(A_X_notB_01)
         Slew(...))
    Path(A => X Cond(!B) SDF_Cond(B==0) 10 10 Delay(A_X_notB_10)
         Slew(...))
    Path(A => X Cond(B) SDF_Cond(B==1) 10 01 Delay(A_X_B_10)
        Slew(...))
    Path(A => X Cond(B) SDF_Cond(B==1) 01 10 Delay(A_X_B_01)
        Slew(...))
    Path(B => X Cond(!A) SDF_Cond(A==0) 01 01 Delay(...) Slew(...))
    Path(B => X Cond(!A) SDF_Cond(A==0) 10 10 Delay(...) Slew(...))
    Path(B => X Cond(A) SDF_Cond(A==1) 10 01 Delay(...) Slew(...))
```

```
Path(B => X Cond(A) SDF_Cond(A==1) 01 10 Delay(...) Slew(...))
)
```

CLEAR

Syntax

`CLEAR(`*`clear_condition`*`)`

Context

A `CLEAR` statement can appear in a `LATCH` or `REGISTER` statement.

Description

Describes when a register or latch output is set to low, asynchronously.

Arguments

`clear_condition`

Describes the pin conditions leading to an asynchronous clear. Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Example

```
Clear(Clr)
```

Related Statement

SET

CLEAR_PRESET_VAR1

Syntax

CLEAR_PRESET_VAR1(*value*)

Context

A CLEAR_PRESET_VAR1 statement can appear within a REGISTER, TEST_REGISTER, LATCH, or TEST_LATCH statement. All of these statements can appear at the CELL level.

Description

Specifies a value for an OUTPUT pin within a REGISTER, TEST_REGISTER, LATCH, or TEST_LATCH statement when CLEAR and SET are both active at the same time.

Arguments

value

0 | 1 | N | T | X
Specifies one of the given values for an OUTPUT pin.

Example

```
CELL(seq_cell
        ...
        REGISTER(
            OUTPUT(Q)
            INPUT(D && !HOLD || Q && HOLD)
            CLOCK(CLK)
            SET(SET)
            CLEAR(CLR)
            CLEAR_PRESET_VAR1(0)
        )
        ....
    )
```

Related Statement

CLEAR_PRESET_VAR2

CLEAR_PRESET_VAR2

Syntax

CLEAR_PRESET_VAR2(*value*)

Context

A CLEAR_PRESET_VAR2 statement can appear within a REGISTER, TEST_REGISTER, LATCH, or TEST_LATCH statement. All of these statements can appear at the CELL level.

Description

Specifies a value for an INVERTED_OUTPUT pin within a REGISTER, TEST_REGISTER, LATCH, or TEST_LATCH statement when CLEAR and SET are both active at the same time.

Arguments

value

0 | 1 | N | T | X
Specifies one of the given values for an INVERTED_OUTPUT pin.

Example

```
CELL(seq_cell
        ...
        REGISTER(
            INVERTED_OUTPUT(Q)
            INPUT(D && !HOLD || Q && HOLD)
            CLOCK(CLK)
            SET(SET)
            CLEAR(CLR)
            CLEAR_PRESET_VAR2(T)
        )
        ....
    )
```

Related Statements

CLEAR_PRESET_VAR1

CLOCK

Syntax

CLOCK(*clock_condition*)

Context

A CLOCK statement can appear in a MEMORY_BUS, LATCH, or REGISTER statement. All of these statements can appear at the CELL level.

Description

Describes the signal that specifies when the clock or latch enable is active. The signal is used for the read and write operation on the bus.

Arguments

clock_condition

Describes the input pin signals for clock or latch enable. Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Checks

See the checks for MEMORY_BUS.

Example

```
CLOCK(CLK && WE)
CLOCK(EN && !HOLD)
```

Related Statements

CLOCK_PIN, PIN, SLAVE_CLOCK

CLOCK_PIN

Syntax

```
CLOCK_PIN
```

Context

A CLOCK_PIN statement can appear within a PIN statement.

Description

Specifies that the pin is a clock pin.

Example

```
PIN(CLK ...
    CLOCK_PIN
    ...
    )
```

Related Statements

PIN, PINTYPE

COND

Syntax

COND(*cond_exp*)

Context

A COND statement can appear in a PATH statement, PATH_EXTENSION statement, or within a timing_check. All of these statements can appear at the CELL level.

Description

A COND statement conditionalizes a delay path or timing check. The expression must be true for the path or timing check to be evaluated. For timing checks that involve two ports (such as SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE), this statement conditionalizes both ports. This statement is used by tools that use TLF without standard delay format (SDF), and that have no access to local signals in the simulation model annotated by SDF.

Note: You should specify an SDF_COND statement with a COND statement to add conditions to the SDF file that is generated. Never use a COND statement with the COND_START and COND_END statements.

Arguments

cond_exp

Describes the condition for a path or timing check. Use an expression of type *conditional_expr*. The expression can reference only the identifiers that are pins of the cells. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Examples

```
Path(A => Z Cond(B) SDF_Cond(B==1'b1) 01 10
        Delay(...) Slew(...))
Setup(D => CLK SDF_Cond(clr_or_set)
        Cond(CLR||SET) 01 posEdge setupModel)
```

COND_END

Syntax

COND_END(*cond_exp*)

Context

A COND_END statement can appear within a SETUP, HOLD, RECOVERY, REMOVAL, SKEW, or NO_CHANGE statement. All of these statements can appear as a timing_check at the CELL level.

Description

Conditionalizes the second edge of a timing check. The condition must be true for the timing check to be evaluated. This statement is used by tools that use TLF without SDF, and that have no access to local signals in the simulation model annotated by SDF.

Note: You should specify an SDF_COND_END statement with a COND_END statement to add conditions to the SDF file that is generated. You can combine a COND_START statement with a COND_END statement to specify separate conditions for the starting and ending edges of a timing check. However, never use a COND statement with the COND_END statement.

Arguments

cond_exp

Describes the condition that must be true for the second edge of a timing check. Use an expression of type *conditional_expr*. The expression can only reference identifiers that are pins of the cells. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Example

```
Setup(D => CLK SDF_Cond_End(clr_or_set)
         Cond_End(CLR||SET) 01 posEdge setupModel)
```

COND_START

Syntax

COND_START(*cond_exp*)

Context

A COND_START statement can appear in a SETUP, HOLD, RECOVERY, REMOVAL, SKEW, or NO_CHANGE statement. All of these statements can appear as a timing_check at the CELL level.

Description

Conditionalizes the first edge of a timing check. The expression must be true for the timing check to be evaluated. This statement is used by tools that use TLF without SDF, and that have no access to local signals in the simulation model annotated by SDF.

Note: You should specify an SDF_COND_START statement with a COND_START statement to add conditions to the SDF file that is generated. You can combine a COND_START statement with a COND_END statement to specify separate conditions for the starting and ending edges of a timing check. However, never use a COND statement with the COND_START statement.

Arguments

cond_exp

Describes the condition to be true for the first edge of a timing check. Use an expression of type *conditional_expr*. The expression can only reference identifiers that are pins of the cells. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Example

```
Setup(D => CLK SDF_Cond_Start(clr_or_set)
         Cond_Start (CLR||SET) 01 posEdge setupModel)
```

CSAT

Syntax

```
CSAT{(value) | ( RISE(value) FALL(value) )}
```

Context

A CSAT statement can appear within a PAD_PROPS statement. A PAD_PROPS statement can appear within a BUS or PIN statement.

Description

Specifies the current slope after threshold (CSAT) for rise and fall transitions. For additional information, see the description of PAD_PROPS.

A single value is used for both rising and falling signals. The RISE and FALL statements can be used to specify different values for rising and falling signals.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Specifies the property value for the rising and falling signals as *float*, *min::max*, or *min:typ:max*.

Example

```
CSAT( RISE(.3) FALL(.4))
```

Related Statements

CTTAT, CTTBT, CSBT, PAD_PROPS

CSBT

Syntax

CSBT{ (*value*) | (RISE(*value*) FALL(*value*)) }

Context

A CSBT statement can appear within a PAD_PROPS statement. A PAD_PROPS statement can appear within a BUS or PIN statement.

Description

Specifies the current slope before threshold (CSBT) for rise and fall transitions. See the description of PAD_PROPS for additional information.

A single value will be used for both rising and falling signals. The RISE and FALL statements can be used to specify different values for rising and falling signals.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Specifies the property value for the rising and falling signals as *float*, *min::max*, or *min:typ:max*.

Example

```
CSBT( RISE(.3) FALL(.4))
```

Related Statements

CSAT, CTTAT, CTTBT, PAD_PROPS

CTTAT

Syntax

CTTAT{(*value*) | (RISE(*value*) FALL(*value*))}

Context

A CTTAT statement can appear within a PAD_PROPS statement. A PAD_PROPS statement can appear within a BUS or PIN statement.

Description

Specifies the current transition time after threshold (CTTAT) for rise and fall transitions. See the description of PAD_PROPS for additional information.

A single value will be used for both rising and falling signals. The RISE and FALL statements can be used to specify different values for rising and falling signals.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Specifies the property value for the rising and falling signals as *float*, *min::max*, or *min:typ:max*.

Example

```
CTTAT( RISE(.3) FALL(.4))
```

Related Statements

CSAT, CSBT, CTTBT, PAD_PROPS

CTTBT

Syntax

CTTBT{(*value*) | (RISE(*value*) FALL(*value*))}

Context

A CTTBT statement can appear within a PAD_PROPS statement. A PAD_PROPS statement can appear within a BUS or PIN statement.

Description

Specifies the current transition time before threshold (CTTBT) for rise and fall transitions. See the description of PAD_PROPS for additional information.

A single value will be used for both rising and falling signals. The RISE and FALL statements can be used to specify different values for rising and falling signals.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Specifies the property value for the rising and falling signals as *float*, *min::max*, or *min:typ:max*.

Example

```
CTTBT( RISE(.3) FALL(.4))
```

Related Statements

CSAT, CSBT, CTTAT, PAD PROPS

DATA_WIDTH

Syntax

DATA_WIDTH(*value*)

Context

A DATA_WIDTH statement can appear within the MEMORY_PROPS statement. The MEMORY_PROPS statement can appear at the CELL level.

Description

Defines the width of the data bus within a memory cell, or it represents the word width within memory. See the description of MEMORY_PROPS for additional information.

Arguments

value

Specifies the number of bits in the data bus. Must be an integer.

Checks

See the checks for MEMORY_PROPS.

Example

```
DATA_WIDTH(32)
```

Related Statements

BUS, MEMORY_BUS, MEMORY_PROPS

DATE

Syntax

DATE(*date*)

Context

A DATE statement can appear in the HEADER statement.

Description

Specifies the date on which the TLF file was created. This statement is for reference purposes only.

Arguments

date

Specifies the TLF creation date. Use a string — a sequence of characters surrounded by double-quotes (") — as shown in the example.

Example

```
Header(
     Library("cmos500k")
     Date("4/20/97")
     )
```

DCURRENT

Syntax

DCURRENT(*value*)

Context

A DCURRENT statement can appear within the PAD_PROPS statement. A PAD_PROPS statement occurs within BUS and PIN statements.

Description

Specifies the value of drive current for a pad. Units are specified by the CURRENT_UNIT statement. See the description of PAD_PROPS for additional information.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Specifies the property value for the drive current as *float*, *min::max*, or *min:typ:max.*

Example

```
DCURRENT(2.7)
```

Related Statements

PAD_PROPS

DEFINE_ATTRIBUTE

Syntax

```
DEFINE_ATTRIBUTE(attribute_name (scope) (type))
scope : LIBRARY | CELL | PIN
type : BOOLEAN | FLOAT | STRING
```

Context

A DEFINE_ATTRIBUTE statement can appear within the LIBRARY and the CELL statement.

Description

The DEFINE_ATTRIBUTE statement is used to define user defined attributes. This should be specified for an attribute before it is referenced.

Arguments

attribute_name

Specifies the name of the attribute. The name of the attribute is case insensitive.

Example

```
Header (
        Library(“user_defined_attributes”)
        TLF_Version(“4.3”)
        )
    DEFINE_ATTRIBUTE(cell_footprint (CELL) (STRING) )
    DEFINE_ATTRIBUTE(in_place_swap_mode (LIBRARY) (STRING) )
    DEFINE_ATTRIBUTE(map_only (CELL) (BOOLEAN) )
    DEFINE_ATTRIBUTE(preferred (CELL) (BOOLEAN) )
    in_place_swap_mode(“no_swapping”)
Cell ( abc
     cell_footprint(“5mil”)
     map_only(true)
     preferred(false)
     DEFINE_ATTRIBUTE(reference_capacitance (PIN) (FLOAT) )
     Pin ( z PinType(output) reference_capacitance(0.5) )
     )
```

Table below shows various .lib attributes that can be mapped to TLF user defined attributes by Syn2tlf

TLF4.3 identifier(.lib simple attribute)	type	scope
Inplace optimization and Logic attribute		
CELL_FOOTPRINT	STRING	CELL
IN_PLACE_SWAP_MODE	STRING	LIBRARY
USER_FUNCTION_CLASS	STRING	CELL
Test/Fault		
DONT_FAULT	STRING	CELL, PIN
COMPLEMENTARY_PIN	STRING	PIN
TEST_OUTPUT_ONLY	BOOLEAN	PIN
X_FUNCTION	STRING	PIN
FAULT_MODEL	STRING	PIN
Enhanced Pad Modeling		
AUXILIARY_PAD_CELL	BOOLEAN	CELL
PAD_TYPE	STRING	CELL
CONNECTION_CLASS	STRING	PIN
DEFAULT_CONNECTION_CLASS	STRING	LIBRARY
Enhanced modeling for sequential cells		
INPUT_MAP	STRING	PIN
INTERNAL_NODE	STRING	PIN

TLF4.3 identifier(.lib simple attribute)	type	scope
Pin attributes for clock-gating cells		
CLOCK_GATE_CLOCK_PIN	BOOLEAN	PIN
CLOCK_GATE_ENABLE_PIN	BOOLEAN	PIN
others (Non .lib attribute)		
SCALE_SLEW_TIMES*	BOOLEAN	LIBRARY
EDGE_TRIGGERED*	BOOLEAN	PIN
STATE_VARIABLE*	BOOLEAN	PIN
STATE_VARIABLE_INVERTED*	BOOLEAN	PIN
STATE_VARIABLE_MAP*	STRING	PIN
CELL_POWER_ARCS*	BOOLEAN	CELL

* These are the define attributes generated by Syn2tlf that have no correspondence in .lib. Rest all define attributes inthe above table are generated by Syn2tlf from corresponding .lib simple attributes. See examples in the Appendix for the usage of these define attributes.

Description of define attributes generated by Syn2tlf having no correspondence in .lib is given below:

CELL_POWER_ARCS:

This is a cell level, boolean user defined attribute. The Pin level internal power of .lib is modeled as pin/path level INTERNAL_ENERGY/SC_ENERGY TLF constructs. The CELL_POWER_ARC attribute should be set to TRUE for modeling Cell level internal_power of .lib to pin/path level INTERNAL_ENERGY/SC_ENERGY TLF constructs.

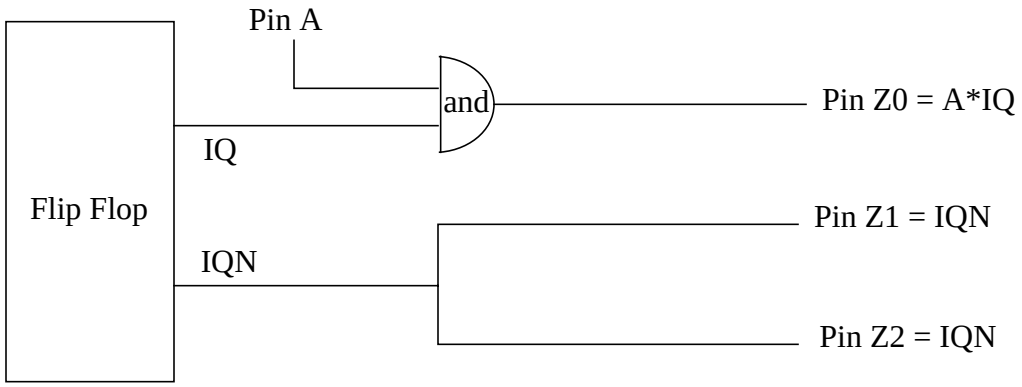
STATE_VARIABLE_MAP:

This is pin level, string type user defined attribute. This attribute can be used to specify the name of state_variable/internal pin which can further be referred in `FUNCTION` or `TEST_FUNCTION` statement to represent the output or inverted output functionality of `REGISTER/TEST_REGISTER/LATCH/TEST_LATCH` construct. Syn2tlf generates this attribute to pass on the ff/latch output variable name contained in the function attribute in .lib.

STATE_VARIBALE:

This is boolean type user defined attribute which can be specified only for internal pins. If set to TRUE, it indicates that the corresponding internal pin represents output functionality of `RESISTER/TEST_REGISTER/LATCH/TEST_LATCH` construct present in the cell. This should be used to model the functionality in case(see the Figure 1):

- there is combinational logic at the output pin
- output pin is driving multple outputs



STATE_VARIBALE_INVERTED:

This is, similar to `STATE_VARIABLE`, boolean type user defined attribute, except that here the internal pin represents the inverted output functionality of `RESISTER/TEST_REGISTER/LATCH/TEST_LATCH` construct present in the cell.

EDGE_TRIGERRED:

This is pin level boolean type user defined attribute. If this attribute is set to TRUE for a Pin, then all timing arcs orginating from that Pin should be treated as rising/falling edge timing arcs. This attribute should be set to TRUE in the pins which may not be clock pins but the timing arcs originating from these pins are of rising/falling edge type. However, the

combinational paths which originate from clock pins are not considered as EDGE_TRIGERRED arcs.

SCALE_SLEW_TIMES:

This is library level boolean user defined attribute. If set to FALSE, this indicates the input slew values are not be scaled. This means only nominal slew values are to be used in delay lookup. However, output slew values are scaled before these are added to the delay values. If the TLF4.3 is created through Syn2tlf, this attribute is set according to the value of scale_slew_times attribute in .lib. Note that, scale_slew_times is not part of the standard .lib format.

DELAY

Syntax

DELAY(*model*)

Context

A DELAY statement can appear within a PATH or PATH_EXTENSION statement after the output transition and before the SLEW statement. The PATH and PATH_EXTENSION statements can appear at the CELL level.

Description

Provides the data to calculate the delay of a signal under the conditions listed in the enclosing PATH statement.

Arguments

model

Does one of the following:

References a previously defined model.

Specifies an inline model consisting of an algorithm clause and parameters. Use the following syntax:
algorithm
{[*parameter*](*value*)|(*cond*[*parameter*](*value*))}...

For valid parameter values, refer to the usage MODEL statement and Chapter 8, “Examples.”

A delay model (whether named or inline) can contain a single value, a *min::max* pair, or a *min:typ:max* triplet. PVT derating can convert a single value into either a *min::max* pair or a *min:typ:max* triplet.

Examples

```
Delay(td_A_to_Z_rise)
Delay((Const (1.0:2.0:3.0)))
```

Related Statement

SLEW

DONT_TOUCH

Syntax

DONT_TOUCH

Context

A DONT_TOUCH statement can appear at the CELL level.

Description

Specifies that the given cell must not be removed during optimization. If DONT_TOUCH is specified, then all instances of the cell must be preserved during optimization.

Example

DONT_TOUCH

Related Statement

DONT_USE

DONT_USE

Syntax

```
DONT_USE
```

Context

A DONT_USE statement can appear at the CELL level.

Description

Specifies that the given cell must not be added to the design during optimization. For example, by using DONT_USE for an I/O cell, you can instruct synthesis tools not to put the cell into the core design.

Example

```
DONT_USE
```

Related Statement

DONT_TOUCH

DRIVETYPE

Syntax

DRIVETYPE(*driveType*)

Context

A DRIVETYPE statement can appear within the BUS and PIN statements.

Description

Specifies the drive type of a pin.

Arguments

driveType

```
cmos | nmos | pmos | nmos_pass | pmos_pass |
cmos_pass | ttl | open_drain | open_source
```

Example

```
DRIVETYPE(cmos)
```

ENABLE

Syntax

ENABLE(*enable_cond*)

Context

An ENABLE statement can appear within the MEMORY_BUS statement at the CELL level, or within the BUS or PIN statements.

Description

Specifies the condition for the tristate enable signal for a bus or pin or for an asynchronous read or write operation on a memory bus. An ENABLE statement is a combinational function expression associated with tristate output pins (or multipin ports).

Arguments

enable_cond

Describes the input pin condition that must be true for the input pin to drive the output. If the condition is false, the output is in a high impedance state. Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Example

```
Pin(A
    Pintype(bidir)
    Function(B)
    Enable(Dir && OE)
    )
```

ENVIRONMENT

Syntax

ENVIRONMENT(*environment*)

Context

An ENVIRONMENT statement can appear within the HEADER statement.

Description

Gives an indication of the specific process, voltage, and temperature (PVT) conditions for which the data in the timing library database was prederated. For example, data can be prederated for conditions that apply to commercial, industrial, or military specifications.

When compiling the TLF file, the environment name is by default added to the compiled file name. Consequently, when several TLF files exist, each of which characterizes a library of cells for different PVT conditions, the environment name allows the user to access the correct data.

Arguments

environment

Specifies the type of application to which the timing data applies. Use a string — a sequence of characters surrounded by double-quotes (") — as shown in the example.

Example

The ENVIRONMENT statement in this library header indicates that the timing data in this TLF file are prederated for a commercial application.

```
Header(
        Library("cmos500k")
        Environment("commercial")
    )
```

EQ_CELLS

Syntax

```
EQ_CELLS(
        (cellName cellName...)...
        )
```

Context

An EQ_CELLS statement can appear at the library level. An EQ_CELLS statement must appear after the CELL statements which describe the cells referenced in the EQ_CELLS statement.

Description

Specifies cells that are logically equivalent. Logically equivalent cells can be connected in parallel for increased drive strength and reduced output delays. For more information on equivalent cells, see “Equivalent Cells” on page 33.

Arguments

cellName

References a cell. The cell must have been defined using a CELL statement.

Example

```
Eq_Cells(
        (INV BIGINV)
        (BUFA BUFB BUFC)
        )
```

EQ_PINS

Syntax

```
EQ_PINS (
        (pinName pinName...)...
        )
```

Context

An EQ_PINS statement can appear in a CELL statement. An EQ_PINS statement must appear after the PIN statements which describe the pins referenced in the EQ_PINS statement.

Description

Specifies pins that are electrically interconnected.

Arguments

pinName

Identifies a pin. If the pin is part of a bus, use an identifier followed by the bit information. The bit information is an integer enclosed in square brackets ([ ]).

Example

In this example, pin A is declared to be electrically connected to bit 3 of vector pin B.

```
Eq_Pins(
        (A B[3])
        )
```

ERROR

Syntax

ERROR(*errValue*)

Context

An ERROR statement can appear within the following limit statements at any level in which they appear:

- FANOUT_LIMIT, FANOUT_MIN
- FLUENCE_LIMIT
- LOAD_LIMIT, LOAD_MIN
- SLEW_LIMIT, SLEW_MIN
- VDROP_LIMIT

Description

Specifies a limit that a delay calculator can compare with the values it calculated to determine when it must generate an error message.

Arguments

errValue

Specifies the boundary value. You can use a float value, a *min::max* pair, or a *min:typ:max* triplet.

Note: No checking will be done if you either specify a tilde (~) as the value or omit the ERROR statement.

FALL

Syntax

FALL(*fallValue*)

Context

A FALL statement can appear within the following TLF statements:

- CSAT, CSBT, CTTAT, CTTBT
- DEFAULT_SLEW, SLEW_LIMIT, SLEW_MIN
- SC_ENERGY, TOTAL_ENERGY
- PROC_MULT
- TEMP_MULT
- VOLT_MULT

Description

Indicates that the given value applies to falling waveforms only. It is usually accompanied by a RISE statement.

Arguments

fallValue

Specifies the property value for a falling signal. The value depends on the property.

Example

```
Properties(
         Default_Slew(Rise(2.10:3.10:4.10) Fall(1.20:2.20:3.20))
         Slew_Limit(Warn(Rise(8) Fall(6))
         Error(Rise(12) Fall(10)))
         )
```

Related Statement

RISE

FLUENCE

Syntax

FLUENCE(*value*)

Context

A FLUENCE statement can appear within a CELL or PATH statement. If the FLUENCE statement appears within a CELL statement, the value is used for all paths that have no PATH specification. If the FLUENCE statement appears within a PATH statement, the value overrides the CELL level value.

Description

Specifies the fluence per transition for a power arc (path) within a cell. Fluence values are assumed to be measured in the unit of coulombs/meter square (C/m^2).

Fluence is a measure of cell degradation due to hot electron effect. Every time a cell drives, its mobility and threshold parameters change slightly, especially for cases of fast input slew or large output load. Over a lifetime at operating frequency, these changes accumulate until the transistor timing moves out of specification. Catastrophic failure seldom occurs, but design timing is affected and might cause circuit malfunction. Fluence can be described as a measure of the flux of injected hot electrons or the amount of damage caused. A FLUENCE statement expresses the same on a per transition basis and is modeled using a 2D table as follows:

$$((fluence)/(transition)) = f(Input\ Slope, Output\ Load)$$

Arguments

value

float | *min*::*max* | *min*:*typ*:*max* | *fluence_model*
Specifies the value for fluence as *float*, *min::max*, *min:typ:max* or *fluence_model.*

Example

This example shows the use of FLUENCE at CELL and PATH levels and FLUENCE_LIMIT at CELL and PIN levels:

```
Header(
     Library("fluence_lib")
     TLF_Version("4.1")
     Generated_By("Maruti")
     )
Cell(myCell
    Fluence_Model(fluenceMod1
        (Spline
            (Input_Slew_Axis 0.1 0.3)
            (Load_Axis 0.01 0.03)
            data(
                // fluence for slew 0.1ns
                (0.00024718 0.000167772)
                // fluence for slew 0.3ns
                (0.000111914 0.000016904)
            )
        )
    )
    Fluence_Model(defFluenceMod
        (Const (0.001))
    )
    Fluence_Model(fluenceMod2
        (Const (0.01:0.02:0.03))
    )

    FLUENCE(defFluenceMod) // Cell level default fluence
    FLUENCE_LIMIT(0.1:0.2:0.3) // FLUENCE_LIMIT at cell level

    Pin(Y Pintype(Output) Function(~((A0 & A1) & A2))
        Capacitance(1.000000)
        // FLUENCE_LIMIT at pin level
        FLUENCE_LIMIT(WARN(0.205) ERROR(0.315))
    )

    Pin(A0 Pintype(Input) Capacitance(17.770000))
    Pin(A1 Pintype(Input) Capacitance(18.600000))
    Pin(A2 Pintype(Input) Capacitance(16.600000))

    PATH(A0 => Y 01 10 DELAY(..) SLEW(..) FLUENCE(fluenceMod1))
    PATH(A0 => Y 10 01 DELAY(..) SLEW(..) FLUENCE(fluenceMod1))
    PATH(A1 => Y 01 10 DELAY(..) SLEW(..) FLUENCE(fluenceMod1))
    PATH(A1 => Y 10 01 DELAY(..) SLEW(..) FLUENCE(fluenceMod1))

    // Here, by default, cell level default fluence
    // defFluenceMod will be used for fluence
    PATH(A2 => Y 01 10 DELAY(..) SLEW(..))
```

```
    PATH(A2 => Y 10 01 DELAY(..) SLEW(..) FLUENCE(fluenceMod2))
)
```

Related Statements

FLUENCE_LIMIT, usage_MODEL

FLUENCE_LIMIT

Syntax

FLUENCE_LIMIT(*value*)

or

FLUENCE_LIMIT(WARN(*value*) ERROR(*value*))

Context

A FLUENCE_LIMIT statement can appear within a CELL statement and within an output PIN statement. If a FLUENCE_LIMIT statement appears at the CELL level, the value is used as the default FLUENCE_LIMIT for all output pins. If a FLUENCE_LIMIT statement appears within an output PIN statement, the value overrides the CELL level value.

Description

Specifies a limit on the total fluence at an output pin. Fluence values are assumed to be measured in the unit of coulombs/meter square (C/m^2).

Fluence limit can be described as the total amount of damage a cell can take based on the following equation:

$$fluence\text{–}limit = \frac{fluence}{transition} frequency \times lifetime$$

In this equation,

$$((fluence)/(transition)) = f(Input\ Slope, Output\ Load) = \text{FLUENCE}$$

Where:

frequency is the operating frequency.

lifetime is the lifetime of the cell at the operating frequency.

TLF provides the values of the construct for warning and error messages when you use the WARN and ERROR statements.

Arguments

value

float | min::max | min:typ:max
Specifies the value for fluence_limit as *float*, *min::max*, or *min:typ:max.*

Examples

See example of FLUENCE.

Related Statement

FLUENCE

FUNCTION

Syntax

`FUNCTION(`*`expression`*`)`

Context

A `FUNCTION` statement can appear within a `PIN` statement for an output or bidirectional pin (or multipin port).

Description

When associated with an output or bidirectional pin (or multipin port), it describes the value of the output pin as a function of the input pins.

Arguments

`expression`

Describes the output pin value in terms of input pin logic values. Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Example

```
Pin(Z Pintype(output)
    Function(A&&B&&C&&D)
    )
Pin(A[7:0] Pintype(bidir)
    Function(B) ENABLE(OE)
    )
```

GENERATED_BY

Syntax

GENERATED_BY(*author*)

Context

A GENERATED_BY statement can appear within the HEADER statement.

Description

Identifies the person or group who created the TLF file. This statement is for reference purposes only.

Arguments

author

Specifies the name of the person or group who created the timing data. Use a string — a sequence of characters surrounded by double-quotes (") — as shown in the example.

Example

```
Header(
     Library("cmos500k")
     Generated_By("tlf_group")
     )
```

GROUND_CURRENT

Syntax

GROUND_CURRENT(*currentModel*)

or

GROUND_CURRENT(*currentModel* [COND(*cond*)])

Note: The COND option can appear at the BUS and PIN levels only.

or

GROUND_CURRENT {RISE(*currentModel*) FALL(*currentModel*))

Note: RISE and FALL are supported at the BUS and PIN levels only.

Context

A GROUND_CURRENT statement can appear within CELL, PATH, BUS, and PIN statements.

Description

Specifies the current flow from load to ground. This current discharges the load. It is used to account for the power dissipation when the output has a fall transition. This definition of ground current also includes short circuit current.

Detailed information about current modeling is given in the description of SUPPLY_CURRENT.

Arguments

currentModel

AVE(*value value*) | TRIPULSE(*value value value*) | GENPULSE(*value* (*value value*)... *value*) | *waveTableModel*

AVE(*value value*)

Specifies single or average value for the current followed by the average value for the duration of the measurement. For example, AVE(0.3 2.0). Here 0.3 is the average current value and 2.0 is the average time value. Units are specified by CURRENT_UNIT and TIME_UNIT.

`TRIPULSE(`*`value value value`*`)`

Approximates current to a triangular pulse. Thus, current pulse can be modeled by four parameters: non zero start time of the rising current, rise time before peak value of the current, peak value of the current, fall time after peak value of the current. For example,

`TRIPULSE(0.5 (1.1 0.6) 1.4)`

where `0.5` is the start time of the `1.1` rise time, `1.1` is the rise time before peak, `0.6` is the current peak value, and `1.4` is the fall time after peak.

`GENPULSE(`*`value`* `(`*`value value`*`)... `*`value`*`)`

Approximates current pulse by a set of linear segments joined together. It is modeled by a series of coordinates. For example,

```
GENPULSE(0.2 (0.4 0.8) (0.6 1.0) (0.8 0.7) 1.0)
```

where `0.2` is the first time value with a current value of `0.0`, successive pairs are (time, current) pairs, and `1.0` is the last time value with a current value of `0.0`.

`waveTableModel`

Specifies either the name of a wavetable previously defined by the WAVETABLE statement or an in line WAVETABLE model description. A waveform is specified as a set of current and time coordinates for each input slew and output load values.

Note: *`waveTableModel`* is an extension of the Spline/Table model. In case of a wave table, each table data value describes a waveform in terms of the current and time coordinates. See WAVETABLE for details.

`value`

`float` | *`min`*`::`*`max`* | *`min`*`:`*`typ`*`:`*`max`* | *`model`*

Specifies the value as *float*, *min::max*, *min:typ:max* or *model.*

Examples

```
GROUND_CURRENT(TRIPULSE(0.2 (0.7 0.6) 0.8))
GROUND_CURRENT(AVE(0.9 5.0) COND(cond))
```

See also the example of SUPPLY_CURRENT.

Related Statements

CELL_SPOWER, INTERNAL_ENERGY, PIN_SPOWER, SC_ENERGY, SUPPLY_CURRENT, TOTAL_ENERGY, WAVETABLE

HEADER

Syntax

```
HEADER(
      LIBRARY(library)
      [TLF_VERSION(tlf_version)]
      [DATE(date)]
      [VENDOR(vendor)]
      [VERSION(version)]
      [GENERATED_BY(author)]
      [TECHNOLOGY(technology)]
      [ENVIRONMENT(environment)]
      )
```

Context

A HEADER statement must appear at the beginning of the TLF file.

Description

Contains general library information. The HEADER statement is required.

Arguments

LIBRARY	Specifies the name of the library database (required).
TLF_VERSION	Specifies the version of the TLF language used.
DATE	Specifies the TLF creation date.
VENDOR	Specifies the vendor name.
VERSION	Specifies the version of the TLF file.
GENERATED_BY	Specifies the name of the person or group who created the timing data.
TECHNOLOGY	Provides an identifier for the fabrication process to which the timing data applies.

ENVIRONMENT — Provides an identifier for the PVT environment condition to which the timing data applies.

Example

```
Header(
     Library("cmos500k")
     TLF_Version("4.1")
     Date("4/20/99")
     Vendor("niftychips")
     Version("5")
     Generated_By("Cadence")
     Technology("CMOS")
     Environment("com")
     )
```

HOLD

Syntax

Within a CELL statement:

```
HOLD(inputPorts {*> | =>} referencePorts [arcType]
     [COND(cond)
          | COND_START(condStart)| COND_END(condEnd)
          | COND_START(condStart) COND_END(condEnd)]
     [SDF_COND(sdfcond)| SDF_COND_START(sdfStart)
          | SDF_COND_END(sdfEnd)
          | SDF_COND_START(sdfStart) SDF_COND_END(sdfEnd)]
     [inputTransition] [pinTrigger]
     model
     )
arcType: NON_SEQ
```

Within a NO_CHANGE statement:

```
HOLD(model)
```

Context

A HOLD statement can appear in a CELL statement, either as a timing check statement following the PATH statements. A HOLD statement can also appear within a NO_CHANGE statement.

Description

A HOLD timing check statement includes pin and model information for a delay calculator to calculate the hold timing check. For a definition of this timing check, see “Hold” on page 23.

When included within a NO_CHANGE timing check statement, the HOLD statement specifies the model that is used for the hold portion of the no-change timing check.

Arguments

inputPorts

Identifies the input pins (usually data pins) to which the hold timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the

list with parentheses. If you specify only one pin, you can omit the parentheses.

`*>`

Indicates a many-to-many mapping. For example, `(A B)*>(Q Q_)` means that the specification can be used for the timing checks `A` to `Q`, `A` to `Q_`, `B` to `Q`, and `B` to `Q_`.

`=>`

Indicates a one-to-one mapping. For example, `(A B)=>(Q Q_)` means that the specification can be used for the timing checks `A` to `Q` and `B` to `Q_`.

`referencePorts`

Identifies the reference pins (usually clock pins) to which the timing check applies. You must have defined the pins using a `PIN` statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

arcType

Specifies the arctype as non-sequential. Non-sequential setup/hold can be used for setup/hold checks on cells that do not contain registers or latches. These are especially relevant for macro-cells consisting only of timing arcs without any explicit `REGISTER` or `LATCH` statements. In such cases, timing analysis tools extract register or latch information by using setup/hold arcs or clock pins. If a latch is extracted for a non-sequential cell, then timing analysis tool can incorrectly perform time borrowing across this latch. Marking setup/hold as non-sequential helps prevent this kind of false extraction

`COND`

Specifies an expression that conditionalizes this hold timing check. The timing check is evaluated only if the condition is true.

`COND_START`

Specifies an expression that conditionalizes the first edge (clock edge) of the hold timing check. The timing check is evaluated only if the condition is true.

`COND_END`

Specifies an expression that conditionalizes the second edge

(data edge) of the hold timing check. The timing check is evaluated only if the condition is true.

SDF_COND

Specifies an expression to be inserted in the SDF file in the COND construct created for the two ports in the HOLD construct.

SDF_COND_START

Specifies an expression to be inserted in the SDF file in the COND construct created for the second port (clock) in the HOLD construct.

SDF_COND_END

Specifies an expression to be inserted in the SDF file in the COND construct created for the first port (data) in the HOLD construct.

inputTransition

Specifies the input transition for this timing check. If not specified, the timing check applies to both rising and falling transitions.

inputTransition	**Use for** . . .
01	Positive edge of edge-triggered cells or high value of the reference signal of level-sensitive cells
10	Negative edge of edge-triggered cells or low value of the reference signal of level-sensitive cells

pinTrigger

Specifies the condition of the reference pin used to determine the timing check. If not specified, the pin trigger is taken from the PIN statement for the reference port.

pinTrigger	**Use for** . . .
posedge	Positive-edge-triggered cells
negEdge	Negative-edge-triggered cells
low	Low-level-sensitive cells

pinTrigger	**Use for** . . .
`high`	High-level-sensitive cells

`model`

Specifies the model information for the timing check. Refer to an existing model, or supply the data in the statement. For valid parameter values, refer to the `usage_MODEL` statement and Chapter 8, “Examples.”

Example

```
Hold(D => CLK 01 posEdge NON_SEQ SDF_COND(clr) COND(clr)
    TchkRiseModel0)
```

Related Statement

`SETUP`

HYSTERESIS

Syntax

```
HYSTERESIS
```

Context

A HYSTERESIS statement can appear within a PAD_PROPS statement. A PAD_PROPS statement can appear within a BUS or PIN statement.

Description

Specifies that the pad has hysteresis. If the statement is not present, it is assumed that hysteresis does not apply to the pad.

See the description of PAD_PROPS for additional information.

Example

```
HYSTERESIS
```

Related Statements

CSAT, CSBT, CTTAT, CTTBT, PAD_PROPS

IGNORE_CELL

Syntax

```
IGNORE_CELL
```

Context

An `IGNORE_CELL` statement can appear at the `CELL` level.

Description

Specifies that a cell is to be ignored in delay calculations.

Example

```
IGNORE_CELL
```

Related Statement

`CELL`

INPUT

Syntax

INPUT(*expression*)

Context

An INPUT statement can appear in a LATCH or REGISTER statement. A LATCH or REGISTER statement can appear within a CELL statement.

Note: INPUT is also used as a predeclared value in the PINTYPE statement. The PINTYPE statement can appear within a PIN statement.

Description

Describes the latch or register data value.

Arguments

expression

Describes the data value in terms of input pin and internal state signals. Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Example

```
Latch(
    Output(Q)
    Inverted_Output(QN)
    Clock(EN)
    Input(D && !HOLD || Q && HOLD)
    )
```

INSERTION_DELAY

Syntax

```
INSERTION_DELAY(
                pinName
                {FAST | SLOW}
                inputTransition internalTransition
                DELAY(delay)
                SLEW(slew)
                )
```

Context

An INSERTION_DELAY statement can appear after the properties within a CELL statement but can be interleaved with timing check statements. For more information on cell level properties, refer to Chapter 5, “Properties.”

Description

Describes the delay of a signal from an input pin to clock pins internal to the cell.

For each pin driving an internal clock tree, specify two INSERTION_DELAY statements (a FAST and a SLOW statement) for each input transition—internal transition pair for which there is an insertion delay.

The INSERTION_DELAY statement enables characterization of the minimum (FAST) and maximum (SLOW) paths through a clock tree embedded within the cell. This data can be used by a clock tree generator to balance the delay to clock pins inside the cell with the delay to clock pins outside the cell.

Arguments

pinName
: Identifies the pin from which the insertion delay to internal clock pins is measured.

FAST, SLOW
: Specifies a lower and upper bound, respectively, for the delay between two pins.

inputTransition

Specifies the signal transition on the specified input to which the INSERTION_DELAY statement applies. Choose either of the following values:
01,10

internalTransition

Specifies the signal transition on the internal clock pin to which the INSERTION_DELAY statement applies. Choose either of the following values:
01,10

Note: The INSERTION_DELAY statement should describe only the active edge paths to internal clock pins. For instance, if the internal clock tree only contains rising edge triggered leaf pins, the INSERTION_DELAY statements for this tree should specify 01 for the *internalTransition* argument.

DELAY

Provides the data to calculate the insertion delay.

SLEW

Provides the data to calculate the slew of the clock signal at the clock pins inside the cell.

Example

```
Insertion_Delay(
                CLK Fast 01 01 Delay(Min_delay) Slew(Min_slew)
                )
Insertion_Delay(
                CLK Slow 01 01 Delay(Max_delay) Slew(Max_slew)
                )
```

INTERNAL_ENERGY

Syntax

INTERNAL_ENERGY(*model*)

or

INTERNAL_ENERGY(*model* [COND(*cond*)])

or

INTERNAL_ENERGY(RISE(*model*) FALL(*model*) [COND(*cond*)])

Note: The COND option applies at the PIN level only.

Context

An INTERNAL_ENERGY statement can appear within the CELL, PATH, BUS and PIN statements.

Description

Specifies the energy consumed within the periphery of a cell. This includes short-circuit energy (SC_ENERGY) and the energy consumed as a result of charging and discharging the output pin load.

See the description of SUPPLY_CURRENT for more information.

Arguments

cond

Specifies a condition that is true only when the short circuit energy value is valid. Use an expression of type *expression*. See “Conditions for Path Delays” on page 50 for more information on this type of expression.

model

Specifies the model information. Refer to an existing model, or supply the data in the statement. The units for energy are derived by multiplying the specified POWER_UNIT by TIME_UNIT. For valid parameter values, refer to the usage_MODEL statement and Chapter 8, “Examples.”

Example

```
Cell (...
    Energy_Model(intEnergyModel
        ( Const
            (5.0)
        )
    )
    INTERNAL_ENERGY(intEnergyModel)
    ...
    )
```

Related Statements

GROUND_CURRENT, SC_ENERGY, SUPPLY_CURRENT, TOTAL_ENERGY

INVERTED_OUTPUT

Syntax

`INVERTED_OUTPUT(`*`pinName...`*`)`

Context

An `Inverted_Output` statement can appear in a `LATCH` or `REGISTER` statement. A `LATCH` or `REGISTER` statement can appear within a `CELL` statement.

Description

Identifies the output pins that connect to the inverted output(s) of a register or latch.

Arguments

`pinName`

Specifies the name of an inverted output pin. You must have defined the pin using a `Pin` statement.

Example

```
Latch(
    Output(Q)
    Inverted_Output(QN)
    Clock(EN && !HOLD)
    Input(D)
    )
```

Related Statement

`OUTPUT`

LATCH

Syntax

```
LATCH(
    CLOCK(clock_condition)
    [SLAVE_CLOCK(clock_condition)]
    INPUT(expression)
    {OUTPUT(pinName...)| INVERTED_OUTPUT(pinName...)|
        OUTPUT(pinName...) INVERTED_OUTPUT(pinName...)}
    [SET(set_condition)]
    [CLEAR(clear_condition)]
    [CLEAR_PRESET_VAR1(value)]
    [CLEAR_PRESET_VAR2(value]
    )
```

Context

A LATCH statement can appear in a CELL statement following the PIN statements.

Note: If a register or latch block has more than one INPUT, CLOCK, OUTPUT, or INVERTED_OUTPUT statements, the latest definition overwrites all the others. Applications that use the TLF parser to read TLF see only one INPUT, CLOCK, OUTPUT, or INVERTED_OUTPUT statement in the register or latch block.

Description

Describes a level-sensitive (transparent) latch.

The sequential-logic behavior must be included for static timing purposes. A static timing analyzer needs to know how separate delays and constraints are related to each other in sequential-logic cells.

Note: This statement does not need to indicate the actual function. It can be an abstracted variant that expresses only the relevant timing relationships between delay paths and timing constraints.

Arguments

CLOCK

Describes when the clock or latch enable is active.

SLAVE_CLOCK
: Describes a secondary clocking signal.

INPUT
: Describes the latch data value.

OUTPUT
: Identifies the output pin.

INVERTED_OUTPUT
: Identifies the pins connecting to the inverted outputs of the latch.

SET
: Describes an asynchronous set.

CLEAR
: Describes an asynchronous clear.

CLEAR_PRESET_VAR1
: Specifies a value for an OUTPUT pin when CLEAR and SET are both active at the same time.

CLEAR_PRESET_VAR2
: Specifies a value for INVERTED_OUTPUT pin when CLEAR and SET are both active at the same time.

Example

```
Latch(
    Output(Q)
    Input(D && !HOLD || Q && HOLD)
    Clock(CLK)
    Set(SET)
    Clear(CLR)
    CLEAR_PRESET_VAR1(0)
    CLEAR_PRESET_VAR2(T)
    )
```

Related Statements

REGISTER, TEST_LATCH

LIBRARY

Syntax

LIBRARY(*library*)

Context

A LIBRARY statement can appear in the HEADER statement.

Description

The LIBRARY statement specifies the name of the library database to which the compiler writes the compiled TLF (ctlf) file by default.

Arguments

library

Specifies the name of the library database. Use a string — a sequence of characters surrounded by double-quotes (") — as shown in the example.

Example

```
Header(
     Library("cmos500k")
     )
```

MAP_TO_STPIN

Syntax

MAP_TO_STPIN(*stTabName* (*inPinMapList* : *stPinMapList*))

Context

A MAP_TO_STPIN statement can appear at PIN level. It only appears in an output pin. The MAP_TO_SPIN statement is optional and can appear multiple times. If the same *stTabName* is specified, the latest definition overrides any previous definitions.

Description

Specifies the mapping from actual pin names to the dummy pin list specified in the state table.

Arguments

stTabName

Specifies the name of the state table as previously defined by STATE_TABLE.

inPinMapList

pinName pinName...
Specifies a list of the actual pin names, separated by commas, to be mapped to the input pins of the state table.

stPinMapList

pinName pinName...
Specifies a list of the actual pin names, separated by commas, to be mapped to the state pins of the state table.

pinName

Specifies the actual name of a pin. This pin must be previously defined using the PIN statement.

Example

```
STATE_TABLE(JK_ff
            (J K CN CD : IQ)
            (- - - 0 : - : 0)
```

```
(- - ~F 1 : - : N)
(0 0 F 1 : 01 : 01)
(1 0 F 1 : - : 1)
(0 1 F 1 : - : 0)
(1 1 F 1 : 01 : 10)

PIN(Z
    ...
    MAP_TO_STPIN(JK_ff (J K CLM CLS : Z)
    )
...
)
```

Related Statement

STATE_TABLE

MEMORY_BUS

Syntax

```
MEMORY_BUS(databusName
         BUSMODE(READ | WRITE | READWRITE)
         ADDRESS_BUS(adrBusName)
         CLOCK(clk_cond)| ENABLE(enb_cond)
         )
```

Context

A MEMORY_BUS statement can appear at CELL level. The MEMORY_BUS statement is optional and can appear multiple times at cell level. Multiple definitions of the same bus name are not allowed.

Description

Specifies a data bus for memory read and write operations.

Data transfer to and from memory takes place through data buses. A data bus is defined in the MEMORY_BUS statement. A data bus can be used for read operation only, write operation only, or for both operations. If data transfer on the bus takes place synchronously with the clock, the bus has a clock expression associated with it. If data transfer on the bus takes place asynchronously, the bus has an enable expression associated with it. Each data bus also has an address bus associated with it, and the operation on the bus takes place with respect to this address bus.

Arguments

databusName
: Specifies the name of the data bus as previously defined using BUS at cell level.

BUSMODE
: Specifies that the bus can be used in either read or write or both types of operations.

ADDRESS_BUS
: Specifies the name of the address bus used with the data bus.

CLOCK

Specifies the condition indicating the clock signal. Used only if the memory operation is synchronous.

ENABLE

Specifies the condition indicating the enable signal. Used only if the memory operation is asynchronous.

Checks

- Data bus must have been defined earlier using the BUS statement
- BUSMODE (READ, WRITE or READWRITE) must match BUSTYPE (output, input, or bidir).
- Two read ports or two write ports on the same cell must not have a common address bus.

Example

```
CELL(xyz
        ...
        BUS(read_b ...)
        BUS(rw_bus ...)
        BUS(adr1_bus ...)
        BUS(adr2_bus ...)
        ...
    MEMORY_BUS(read_b
            BUSMODE(READ)
            ADDRESS_BUS(adr1_bus)
                )
    MEMORY_BUS(rw_bus
            BUSMODE(READWRITE)
            ADDRESS_BUS(adr2_bus)
            CLOCK(clk)
                )
        ...
        )
```

Related Statements

BUS, MEMORY_PROPS, PIN

MEMORY_OPR

Syntax

MEMORY_OPR(*memOpr*)

Context

A MEMORY_OPR statement can appear within the MEMORY_PROPS statement. A MEMORY_PROPS statement can appear at the library level within the PROPERTIES statement or at the CELL level.

Description

Specifies asynchronous or synchronous type of memory operation. See the description of MEMORY_PROPS for more information.

Arguments

memOpr

ASYNCHRONOUS | SYNCHRONOUS

ASYNCHRONOUS	Specifies operation occurs with an enable signal.
SYNCHRONOUS	Specifies operation occurs with a clock signal.

Checks

- For asynchronous memory, data bus must not have a CLOCK condition.
- For synchronous memory, data bus must not have an ENABLE condition.

Example

```
MEMORY_OPR(SYNCHRONOUS)
```

Related Statements

BUS, MEMORY_PROPS

MEMORY_PROPS

Syntax

```
MEMORY_PROPS(
         MEMORY_TYPE(...)
         MEMORY_OPR(...)
         ADDRESS_WIDTH(...)
         DATA_WIDTH(...)
         )
```

Context

A MEMORY_PROPS statement can appear at the library level within the PROPERTIES statement or at the CELL level. At the library level, the MEMORY_PROPS statement is optional and can appear only once. At the CELL level, if the MEMORY_BUS statement appears within the CELL statement, the MEMORY_PROPS statement is also required.

Description

Defines the properties of a memory cell. Memory properties are type of memory, synchronous or asynchronous operation, address width, and data width.

A memory device has several properties including type, for example ROM, memory operation (clocked or enabled), address width, and data width. Synchronous memory operation occurs at a clock. This operation requires specifying a clock condition for the data bus within the MEMORY_BUS statement. Asynchronous memory operations are performed using an enable signal. This operation requires specifying an enable condition for the data bus within the MEMORY_BUS statement. Address width is the number of bits in the address. An address bus is defined separately using a BUS statement. Data width indicates word width of memory. A memory data bus is also defined separately using a BUS statement.

Arguments

MEMORY_TYPE
: Specifies the type of memory (ROM, RAM, SRAM, DRAM).

MEMORY_OPR
: Specifies the memory operation (synchronous or asynchronous).

ADDRESS_WIDTH
: Specifies the address width of memory.

DATA_WIDTH
: Specifies the data width of memory.

Checks

- Address width defined using ADDRESS_WIDTH must match the size of address bus as defined using BUS.
- Data width defined using DATA_WIDTH must match the size of data bus as defined using BUS.
- For synchronous memory, data bus must not have an enable condition.
- For asynchronous memory, data bus must not have a clock condition.

Example

```
CELL(xtSm1
     ...
     MEMORY_PROPS(
          MEMORY_TYPE(RAM)
          MEMORY_OPR(SYNCHRONOUS)
          ADDRESS_WIDTH(8)
          DATA_WIDTH(32)
     )
     ...
     MEMORY_BUS(
          ...
     )
     )
```

See also the following examples for more information:

- Memory (Asynch 2x2 RAM with Dual Port) on page 378
- Memory (ROM cell) on page 383
- Memory (Synch 2x2 RAM with Single Port) on page 386

Related Statements

ADDRESS_WIDTH, DATA_WIDTH, MEMORY_BUS, MEMORY_OPR, MEMORY_TYPE

MEMORY_TYPE

Syntax

MEMORY_TYPE(*memType*)

Context

A MEMORY_TYPE statement can appear within the MEMORY_PROPS statement. A MEMORY_PROPS statement can appear at the library level within the PROPERTIES statement or at the CELL level.

Description

Specifies the type of memory.

See the description of MEMORY_PROPS for more information.

Arguments

memType

ROM | RAM | SRAM | DRAM

ROM

Specifies read-only memory. A ROM cell must have at least one bus with BUSMODE as READ. It must not have a bus with BUSMODE as WRITE.

RAM

Specifies random access memory. A RAM cell must contain at least one bus with BUSMODE as READWRITE and BUSTYPE as bidir.

SRAM

Specifies static random access memory. The RAM restrictions given above apply.

DRAM

Specifies dynamic random access memory. The RAM restrictions given above apply.

Checks

- A ROM cell must have at least one bus with BUSMODE as READ. It must not have a bus with BUSMODE as WRITE.
- A RAM, SRAM, or DRAM cell must contain at least one bus with BUSMODE as READWRITE and BUSTYPE as bidir.

Example

```
MEMORY_TYPE(RAM)
```

Related Statements

BUS, MEMORY OPR, MEMORY PROPS

MOBILITY_LIMIT

Syntax

MOBILITY_LIMIT(*value*)

Context

A MOBILITY_LIMIT statement can appear at the CELL, BUS, and PIN level.

Description

Specifies a limit on the mobility degradation of an output pin due to a lifetime accumulation of damage from hot electron effect.

See the description of THRESHOLD_LIMIT.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Specifies the MOBILITY_LIMIT value as *float*, *min::max*, or *min:typ:max.*

Example

```
MOBILITY_LIMIT(0.25)
```

Related Statement

FLUENCE

MPWH

Syntax

```
MPWH(inputPorts
     [OTHER_PINS(other_pin...)]
     [COND(cond)]
     [SDF_COND(sdfcond)]
     model
     )
```

Context

An MPWH statement can appear within a CELL statement following the PATH statements.

Description

Includes pin and model information for a delay calculator to calculate the MPWH timing check. For a definition of this timing check, see “Minimum Pulse Width High” on page 27.

Arguments

inputPorts

Identifies the input pins (usually clock pins) to which the MPWH timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

OTHER_PINS

Identifies an output pin. The table algorithm uses the capacitance load of the output pin in the calculation of the timing check.

COND

Specifies an expression that conditionalizes this MPWH timing check. The timing check is evaluated only if the condition is true.

SDF_COND

Specifies an expression to be inserted in the SDF file in the SDF COND construct created for the port.

model

Specifies the model information needed to calculate the minimum pulse width high. Refer to an existing model, or supply the data in the statement. For valid parameter values, refer to the usage_MODEL statement and Chapter 8, “Examples.”

Example

```
MPWH(CK Other_Pins(Q) SDF_Cond(clr_or_set)
        Cond(CLR || SET) mpwh_CK)
```

Related Statement

MPWL

MPWL

Syntax

```
MPWL(inputPorts
    [OTHER_PINS(other_pin...)]
    [COND(cond)]
    [SDF_COND(sdfcond)]
    model
    )
```

Context

An MPWL statement can appear within a CELL statement following the PATH statements.

Description

Includes pin and model information for a delay calculator to calculate the MPWL timing check. For a definition of this timing check, see “Minimum Pulse Width Low” on page 27.

Arguments

inputPorts

Identifies the input pins (usually clock pins) to which the MPWL timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

OTHER_PINS

Identifies an output pin. The table algorithm uses the capacitance load of the output pin in the calculation of the timing check.

COND

Specifies an expression that conditionalizes this MPWL timing check. The timing check is evaluated only if the condition is true.

SDF_COND

Specifies an expression to be inserted in the SDF file in the SDF COND construct created for the port.

model

Specifies the model information needed to calculate the minimum pulse width low. Refer to an existing model, or supply the data in the statement. For valid parameter values, refer to the `usage Model` statement and Chapter 8, “Examples.”

Example

```
MPWL(CK Other_Pins(Q) SDF_Cond(clr_or_set)
        Cond(CLR || SET) mpwl_CK)
```

Related Statement

`MPWH`

NO_CHANGE

Syntax

```
NO_CHANGE(inputPorts {*> | =>} referencePorts
        [COND(cond)
            | COND_START(condStart)| COND_END(condEnd)
            | COND_START(condStart) COND_END(condEnd)]
        [SDF_COND(sdfcond)| SDF_COND_START(sdfStart)
            | SDF_COND_END(sdfEnd)
            | SDF_COND_START(sdfStart) SDF_COND_END(sdfEnd)]
        [inputTransition][referenceState]
        SETUP(model) HOLD(model)
        )
```

Context

A NO_CHANGE statement can appear within a CELL statement following the PATH statements.

Description

Specifies the pin and model information needed to check a no-change condition. Some logic configurations require that an input remain stable while a reference input is in the active state, such as in strobe inputs or gated clocks. For a definition of this timing check, refer to “No_Change” on page 25.

Arguments

inputPorts

Identifies the input pins (usually data pins) to which the hold timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

*>

Indicates a many-to-many mapping. For example, (A B)*>(Q Q_) means that the specification can be used for the timing checks A to Q, A to Q_, B to Q, and B to Q_.

=>

Indicates a one-to-one mapping. For example, (A B)=>(Q Q_)

means that the specification can be used for the timing checks A to Q and B to Q_.

referencePorts

Identifies the reference pins (usually clock pins) to which the timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

COND

Specifies an expression that conditionalizes this no-change timing check. The timing check is evaluated only if the condition is true.

COND_START

Specifies an expression that conditionalizes the first edge (clock edge) of the no-change timing check. The timing check is evaluated only if the condition is true.

COND_END

Specifies an expression that conditionalizes the second edge (data edge) of the no-change timing check. The timing check is evaluated only if the condition is true.

SDF_COND

Specifies an expression to be inserted in the SDF file in the SDF COND construct created for the two ports in the SDF NOCHANGE construct.

SDF_COND_START

Specifies an expression to be inserted in the SDF file in the SDF COND construct created for the second port (clock) in the SDF NOCHANGE construct.

SDF_COND_END

Specifies an expression to be inserted in the SDF file in the SDF COND construct created for the first port (data) in the SDF NOCHANGE construct.

inputTransition

Specifies the input transition for this timing check. If not

specified, the timing check applies to both rising and falling transitions.

inputTransition	**Use for** . . .
`01`	Positive edge of edge-triggered cells or high value of the reference signal of level-sensitive cells
`10`	Negative edge of edge-triggered cells or low value of the reference signal of level-sensitive cells

`referenceState`

Specifies the active state of the reference signal. If not specified, the reference signal active state is inferred by the Pin type information (high or low) of the reference pin signal.

referenceState	**Use to check . . .**
`low`	No-change condition while reference signal is low
`high`	No-change condition while reference signal is high

`SETUP(`*`model`*`)`

Specifies the model information needed to calculate the minimum time that the leading edge of the input signal must precede the leading edge of the reference signal. Refer to an existing model, or supply the data in the statement. For valid parameter values, refer to the `usage MODEL` statement and Chapter 8, “Examples.”

`HOLD(`*`model`*`)`

Specifies the model information needed to calculate the minimum time that the opposite edge of the input signal must follow the opposite edge of the reference signal. Refer to an existing model, or supply the data in the statement.

Examples

```
No_Change(HOLD => CLK 01 high Setup(...)Hold(...))
No_Change(ADDR => WEB 10 high Setup(...) Hold(...))
No_Change    (D => CLK SDF_Cond_End(clr_or_set)
             Cond_End(CLR || SET) 01 high Setup(...)
             Hold(...))
```

OTHER_PINS

Syntax

OTHER_PINS(*pinName...*)

Context

An OTHER_PINS statement can appear within a PATH_EXTENSION, MPWH, MPWL, or PERIOD statement. These statements can all appear at the CELL level.

Description

Identifies other pins when more than two pins are needed to describe a timing relationship. It is used, for instance, to identify an output pin whose load affects the timing check.

Arguments

pinName

Specifies an extra pin needed to describe a timing relationship. You must have defined the pin using a PIN statement.

Note: This pin cannot be listed as input port, output port, or reference port in a path description or timing check.

Example

```
Path_Extension(CK => QB Other_Pins(Q) 01 10
        Delay(CK_QB_delay_01_10) Slew(...))
MPWL(CK Other_Pins(Q) SDF_Cond(clr_or_set)
        Cond(CLR || SET) mpwl_CK)
```

OUTPUT

Syntax

OUTPUT(*pinName...*)

Context

An OUTPUT statement can appear in LATCH and REGISTER statements at the CELL level. It can also be used as a predeclared value in the PINTYPE statement at the PIN level.

Description

Identifies a pin (or space-separated list of pins) where the output is attached.

Arguments

pinName

Specifies the name of the output pin. You must have defined the pin using a PIN statement.

Example

```
Latch(
    Output(Q)
    Inverted_Output(QN)
    Clock(EN && !HOLD)
    Input(D)
    )
```

Related Statement

INVERTED_OUTPUT

PAD_CELL

Syntax

PAD_CELL

Context

A PAD_CELL statement can appear within a CELL statement.

Description

Specifies that the given cell is a pad cell. A pad cell acts as an interface to the world external to the design.

Example

See example for Units and Pad Modeling on page 390.

Related Statements

PAD_PIN

PAD_PIN

Syntax

PAD_PIN

Context

A PAD_PIN statement can appear within BUS and PIN statements.

Description

Specifies that the given pin is a pad pin that acts as an interface with the environment external to the design.

Example

PAD_PIN

Related Statements

PAD_PROPS, PINTYPE

PAD_PROPS

Syntax

```
PAD_PROPS(
        INPUT_VOLTAGE(...)
        OUTPUT_VOLTAGE(...)
        DCURRENT(...)
        CSAT(...)
        CSBT(...)
        CTTAT(...)
        CTTBT(...)
        HYSTERESIS(...)
        )
```

Context

A PAD_PROPS statement can appear within a BUS or PIN statement. The PAD_PROPS statement is optional and can appear only once within the BUS and PIN statements.

Description

Specifies the pad properties.

A pad cell is identified by the presence of the PAD_CELL statement. A pad pin is identified by the presence of the PAD_PIN statement. A PAD_PROPS statement specifies the properties of a pad pin.

Input voltage and output voltage of a pad pin describes the voltage levels. Various voltage level could be minimum and maximum levels and low and high voltage threshold levels. Minimum input voltage indicates the minimum acceptability limit. Similarly maximum input voltage is the maximum acceptability limit. Low input threshold voltage is maximum input voltage for which input to the core is guaranteed to be at logic 0. Similarly, high input threshold voltage is the maximum input voltage for which the input core is guaranteed to be at logic 1. In case of output voltage, maximum and minimum limits are for generated output voltage. Similarly, low and high threshold voltages are maximum and minimum output voltage generated to represent logic 0 and logic 1, respectively.

Drive current is the current that can be generated by an output pad and is a pulling current for a pull-up or pull-down device.

Current slope and current transition time limits peak noise and smooths out fast output transition. These parameters model the current pulse. Pulse is linearly approximated on both

the sides of the peak or threshold. CSBT is slope of the pulse before threshold and similarly CSAT is slope of the pulse after threshold. CTTBT is the transition time for the current to reach the peak. Similarly, CTTAT is transition time after the peak.

Hysteresis for the pad is made true to accommodate longer transition time when it is subject to more noise problems. Pad with hysteresis has wider threshold tolerances. Due to this voltage, spikes on the input do not propagate to core. Power consumption also cuts down for this kind of pad because an input signal changes only after a threshold has been exceeded.

Arguments

INPUT_VOLTAGE
: Specifies the input voltage level for a pad.

OUTPUT_VOLTAGE
: Specifies the output voltage level for a pad.

DCURRENT
: Specifies the drive current for a pin.

CSAT
: Specifies the current slope after a threshold.

CSBT
: Specifies the current slope before a threshold.

CTTAT
: Specifies the current transition time after a threshold.

CTTBT
: Specifies the current transition time before a threshold.

HYSTERESIS
: Specifies that the pad has hysteresis.

Example

```
HEADER(...)
    usage_MODEL(...)...
    PROPERTIES(
             ...
             INPUT_VOLTAGE(inpVolPad
                           VOLT_LOW_THRESHOLD(...)
```

```
                        VOLT_HIGH_THRESHOLD(...)
                        VOLT_MIN(...)
                        VOLT_MAX(...)
                        )
            )
    CELL(inpPad
        ...
        PIN(extPin
        PAD_PROPS(
                INPUT_VOLTAGE(inpVolPad)
                CSAT(.7)
                CSBT(.4)
                CTTAT(.34)
                CTTBT(.46)
                HYSTERESIS
                )
            )
        )
...
```

See the example for Units and Pad Modeling on page 390 for more information.

Related Statements

CSAT, CSBT, CTTAT, CTTBT, HYSTERESIS, INPUT_VOLTAGE, OUTPUT_VOLTAGE

PATH

Syntax

Syntax for timing specific paths

```
PATH(inputPorts {*> | =>} outputPorts
    [COND(cond)]
    [SDF_COND(sdfcond)]
    [FAST | SLOW]
    [inputTransition] [outputTransition]
    timing_constructs
)
timing_constructs: DELAY(delay) SLEW(slew)
                                  [WAVEFORM_TAIL_RES(resistance)]
```

Syntax for Power/SI specific paths

```
PATH(inputPorts {*> | =>} outputPorts
    [COND(cond)]
    [SDF_COND(sdfcond)]
    [FAST | SLOW]
    [inputTransition] [outputTransition]
    [timing_constructs]
    power_constructs
)
timing_constructs : DELAY(delay) SLEW(slew)
                                  [WAVEFORM_TAIL_RES(resistance)]
Power_constructs : [SUPPLY_CURRENT] [GROUND_CURRENT]
                     [INTERNAL_ENERGY] [TOTAL_ENERGY]
                     [SC_ENERGY] [FLUENCE]
```

Context

A PATH statement can appear in a CELL statement after the properties but can be interleaved with timing check statements. For more information on CELL level properties, refer to Chapter 5, “Properties.”

Description

Describes the delay of a signal transition through a cell, from an input pin to an output pin. The PATH statement contains pin signal direction and model information for cell signal delays.

You can specify more than one PATH statement for a given input and output pin pair, each describing a different case of either input transition, output transition, or condition.

Arguments

inputPorts

Identifies a pin at which the path originates. You must have defined the pin using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

*>

Indicates a many-to-many mapping. For example, (A B)*>(Q Q_) means that the specification can be used for the timing checks A to Q, A to Q_, B to Q, and B to Q_.

=>

Indicates a one-to-one mapping. For example, (A B)=>(Q Q_) means that the specification can be used for the timing checks A to Q and B to Q_.

outputPorts

Identifies a pin at which the path ends. You must have defined the pin using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

COND

Specifies an expression that conditionalizes the PATH statement. The path is valid only if the condition is true.

SDF_COND

Specifies an expression to be inserted in the SDF file in the SDF COND construct preceding the SDF IOPATH construct that is created for this PATH statement.

FAST, SLOW

Specifies a lower and upper bound, respectively, for the delay between two pins. Two PATH statements using FAST and SLOW

can replace any number of PATH statements that have conditional expressions (COND, SDF_COND). Use the FAST and SLOW options when characterizing a high-level cell, where the number and size of the conditional expressions become excessive while providing minimal benefit. For more information on these options, refer to "Fast and Slow Paths" on page 29.

inputTransition

Specifies the signal transition at the input of the path. Choose either of the following values:
01, 10

outputTransition

Specifies the signal transition at the output of the path. Choose one of the following values:
01, 10, 0Z, Z0, 1Z, Z1
0X, X0, 1X, X1, XZ, ZX

timing_constructs

DELAY provides the data to calculate the delay of a signal.
SLEW provides the data to calculate the slew of a signal at the end of the path.
WAVEFORM_TAIL_RES provides the transient resistance to be used by delay calculation.

Power_constructs

[SUPPLY_CURRENT] [GROUND_CURRENT] [INTERNAL_ENERGY] [TOTAL_ENERGY]
[SC_ENERGY] [FLUENCE]
You must specify atleast one of the six power_constructs.

Examples

```
Path(Clk => Q 01 01 Delay(...) Slew(...))
Path(Clk => Q 01 10 Delay(...) Slew(...))

Path((A B)*> X 01 01 Delay(...) Slew(...))

Path(A => Z Cond(!B) SDF_Cond(B==1'b0) 01 10
        Delay(...) Slew(...))
```

Related Statement

FLUENCE, INTERNAL_ENERGY, GROUND_CURRENT, PATH_EXTENSION, SC_ENERGY, SUPPLY_CURRENT, TOTAL_ENERGY

PATH_EXTENSION

Syntax

```
PATH_EXTENSION(inputPorts {*> | =>} outputPorts
            [OTHER_PINS(other_pin...)]
            [COND(cond)]
            [SDF_COND(sdfcond)]
            [FAST | SLOW]
            inputTransition outputTransition
            DELAY(delay) SLEW(slew)
            )
```

Context

A PATH_EXTENSION statement can appear within a CELL statement after the properties but can be interleaved with timing check statements. For more information on cell level properties, refer to Chapter 5, “Properties.”

Description

Describes an output-pin to output-pin delay path for which the Verilog® language requires that a path be referenced from the original signal-source input pin. When generating SDF, a delay path represented by a PATH_EXTENSION statement is transformed by adding the delay of the input-to-output path that drives the first output pin of the output-to-output path. Additionally, the identity of the path changes so the source pin is the same as that of the added input-to-output delay path. See “Output-to-Output Timing Paths” on page 31 for more details.

Arguments

inputPorts

Identifies a pin at which the path originates. You must have defined the pin using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

*>

Indicates a many-to-many mapping. For example, (A B)*>(Q Q_) means that the specification can be used for the timing checks A to Q, A to Q_, B to Q, and B to Q_.

`=>`

Indicates a one-to-one mapping. For example, `(A B)=>(Q Q_)` means that the specification can be used for the timing checks `A` to `Q` and `B` to `Q_`.

`outputPorts`

Identifies a pin at which the path ends. You must have defined the pin using a `PIN` statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

`OTHER_PINS`

Specifies the source pin(s) that drives the output-to-output path. This statement should be used only if the source output pin of an output-to-output delay is driven by more than one input pin. See "Special Cases" on page 31.

`COND`

Specifies an expression that conditionalizes the `PATH_EXTENSION` statement. The path is valid only if the condition is true.

`SDF_COND`

Specifies an expression to be inserted in the SDF file in the SDF `COND` construct preceding the SDF `IOPATH` construct that is created for this `PATH_EXTENSION` statement.

`FAST, SLOW`

Specifies a lower and upper bound, respectively, for the delay between two pins. Two `PATH_EXTENSION` statements using `FAST` and `SLOW` can replace any number of `PATH_EXTENSION` statements using conditional expressions (`COND`, `SDF_COND`). Use the `FAST` and `SLOW` options when characterizing a high-level cell and the number and size of the conditional expressions become excessive while providing minimal benefit. For more information on these options, refer to "Fast and Slow Paths" on page 29.

`inputTransition`

Specifies the signal transition at the input of the extended path. Choose either of the following values:
`01, 10`

outputTransition

Specifies the signal transition at the output of the extended path. Choose one of the following values:
01, 10, 0Z, Z0, 1Z, Z1
0X, X0, 1X, X1, XZ, ZX

DELAY

Provides the data to calculate the delay of a signal.

SLEW

Provides the data to calculate the slew of a signal at the end of the path.

Example

```
Model(CLK_Q_del_01_01 (...))
Model(CLK_Q_del_10_10 (...))
Model(Q_QB_del_01_10 (...))
Model(Q_QB_del_10_01 (...))
Pin(D...)
Pin(Q...)
Pin(QB...)
Pin(CLK...)
        .
        .
        .
Path(CLK => Q 01 01 Delay(CLK_Q_del_01_01) Slew(...))
Path(CLK => Q 10 10 Delay(CLK_Q_del_10_01) Slew(...))
Path_Extension(Q => QB 01 10 Delay(Q_QB_del_01_10) Slew(...))
Path_Extension(Q => QB 10 01 Delay(Q_QB_del_10_01) Slew(...))
```

Related Statement

PATH

PERIOD

Syntax

```
PERIOD(inputPorts
    [OTHER_PINS(other_pin...)]
    [COND(cond)]
    [SDF_COND(sdfcond)]
    [inputTransition] model
    )
```

Context

A PERIOD statement can appear within a CELL statement following the PATH statements.

Description

Includes pin and model information for a delay calculator to calculate the period timing check. For a definition of this timing check, see “Period” on page 27.

Arguments

inputPorts

Identifies the input pins (usually clock pins) to which the period timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

OTHER_PINS

Identifies the output pins. The table algorithm uses the capacitance load of the output pin in the calculation of the timing check.

COND

Specifies an expression that conditionalizes this period timing check. The timing check is evaluated only if the condition is true.

SDF_COND

Specifies an expression to be inserted in the SDF file in the SDF

`COND` construct created for the port in the SDF `PERIOD` construct.

`inputTransition`

Specifies the input transition for which the period timing check must be calculated. If the input transition is not specified, the specified model is used for both transitions.

inputTransition	**Use for . . .**
`01`	Positive edge of edge-triggered cells or high value of the reference signal of level-sensitive cells
`10`	Negative edge of edge-triggered cells or low value of the reference signal of level-sensitive cells

`model`

Specifies the model information used to calculate the minimum period. Refer to an existing model, or supply the data in the statement. For valid parameter values, refer to the `usage_MODEL` statement and Chapter 8, “Examples.”

Example

```
Period(CK Other_Pins(Q) periodModel)
```

PIN

Syntax

```
PIN(pinName
    PINTYPE(usage)
    [CLOCK_PIN]
    [FUNCTION(expression)]
    [ENABLE(enable_condition)]
    [Properties]
)
```

Note: Even though some of the sections are optional, you must specify the sections that you want to include in the order specified in the syntax above.

Context

A PIN statement can appear in a CELL statement following the usage MODEL statements.

Description

Describes a single pin or a collection of pins (a logical grouping of pins, such as DATA[7:0]). A PIN statement is required for each pin that is referenced later in PATH, PATH_EXTENSION, or timing check statements. Every pin of a cell needs a PIN statement if it is to be considered in the timing analysis.

Arguments

pinName

Identifies the pin. If you want to specify a collection of pins that is not a bus, follow the identifier by the bit information. The bit information (for a single-bit pin) is an integer enclosed in square brackets ([ ]). If the cell contains several pins of a bus, follow the identifier by a range specification. The range specification contains two integers (representing the lower and upper bounds of the range) separated by a colon and enclosed in brackets ([ ]). A collection of pins specified within the PIN statement is not considered a bus in the design. To specify a bus, use the BUS statement.

PINTYPE(*usage*)

Indicates the direction of signal applied to the pin or indicates the

type of pin. *usage* is either `input`, `output`, `bidir`, `ground`, `supply, or internal`.

`CLOCK_PIN`

Specifies that the pin is a clock pin.

`FUNCTION(`*`expression`*`)`

Describes the value of an output pin or output port as a function of the input pins. Can be associated only with output or bidirectional pins (or multipin port). Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

`ENABLE(`*`enable_condition`*`)`

Describes the input pin condition that must be true for the input pin to drive the output of a tristate cell. If the condition is false, the output is in a high impedance state. Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

`Properties`

Describes properties specific to the pin being described. Pin properties defined in the `PIN` statement take precedence over pin properties declared at the cell or library level.
For a list of pin level properties, refer to table TBD.

Example

```
Pin(Y
    Pintype(output)
    Function(A && !B[0])
    Load_Limit (Warn(40) Error(50))
    Slew_Limit (Warn(10) Error(20))
    Default_Load(10.000:20.000:30.000)
    Capacitance(20.000)
    )
```

PINTYPE

Syntax

PINTYPE(*usage*)

Context

A PINTYPE statement can appear at the PIN level.

Description

Specifies the direction or type of pin.

Arguments

usage

```
input | output| bidir | ground | supply |
internal
```

input	Specifies an input pin.This is the default if PINTYPE is not specified.
output	Specifies an output pin.
bidir	Specifies a bidirectional pin.
ground	Specifies a POWER PIN connected to GROUND.
supply	Specifies a POWER PIN connected to SUPPLY.
internal	Specifies an internal node pin. See example for Internal Pin on page 403.

Related Statements

CLOCK_PIN, PAD_PIN

PROPAGATION_DELAY_TABLE

Syntax

PROPAGATION_DELAY_TABLE

Context

A PROPAGATION_DELAY_TABLE statement can appear within the LIBRARY, PROPERTIES, and the CELL statements.
Specified at the LIBRARY level, this construct is applicable for all cells in that library. If it is not specified at the LIBRARY level, all cells in the library have cell based delays by default, except those cells for which this attribute is defined at the CELL level.

Description

A PROPAGATION_DELAY_TABLE attribute is used to differentiate between cell based delays and propagation delays. It specifies that the delays are coming from rise/fall_propogation constructs of the .lib file.
You can differentiate the cells that have propagation based delays from those having cell based delays by not specifying PROPAGATION_DELAY_TABLE at LIBRARY level and specifying it at CELL level for those cells that have propagation based delays.
Cadence recommends using libraries with cell delay tables rather than propagation delay tables.

Example

```
Header (
        Library(“all_cells_having_propagation_based_delays”)
        TLF_Version(“4.3”)
        )
Properties(
            PROPAGATION_DELAY_TABLE
            // All cells in the Library have propagation delays
            ...
         )
```

PULL

Syntax

```
PULL(UP | DOWN | EITHER)
```

Context

A PULL statement can appear within the BUS and PIN statements.

Description

Specifies the pull at a pin. PULL(UP) and PULL(DOWN) can be used to specify tie-off pins.

Arguments

UP
: Specifies a pull-up device on the pin.

DOWN
: Specifies a pull-down device on the pin.

EITHER
: Specifies that either a pull-up or pull-down device can apply to the pin.

Example

```
PULL(UP)
```

Related Statements

PULL_CURRENT, PULL_RESISTANCE

PULL_CURRENT

Syntax

PULL_CURRENT(*value*)

Context

A PULL_CURRENT statement can appear within the BUS and PIN statements.

Description

Specifies the current drawing capability of a pull-up or pull-down device on a pin.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Provides the value of the current in units specified by CURRENT_UNIT.

Example

```
PULL_CURRENT(0.2)
```

Related Statements

CURRENT_UNIT, PULL, PULL_RESISTANCE

PULL_RESISTANCE

Syntax

PULL_RESISTANCE(*value*)

Context

A PULL_RESISTANCE statement can appear within the BUS and PIN statements.

Description

Specifies the resistance of a pull-up or pull-down device on a pin.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max*
Provides the value of the resistance in units specified by RES_UNIT.

Example

```
PULL_RESISTANCE(0.2)
```

Related Statements

PULL, PULL_CURRENT, RES_UNIT

RECOVERY

Syntax

```
RECOVERY (inputPorts {*> | =>} referencePorts
        [COND(cond)
            | COND_START(condStart)| COND_END(condEnd)
            | COND_START(condStart) COND_END(condEnd)]
        [SDF_COND(sdfcond)| SDF_COND_START(sdfStart)
            | SDF_COND_END(sdfEnd)
            | SDF_COND_START(sdfStart) SDF_COND_END(sdfEnd)]
        [input_transition] [pinTrigger]
        model
        )
```

Context

A RECOVERY statement can appear in a CELL statement following the PATH statements.

Description

Provides pin and model information for a delay calculator to calculate the recovery timing check. For a definition of this timing check, see “Recovery” on page 26.

Arguments

inputPorts

Identifies the input pins (usually an asynchronous control pin) to which the timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

*>

Indicates a many-to-many mapping. For example, (A B)*>(Q Q_) means that the specification can be used for the timing checks A to Q, A to Q_, B to Q, and B to Q_.

=>

Indicates a one-to-one mapping. For example, (A B)=>(Q Q_) means that the specification can be used for the timing checks A to Q and B to Q_.

referencePorts

Identifies the reference pins (usually clock pins) to which the timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

COND

Specifies an expression that conditionalizes this recovery timing check. The timing check is evaluated only if the condition is true. For more information on the COND statement see page 6-202.

COND_START

Specifies an expression that conditionalizes the first edge (asynchronous control edge) of the recovery timing check. The timing check is evaluated only if the condition is true. For more information on the COND_START statement see page 6-204.

COND_END

Specifies an expression used to conditionalize the second edge (clock edge) of the recovery timing check. The timing check is evaluated only if the condition is true. For more information on the COND_END statement see page 6-203.

SDF_COND

Specifies an expression to be inserted in the SDF file in the COND construct created for the two ports in the RECOVERY construct. For more information on the SDF_COND statement see page 6-315.

SDF_COND_START

Specifies the condition to be inserted in the SDF file in the COND construct created for the first port (asynchronous control port) of the RECOVERY construct. For more information on the SDF_COND_START statement see page 6-319.

SDF_COND_END

Specifies the condition to be inserted in the SDF file in the COND construct created for the second port (clock port) of the RECOVERY construct. For more information on the SDF_COND_END statement see page 6-317.

input_transition

Specifies the input transition for which this timing information is valid. If not specified, the specified recovery model is used for both transitions.

input_transition	**Use for . . .**
`01`	Positive edge of edge-triggered cells or high value of the reference signal of level-sensitive cells
`10`	Negative edge of edge-triggered cells or low value of the reference signal of level-sensitive cells

pinTrigger

Specifies the condition of the reference pin used to determine the timing check. If not specified, the pin trigger is taken from the `PIN` statement for the reference port.

pinTrigger	**Use for . . .**
`posedge`	Positive-edge-triggered cells
`negEdge`	Negative-edge-triggered cells
`low`	Low-level-sensitive cells
`high`	High-level-sensitive cells

model

Specifies the model information for the timing check. Refer to an existing model, or supply the data in the statement. For valid model parameter values, refer to the `usage_MODEL` statement and Chapter 8, “Examples.”

Examples

```
Recovery(CLR => CLK SDF_Cond_End(clr_or_set)
    Cond_End(CLR || SET) 01 posEdge recModel)
```

REGISTER

Syntax

```
REGISTER(
        CLOCK(clock_condition)
        [SLAVE_CLOCK(clock_condition)]
        INPUT(expression)
        {OUTPUT(pinName...) | INVERTED_OUTPUT(pinName...)|
            OUTPUT(pinName...) INVERTED_OUTPUT(pinName...)}
        [SET(set_condition)]
        [CLEAR(clear_condition)]
        [CLEAR_PRESET_VAR1(value)]
        [CLEAR_PRESET_VAR2(value)]
        )
```

Context

A REGISTER statement can appear in a CELL statement following the PIN statements.

Note: If there are more than one INPUT, CLOCK, OUTPUT, or INVERTED_OUTPUT statements in a register or latch block, the latest definition overwrites all the others. Applications using the TLF parser to read TLF see only one INPUT, CLOCK, OUTPUT, or INVERTED_OUTPUT statement in the register or latch block.

Description

Describes an edge-triggered device.

The sequential-logic behavior must be included for static timing purposes. A static timing analyzer needs to know how separate delays and constraints are related to each other in sequential-logic cells.

Note: This statement does not need to indicate the actual function. It can be an abstracted variant that expresses only the relevant timing relationships between delay paths and timing constraints.

Arguments

CLOCK

Describes when the primary clock is active.

SLAVE_CLOCK
: Describes a secondary clocking signal.

INPUT
: Describes the register data value.

OUTPUT
: Identifies the output pin.

INVERTED_OUTPUT
: Identifies the pins connecting to the inverted outputs of the register.

SET
: Describes an asynchronous set.

CLEAR
: Describes an asynchronous clear.

CLEAR_PRESET_VAR1
: Specifies a value for an OUTPUT pin when CLEAR and SET are both active at the same time.

CLEAR_PRESET_VAR2
: Specifies a value for an INVERTED_OUTPUT pin when CLEAR and SET are both active at the same time.

Example

```
Register(
        Input(D && !HOLD || Q && HOLD)
        Clock(CLK)
        Set(SET)
        Clear (CLR)
        Output(Q)
        )
```

Related Statement

LATCH

REMOVAL

Syntax

```
REMOVAL(inputPorts {*> | =>} referencePorts
        [COND(cond)
            | COND_START(condStart)| COND_END(condEnd)
            | COND_START(condStart) COND_END(condEnd)]
        [SDF_COND(sdfcond)| SDF_COND_START(sdfStart)
            | SDF_COND_END(sdfEnd)
            | SDF_COND_START(sdfStart) SDF_COND_END(sdfEnd)]
        [inputTransition] [pinTrigger]
        model
        )
```

Context

A REMOVAL statement can appear in a CELL statement following the PATH statements.

Description

Provides pin and model information for the removal timing check. For a definition of this timing check, see “Removal” on page 25.

Arguments

inputPorts

Identifies the input pins (usually an asynchronous control pin) to which the timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

*>

Indicates a many-to-many mapping. For example, (A B)*>(Q Q_) means that the specification can be used for the timing checks A to Q, A to Q_, B to Q, and B to Q_.

=>

Indicates a one-to-one mapping. For example, (A B)=>(Q Q_) means that the specification can be used for the timing checks A to Q and B to Q_.

referencePorts

Identifies the reference pins (usually clock pins) to which the timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

COND

Specifies an expression that conditionalizes this removal timing check. The timing check is evaluated only if the condition is true. For more information on the COND statement see page 6-202.

COND_START

Specifies an expression that conditionalizes the first edge (clock edge) of the removal timing check. The timing check is evaluated only if the condition is true. For more information on the COND_START statement see page 6-204.

COND_END

Specifies an expression used to conditionalize the second edge (asynchronous control edge) of the removal timing check. The timing check is evaluated only if the condition is true. For more information on the COND_END statement see page 6-203.

SDF_COND

Specifies an expression to be inserted in the SDF file in the COND construct created for the two ports in the REMOVAL construct. For more information on the SDF_COND statement see page 6-315.

SDF_COND_START

Specifies the condition to be inserted in the SDF file in the COND construct created for the second port (clock port) in the REMOVAL construct. For more information on the SDF_COND_START statement see page 6-319.

SDF_COND_END

Specifies the condition to be inserted in the SDF file in the COND construct created for the first port (asynchronous control port) in the REMOVAL construct. For more information on the SDF_COND_END statement see page 6-317.

inputTransition

Specifies the input transition for which this timing information is

valid. If not specified, the specified removal model is used for both transitions.

inputTransition	Use for . . .
`01`	Positive edge of edge-triggered cells or high value of the reference signal of level-sensitive cells
`10`	Negative edge of edge-triggered cells or low value of the reference signal of level-sensitive cells

`pinTrigger`

Specifies the condition of the reference pin used to determine the timing check. If not specified, the pin trigger is taken from the `PIN` statement for the reference port.

pinTrigger	Use for . . .
`posedge`	Positive-edge-triggered cells
`negEdge`	Negative-edge-triggered cells
`low`	Low-level-sensitive cells
`high`	High-level-sensitive cells

`model`

Specifies the model information for the timing check. Refer to an existing model, or supply the data in the statement. For valid model parameter values, refer to the `usage MODEL` statement and Chapter 8, “Examples.”

Examples

```
Removal(CLRB => CLK removalCLRB)
```

```
Removal(D => CLK SDF_Cond_End(clr_or_set)
    Cond_End(CLR || SET) 01 posEdge removalModel)
```

RISE

Syntax

`RISE(riseValue)`

Context

A `RISE` statement can appear with the following predefined properties:

- CSAT, CSBT, CTTAT, CTTBT
- DEFAULT_SLEW, SLEW_LIMIT, SLEW_MIN
- SC_ENERGY, TOTAL_ENERGY
- PROC_MULT
- TEMP_MULT
- VOLT_MULT

Description

Indicates that the given value applies to falling waveforms only. It is usually accompanied by a FALLstatement.

Arguments

riseValue

Specifies the property value for a rising signal. The value depends on the property.

Example

```
Properties(
        Default_Slew(Rise(2.10:3.10:4.10) Fall(1.20:2.20:3.20))
        Slew_Limit(Warn(Rise(8) Fall(6))
        Error(Rise(12) Fall(10)))
        )
```

Related Statement

FALL

ROUTING_PROPS

Syntax

```
ROUTING_PROPS( layerName
             AVAILABLE_TRACK(...)
             TRACK_AREA(...)
             )
```

Context

A ROUTING_PROPS statement can appear at the CELL level. The ROUTING_PROPS statement is optional and can appear more than once at the cell level. Multiple ROUTING_PROPS statements for the same layer name are not allowed.

Description

A ROUTING_PROPS statement specifies the routing property of a cell on a given layer. Routing properties of a cell are the number of tracks available for routing and the total track area for routing.

Arguments

layerName
: Specifies the name of a layer that was previously defined by the ROUTING_LAYER statement.

AVAILABLE_TRACK
: Specifies the number of tracks available for routing on the layer.

TRACK_AREA
: Specifies the total track area for routing on the layer.

Example

```
ROUTING_PROPS(metal_1
             ...)
```

Also see example for Routing on page 393.

Related Statements

MIN_POROSITY, ROUTING_LAYER

SC_ENERGY

Syntax

SC_ENERGY(*value*)

or

SC_ENERGY(*value* [COND(*cond*)])

or

SC_ENERGY(RISE(*value*) FALL(*value*) [COND(*cond*)])

Note: The COND option applies at the PIN level only.

Context

An SC_ENERGY statement can appear within the CELL, PATH, BUS, and PIN statements.

Description

Specifies the energy consumption due to temporary short circuit current flow.

See the description of SUPPLY_CURRENT for more information.

A single value will be used for both rising and falling signals. The RISE and FALL statements can be used to specify different values for rising and falling signals.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max* | *modelName*
Specifies the value of the short circuit energy. The units for energy are derived by multiplying the specified POWER_UNIT by TIME_UNIT.

cond

Specifies that the short circuit energy value is valid when this condition is true. Use an expression of type *expression*. Refer to "Conditions for Path Delays" on page 50 for more information on this type of expression.

modelName

Specifies the name of a table model. See "Spline or Table Models" on page 46.

Example

```
SC_ENERGY(5.0)
SC_ENERGY(scEnergyModel)
```

Related Statements

GROUND_CURRENT, INTERNAL_ENERGY, SUPPLY_CURRENT, TOTAL_ENERGY

SCAN_EQUIVALENT

Syntax

SCAN_EQUIVALENT(*scanCellName*)

Context

A SCAN_EQUIVALENT statement can appear at the CELL level only.

Description

Used to specify the scan equivalent of a non scan cell.

Arguments

scanCellName
Specifies name of the scan cell.

Example

```
SCAN_EQUIVALENT(ScanCell_abc)
```

SCAN_PINTYPE

Syntax

SCAN_PINTYPE(*usage*)

Context

A SCAN_PINTYPE statement can appear at PIN level.

Description

Defines the pin as a scan pin.

Arguments

usage

```
input | input_inverted | output |
output_inverted | enable | enable_inverted |
clock | clock_a | clock_b | auxiliary_clock
```

input	Specifies an input scan pin.
input_inverted	Specifies an input scan pin having inverted polarity.
output	Specifies an output scan pin.
output_inverted	Specifies an output scan pin having inverted polarity.
enable	Specifies an enable scan pin.
enable_inverted	Specifies an enable scan pin having inverted polarity.
clock	Specifies a clock pin.
clock_a	Specifies a clock pin for master latch.
clock_b	Specifies a clock pin for slave latch.
auxiliary_clock	Specifies an auxiliary clock pin.

Example

```
SCAN_PINTYPE(enable)
```

Related Statements

SCAN_EQUIVALENT, TEST_LATCH, TEST_REGISTER

SDF_COND

Syntax

SDF_COND(*cond_exp*)

Context

An SDF_COND statement can appear in a PATH, PATH_EXTENSION, or any timing_check statement. All of these statements appear at the CELL level.

Description

The SDF_COND statement adds a condition to the SDF statement that corresponds to a TLF PATH or timing check statement. The expression must be true for the path or timing check to be evaluated.

For timing checks that require two ports (such as Setup, Hold, Recovery, Removal, Skew, No_Change), the SDF_Cond statement conditionalizes both ports.

Note: You should specify an SDF_COND statement with a COND statement. Never use an SDF_COND statement with the SDF_COND_START and SDF_COND_END statements.

Arguments

cond_exp

Specifies the condition to be added to the SDF file. Use an expression of type *cond_expr* when used with a PATH statement. Use an expression of type *sdf_timing_check* with a timing check statement. The expression can reference signals that are not pins of the cell. See "Conditional Expressions" on page 50 for more information on this type of expression.

Note: SDF restricts the conditions allowed for timing checks to an identifier, its inverse, or an equality test against a constant. To model conditions that involve more complicated expressions, the library that the SDF annotates must define a local signal that computes the value of the condition. The SDF annotation then refers to the local signal. For this reason, the identifiers in the COND construct (in the SDF file) are not restricted to pin names.

Examples

```
Path(A => Z Cond(!B) SDF_Cond(B==1'b0) 01 10
        Delay(...) Slew(...))
```

The previous TLF statement generates the following SDF statement:

```
(COND B==1'b0 (IOPATH A Z (...) (...)))
```

In the following TLF timing check, the `clr_or_set` signal is the name of an identifier in the simulation model that computes the value of the expression `CLR || SET`.

```
Setup(D => CLK SDF_Cond(clr_or_set)
        Cond(CLR || SET) 01 posEdge ...)
```

The previous TLF statement generates the following SDF statement:

```
(SETUP(COND clr_or_set (posedge D))
         (COND clr_or_set (posedge CLK)) (...))
```

Related Statement

COND

SDF_COND_END

Syntax

SDF_COND_END(*cond_exp*)

Context

An SDF_COND_END statement can appear in SETUP, HOLD, RECOVERY, REMOVAL, SKEW, and NO_CHANGE statements. All of these statements appear at the CELL level.

Description

The SDF_COND_END statement adds a condition to the second edge of a timing check in the corresponding SDF timing check statement. The expression must be true for the timing check to be evaluated.

Note: You should specify an SDF_COND_END statement with a COND_END statement. You can combine an SDF_COND_START statement with an SDF_COND_END statement to specify separate conditions for the starting and ending edges of a timing check. However, you must never use an SDF_COND statement with the SDF_COND_END statement.

Arguments

cond_exp

Specifies the condition for the second edge of a timing check. Use an expression of type *sdf_timing_check*. The expression can reference signals that are not pins of the cell. See "Conditional Expressions" on page 50 for more information on this type of expression.

Note: SDF restricts the conditions allowed for timing checks to an identifier, its inverse, or an equality test against a constant. To model conditions that involve more complicated expressions, the library that the SDF annotates must define a local signal that computes the value of the condition. The SDF annotation then refers to the local signal. For this reason, the identifiers in the COND construct (in the SDF file) are not restricted to pin names.

Examples

The clr_or_set signal name in the following TLF timing checks is an identifier in the simulation model that computes the value of the expression CLR || SET.

```
Setup   (D => CLK SDF_Cond_End(clr_or_set)
        COND_END(CLR || SET) 01 posEdge ...)
Hold    (D => CLK SDF_Cond_End(clr_or_set)
        COND_END(CLR || SET) 01 posEdge ...)
Recovery (CLR => CLK SDF_Cond_End(clr_or_set)
         COND_END(CLR || SET) 01 posEdge ...)
Removal  (CLR => CLK SDF_Cond_End(clr_or_set)
         COND_END(CLR || SET) 01 posEdge ...)
No_Change   (D => CLK SDF_Cond_End(clr_or_set)
            COND_END(CLR || SET) 01 posEdge ...)
Skew    (CLK1 => CLK2 SDF_Cond_End(clr_or_set)
         COND_END(CLR || SET) posEdge posEdge ...)
```

The previous TLF statements generate the following SDF statements:

```
(SETUP (posedge D) (COND clr_or_set (posedge CLK))
        (...))
(HOLD (COND clr_or_set (posedge D)) (posedge CLK)
        (...))
(RECOVERY (posedge CLR) (COND clr_or_set
        (posedge CLK))(...))
(REMOVAL (COND clr_or_set (posedge CLR))
        (posedge CLK) (...))
(NOCHANGE (COND clr_or_set (posedge D))
        (posedge CLK) (...))
(SKEW (posedge CLK1) (COND clr_or_set
        (posedge CLK2)) (...))
```

Related Statement

COND_END

SDF_COND_START

Syntax

SDF_COND_START(*cond_exp*)

Context

An SDF_COND_START statement can appear in SETUP, HOLD, RECOVERY, REMOVAL, SKEW, and NO_CHANGE statements. All of these statements appear at the CELL level.

Description

The SDF_COND_START statement adds a condition to the first edge of a timing check in the corresponding SDF timing check statement. The expression must be true for the timing check to be evaluated.

Note: You should specify an SDF_COND_START statement with a COND_START statement. You can combine an SDF_COND_START statement with an SDF_COND_END statement to specify separate conditions for the starting and ending edges of a timing check. However, you must never use an SDF_COND statement with the SDF_COND_START statement.

Arguments

cond_exp

Specifies the condition for the first edge of a timing check. Use an expression of type *sdf_timing_check*. The expression can reference signals that are not pins of the cell. See "Conditional Expressions" on page 50 for more information on this type of expression.

Note: SDF restricts the conditions allowed for timing checks to an identifier, its inverse, or an equality test against a constant. To model conditions that involve more complicated expressions, the library that the SDF annotates must define a local signal that computes the value of the condition. The SDF annotation then refers to the local signal. For this reason, the identifiers in the COND construct (in the SDF file) are not restricted to pin names.

Examples

The clr_or_set signal name in the following TLF timing checks is an identifier in the simulation model that computes the value of the expression CLR || SET.

```
Setup    (D => CLK SDF_Cond_Start(clr_or_set)
         COND_START(CLR || SET) 01 posEdge ...)
Hold     (D => CLK SDF_Cond_Start(clr_or_set)
         COND_START(CLR || SET) 01 posEdge ...)
Recovery (CLR => CLK SDF_Cond_Start(clr_or_set)
         COND_START(CLR || SET) 01 posEdge ...)
Removal  (CLR => CLK SDF_Cond_Start(clr_or_set)
         COND_START(CLR || SET) 01 posEdge ...)
No_Change(D => CLK SDF_Cond_Start(clr_or_set)
         COND_START(CLR || SET) 01 posEdge ...)
Skew     (CLK1 => CLK2 SDF_Cond_Start(clr_or_set)
         COND_START(CLR || SET) posEdge posEdge ...)
```

The previous TLF statements generate the following SDF statements:

```
(SETUP (COND clr_or_set (posedge D)) (posedge CLK)
        (...))
(HOLD (posedge D) (COND clr_or_set (posedge CLK))
        (...))
(RECOVERY (COND clr_or_set (posedge CLR))
        (posedge CLK) (...))
(REMOVAL (posedge CLR) (COND clr_or_set
        (posedge CLK)) (...))
(NOCHANGE (posedge D) (COND clr_or_set
        (posedge CLK)) (...))
(SKEW (COND clr_or_set (posedge CLK1))
        (posedge CLK2) (...))
```

Related Statement

COND_START

SET

Syntax

SET(*set_condition*)

Context

A SET statement can appear in LATCH and REGISTER statements. The LATCH and REGISTER statements can appear in a CELL statement.

Description

Describes when the output is set high asynchronously.

Arguments

set_condition

Describes the pin conditions leading to an asynchronous set. Use an expression of type *expression*. For more information, see "Conditions for Path Delays" on page 50.

Example

```
Register(
        Input(D && !HOLD || Q && HOLD)
        Clock(CLK)
        Set(SET)
        Clear (CLR)
        Output(Q)
        )
```

Related Statement

CLEAR

SETUP

Syntax

Within a CELL statement:

```
SETUP(inputPorts {*> | =>} referencePorts [arcType]
     [COND(cond)
          | COND_START(condStart)| COND_END(condEnd)
          | COND_START(condStart) COND_END(condEnd)]
     [SDF_COND(sdfcond)| SDF_COND_START(sdfStart)
          | SDF_COND_END(sdfEnd)
          | SDF_COND_START(sdfStart) SDF_COND_END(sdfEnd)]
     [inputTransition] [pinTrigger]
     model
     )
```

```
arcType:NON_SEQ
```

Within a NO_CHANGE statement:

```
SETUP(model)
```

Context

A SETUP statement can appear in a CELL statement, either as a timing_check statement following the PATH statements, or within a NO_CHANGE statement.

Descriptions

The SETUP timing check statement includes pin and model information for a delay calculator to calculate the setup timing check. For a definition of this timing check, see “Setup” on page 22.

When included within a NO_CHANGE timing check statement, the SETUP statement indicates which model is used for the setup portion of the no-change timing check.

Arguments

inputPorts

Identifies the input pins (usually data pins) to which the timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list

with parentheses. If you specify only one pin, you can omit the parentheses.

`*>`

Indicates a many-to-many mapping. For example, `(A B)*>(Q Q_)` means that the specification can be used for the timing checks `A` to `Q`, `A` to `Q_`, `B` to `Q`, and `B` to `Q_`.

`=>`

Indicates a one-to-one mapping. For example, `(A B)=>(Q Q_)` means that the specification can be used for the timing checks `A` to `Q` and `B` to `Q_`.

`referencePorts`

Identifies the reference pins (usually clock) to which the timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

arcType

Specifies the arctype as non-sequential. Non-sequential setup/hold can be used for setup/hold checks on cells that do not contain registers or latches. These are especially relevant for macro-cells that only have timing arcs without any explicit `REGISTER` or `LATCH` statements. In such cases, timing analysis tools extract register or latch information by using setup/hold arcs or clock pins. If a latch is extracted for a non–sequential cell, then timing analysis tool can incorrectly perform time borrowing across this latch. Marking setup/hold as non- sequential helps prevent this kind of false extraction.

`COND`

Specifies an expression that conditionalizes this setup timing check. The timing check is evaluated only if the condition is true. For more information on the `COND` statement see page 6-202.

`COND_START`

Specifies an expression that conditionalizes the first edge (data) of the setup timing check. The timing check is evaluated only if the condition is true. For more information on the `COND_START` statement see page 6-204.

COND_END

Specifies an expression that conditionalizes the second edge (clock) of the setup timing check. The timing check is evaluated only if the condition is true. For more information on the COND_END statement see page 6-203.

SDF_COND

Specifies an expression to be inserted in the SDF file in the COND construct created for the two ports in the SETUP construct. For more information on the SDF_COND statement see page 6-315.

SDF_COND_START

Specifies an expression to be inserted in the SDF file in the COND construct created for the first port (data) in the SETUP construct. For more information on the SDF_COND_START statement see page 6-319.

SDF_COND_END

Specifies an expression to be inserted in the SDF file in the COND construct created for the second port (clock) of the SETUP construct. For more information on the SDF_COND_END statement see page 6-317.

inputTransition

Specifies the input transition for this timing check. If not specified, the timing check applies to both rising and falling transitions.

inputTransition	**Use for . . .**
01	Positive edge of edge-triggered cells or high value of the reference signal of level-sensitive cells
10	Negative edge of edge-triggered cells or low value of the reference signal of level-sensitive cells

pinTrigger

Specifies the condition of the reference pin used to determine

the timing check. If not specified, the pin trigger is taken from the `PIN` statement for the reference port.

pinTrigger	Use for . . .
`posedge`	Positive-edge-triggered cells
`negEdge`	Negative-edge-triggered cells
`low`	Low-level-sensitive cells
`high`	High-level-sensitive cells

`model`

Specifies model information used to calculate the timing check. Refer to an existing model, or supply the data in the statement. For valid parameter values, refer to the `usage MODEL` statement and Chapter 8, “Examples.”

Example

```
Setup   (D => CLK 01 posEdge NON_SEQ SDF_COND(clr)COND(clr)
        TchkRiseModel0
```

Related Statement

`HOLD`

SKEW

Syntax

```
SKEW( (inputPorts) => (referencePorts)
    [pinTrigger] [pinTrigger]
    [COND(cond)
        | COND_START(condStart)| COND_END(condEnd)
        | COND_START(condStart) COND_END(condEnd)]
    [SDF_COND(sdfcond)| SDF_COND_START(sdfStart)
        | SDF_COND_END(sdfEnd)
        | SDF_COND_START(sdfStart) SDF_COND_END(sdfEnd)]
    model
)
```

Context

A SKEW statement can appear in a CELL statement following the PATH statements.

Description

The SKEW statement includes pin and model information for a delay calculator to calculate the skew timing check. For a definition of this timing check, see “Skew” on page 24.

Arguments

inputPorts

Identifies the input pins (used for the stamp event) to which the skew timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the list with parentheses. If you specify only one pin, you can omit the parentheses.

=>

Indicates a one-to-one mapping. For example, (A B)=>(Q Q_) means that the specification can be used to check the skew between pin A and pin Q, and between pin B and pin Q_.

referencePorts

Identifies the reference pins (used for the check event) to which the timing check applies. You must have defined the pins using a PIN statement. If you specify several pins, you must enclose the

list with parentheses. If you specify only one pin, you can omit the parentheses.

pinTrigger

Specifies the condition of the pin used to determine the timing check. The first condition applies to the first pin, while the second condition applies to the second pin. If only one condition is specified, it applies to both pins. If not specified, the pin trigger is taken from the PIN statement for the reference port. Use these pin triggers:

pinTrigger	**Use for . . .**
posedge	Positive-edge-triggered cells
negEdge	Negative-edge-triggered cells
low	Low-level-sensitive cells
high	High-level-sensitive cells

COND

Specifies an expression that conditionalizes this skew timing check. The timing check is evaluated only if the condition is true. For more information on the COND statement see page 6-202.

COND_START

Specifies an expression that conditionalizes the first edge of the skew timing check. The timing check is evaluated only if the condition is true. For more information on the COND_START statement see page 6-204.

COND_END

Specifies an expression that conditionalizes the second edge of the skew timing check. The timing check is evaluated only if the condition is true. For more information on the COND_END statement see page 6-203.

SDF_COND

Specifies an expression to be inserted in the SDF file in the COND construct created for the two ports in the SKEW construct. For more information on the SDF_COND statement see page 6-315.

SDF_COND_START

Specifies an expression to be inserted in the SDF file in the COND construct created for the first port in the SKEW construct. For more information on the SDF_COND_START statement see page 6-319.

SDF_COND_END

Specifies an expression to be inserted in the SDF file in the COND construct created for the second port in the SKEW construct. For more information on the SDF_COND_END construct see page 6-317.

model

Specifies the model information for the timing check. Refer to an existing model or supply the data in the statement. For the valid parameter values, refer to the usage_MODEL statement and Chapter 8, “Examples.”

Examples

```
Skew(CLK_A => CLK_B posEdge posEdge (CONST(1.5)))

Skew(D => CLK SDF_Cond_End(clr_or_set)
        Cond_End(CLR || SET) posEdge posEdge skewMod)
```

SLAVE_CLOCK

Syntax

SLAVE_CLOCK(*clock_condition*)

Context

A SLAVE_CLOCK statement can appear in LATCH and REGISTER statements. The LATCH and REGISTER statements can appear in a CELL statement.

Description

Describes when a secondary clock or latch enable (for registers and latches that require two clocks) is active.

Arguments

clock_condition

Describes the secondary clock or latch enable in terms of the input pin signals. Use an expression of type *expression*. See "Conditions for Path Delays" on page 50 for more information on this type of expression.

Example

```
Slave_Clock(PHI2)
```

Related Statement

CLOCK

SLEW

Syntax

SLEW(*model*)

Context

A SLEW statement can appear in PATH and PATH_EXTENSION statements after the DELAY statement. Both PATH and PATH_EXTENSION can appear within CELL statements.

Description

Provides the data to calculate the output slew of a signal under the conditions listed in the enclosing PATH statement.

Arguments

model

SPecifies either:

- Reference to a previously defined model.
- An inline model consisting of an algorithm clause and parameters. Use the following syntax:

  algorithm
  {[*parameter*](*value*)|(*cond*[*parameter*](*value*))}...

  For valid parameter values, refer to the usage_MODEL statement and Chapter 8, "Examples."

  A slew model (whether named or inline) can contain a single value, a "min::max" pair, or a "min:typ:max" triplet. PVT derating can convert a single value into either a "min::max" pair or a "min:typ:max" triplet.

Examples

```
Slew(td_A_to_Z_rise)
Slew((Const(1.0:2.0:3.0)))
```

Related Statement

DELAY

STATE_FUNCTION

Syntax

```
STATE_FUNCTION(...)
```

Context

A STATE_FUNCTION statement can appear at the PIN level in an output/inout pin of a cell containing statetable.

Description

Describes the functionality of an output/inout pin of a cell containing statetable. The syntax for this statement is same as that of the FUNCTION statement.

Example

```
PIN(Q
    PINTYPE(OUTPUT)
    STATE_FUNCTION(Q1)
    )
```

Related Statements

FUNCTION, TEST_FUNCTION

STATE_TABLE

Syntax

```
STATE_TABLE(stTabName
        (pinList : currentStateList)
        ((pinValue : currentStateValue : nextStateValue)...)
        )
```

Context

A STATE_TABLE statement can appear within a CELL statement. The STATE_TABLE statement is optional and can appear more than once. Each STATE_TABLE statement must specify a unique table name; redefinition of existing tables is not allowed.

Description

Specifies the behavior of a sequential circuit in the form of a state table.

Arguments

stTabName

Specifies the name of the state table. Multiple state tables can exist in a cell as long as each one has a unique name.

pinList

Specifies a list of input pins separated by blank spaces. The pin names specified are dummy pin names. Actual pin names are mapped to these by the MAP_TO_STPIN command.

currentStateList

Specifies a list of current states.

nextStateList

Specifies a list of next states. This has to be the same as *currentStateList.*

pinValue

Specifies values for all the pins mentioned in *pinList*. Specify one of the following values:

Value	Meaning
0	
1	
01	Expands to both 0 and 1
10	Expands to both 1 and 0
R	Rise transition
~R	Not rising edge
F	Fall transition
~F	Not falling edge
-	Don’t care

currentStateValue

Provides values for the current state. You can specify the same value as that of *pinValue* with the addition of Z (High Impedance).

nextStateValue

Specify values for the next state. Specify one of the following:

Value	Meaning
0	
1	
01	Expands to both 0 and 1
10	Expands to both 1 and 0
N	No event from the current value
-	Output not specified
X	Unknown
Z	High impedance

Examples

```
STATE_TABLE(seqTab_1
        (A B : Q)
        ((0 1 : 0 : 0)
        (1 0 : 0 : 0)
        (0 0 : 0 : 1)
        (1 1 : 0 : 1)...
        )
```

The following is an example of a JK flip flop with direct clear (active low) and Negative Edge Clock

```
STATE_TABLE(JK_ff
            (J K CN CD : IQ)
            (- - - 0 : - : 0)
            (- - ~F 1 : - : N)
            (0 0 F 1 : 01 : 01)
            (1 0 F 1 : - : 1)
            (0 1 F 1 : - : 0)
            (1 1 F 1 : 01 : 10)
            )
```

See also the example for 3D Table and Design Rule Checks on page 369.

Related Statements

FUNCTION, MAP_TO_STPIN

SUPPLY_CURRENT

Syntax

SUPPLY_CURRENT(*currentModel*)

or

SUPPLY_CURRENT(*currentModel* COND(*cond*))

Note: The COND option can appear at the BUS and PIN levels only.

or

SUPPLY_CURRENT {RISE(*currentModel*) FALL(*currentModel*))

Note: RISE and FALL are supported at the BUS and PIN levels only.

Context

A SUPPLY_CURRENT statement can appear within the CELL, PATH, BUS, and PIN statements.

Description

Specifies the current drawn from the supply to charge the load connected to a cell. This definition of supply current also includes the short circuit current. Either an average value of the current can be specified or it can be specified as a pulse or waveform.

A cell is said to consume dynamic power when it is active. Dynamic power dissipated has two components:

1. Power dissipated in charging and discharging of the load connected to the cell.
2. Power dissipated in temporary short circuiting of supply and ground while making transition.

Power consumed in charging the load occurs due to current drawn from supply. This current is specified in the library using the statement SUPPLY_CURRENT. Load connected to an output gets charged whenever output makes a rise transition.

Power consumed in discharging the load occurs due to current flow from load to ground. This current is specified in the library using the statement GROUND_CURRENT. Load connected to an output gets discharged whenever output makes a fall transition.

Note: Both the definitions for SUPPLY_CURRENT and GROUND_CURRENT include the short circuit current; and, therefore, the short circuit current is not modeled separately.

Given the current and load information, power can be computed by the power analysis tools. Current can be modeled in four ways:

1. *Average Value* - The construct AVE specifies a single or average value for the current along with the average value for the duration of the measurement.
2. *Triangular Pulse* - The construct TRIPULSE approximates the current to a triangular pulse. Thus, current pulse can be modeled by four parameters: non-zero start time of the rising current, rise time before peak value of current, peak value of the current, fall time after peak value of current.
3. *General Pulse* - The construct GENPULSE approximates the current pulse by a set of linear segments joined together. It is modeled by a series of coordinates.
4. *Wave table* - The *waveTableModel* statement models the current waveform directly.

Supply current and ground current (including the effect of short circuit current) statements specify the average value or current waveforms which are used for calculating the dynamic power consumption. This consumption can also be modeled as energy consumption per transition using the various ENERGY statements directly. Three types of modeling exist for dynamic energy consumption:

1. Modeling only short circuit energy consumption using SC_ENERGY statement: Energy consumed in charging and discharging of the output pin load, wire load and next stage input pin load is calculated by power analysis tool and is not modeled in the library.
2. Modeling energy consumed due to short circuit current and charging and discharging of the output pin load: Since this models the energy consumed within the periphery of the cell, it is called internal energy of the cell and can be specified using the INTERNAL_ENERGY statement. Energy consumed in charging and discharging of the wire load and next stage input pin load is calculated by the power analysis tool and is not modeled in the library.
3. Modeling design-dependent component of energy (consumption due to wire load and next stage input pin load) along with internal energy of the cell as TOTAL_ENERGY: Total energy is the sum of energy consumed due to charging and discharging of wire load, next stage input pin load and the internal energy. In this case, power analysis tools need not compute any component of energy since all the information is available in the library.

If dynamic power has been modeled at path level, it is said to be consumed for the given transitions on a pair of pins. If dynamic power has not been specified for a transition, it is assumed to be zero.

Dynamic power can be modeled either in current form or in energy form for a pin or for a path. However, mixing models is not allowed. If dynamic power has been modeled using energy for a path or for a pin, a current modeling statement cannot be used for the same path or pin.

Arguments

currentModel
: AVE(*value value*) | TRIPULSE(*value value value*) | GENPULSE(*value* (*value value*)... *value*) | *waveTableModel*

AVE(*value value*)
: Specifies a single or average value for the current followed by the average value for the duration of the measurement. For example:
AVE(0.3 2.0)
Here 0.3 is the average current value and 2.0 is the average time value. Units are specified by CURRENT_UNIT and TIME_UNIT.

TRIPULSE(*value value value*)
: Approximates current to a triangular pulse. Thus, current pulse can be modeled by four parameters - non zero start time of the rising current, rise time before peak value of current, peak value of the current, fall time after peak value of current. For example:
TRIPULSE(0.5 (1.1 0.6) 1.4)
where 0.5 is the start time of the 1.1 rise time, 1.1 is the rise time before peak, 0.6 is the current peak value, and 1.4 is the fall time after peak.

GENPULSE(*value* (*value value*)... *value*)
: Approximates current by a set of linear segments joined together. It is modeled by a series of coordinates. For example,
GENPULSE(0.2 (0.4 0.8) (0.6 1.0) (0.8 0.7) 1.0)
where 0.2 is the first time value with a current value of 0.0, each successive pairs are (time, current) pairs, and 1.0 is the last time value with a current value of 0.0.

waveTableModel
: Specifies either the name of a wavetable as previously defined by the WAVETABLE statement or an in-line WAVETABLE model description. A waveform is specified as a set of current and time co-ordinates for each input slew and output load values.

Note: *waveTableModel* is an extension of the spline/table model. In case of a wave table, each table data value describes a waveform in terms of the current and time

coordinates. See WAVETABLE for details.

value

float | *min*::*max* | *min*:*typ*:*max* | *model*

Example

```
PIN(Z
    PINTYPE(...)
    ...
    SUPPLY_CURRENT(TRIPULSE(0.5(1.2 0.65) 1.5) COND(!A & B))
    GROUND_CURRENT(AVE(0.5 2.0))
    )
CELL(bgCr3
    ...
    PATH(A=>B 01 01 DELAY(...) SLEW(...)
        SUPPLY_CURRENT(AVE(0.4 2.0))
        GROUND_CURRENT(AVE(0.3 2.0)) ...)
    PATH(A=>D 01 10 DELAY(...) SLEW(...)
        INTERNAL_ENERGY(...)
        COND(cond) ...)
    PATH(A=>D 01 10 DELAY(...) SLEW(...)
        SUPPLY_CURRENT(WaveTable1)
        GROUND_CURRENT(WaveTable2) ...)
    )
```

Related Statements

GROUND CURRENT, INTERNAL ENERGY, SC ENERGY, TOTAL ENERGY, WAVETABLE

TECHNOLOGY

Syntax

TECHNOLOGY(*technology*)

Context

A TECHNOLOGY statement can appear in the HEADER statement.

Description

Identifies the vendor process for which this library was characterized. This statement is for reference purposes. If more than one TLF file is used, some applications reading TLF have the ability to check the consistency of this parameter across TLF files.

Arguments

technology

Specifies the fabrication process to which the timing data applies. Use a string — a sequence of characters surrounded by double-quotes (").

Example

```
Header(
    Library("cmos500k")
    Technology("CMOS")
    )
```

TEST_FUNCTION

Syntax

```
TEST_FUNCTION(...)
```

Context

A TEST_FUNCTION statement can appear at the PINlevel for non scan output/inout pin of scan cell.

Description

Describes non scan behavior of sequential cells.

Example

```
PIN(Q
    PINTYPE(OUTPUT)
    TEST_FUNCTION(Q1)
    )
```

Related Statements

FUNCTION, STATE_FUNCTION

TEST_LATCH

Syntax

```
TEST_LATCH(
         CLOCK(clock_condition)
         [SLAVE_CLOCK(clock_condition)]
         INPUT(expression)
         {OUTPUT(pinName...)| INVERTED_OUTPUT(pinName...)|
             OUTPUT(pinName...) INVERTED_OUTPUT(pinName...)}
         [SET(set_condition)]
         [CLEAR(clear_condition)]
         [CLEAR_PRESET_VAR1(value)]
         [CLEAR_PRESET_VAR2(value)]
         )
```

Context

A TEST_LATCH statement can appear only at the CELL level.

Description

Describes a level-sensitive (transparent) *non-scan* latch.

Arguments

See the arguments for LATCH.

Example

See the example for LATCH. See also the example for Scan Cell Modeling on page 409.

Related Statements

LATCH, REGISTER, TEST_REGISTER

TEST_REGISTER

Syntax

```
TEST_REGISTER(
              CLOCK(clock_condition)
              [SLAVE_CLOCK(clock_condition)]
              INPUT(expression)
              {OUTPUT(pinName...) |
               INVERTED_OUTPUT(pinName...)|
              OUTPUT(pinName...) INVERTED_OUTPUT(pinName...)}
              [SET(set_condition)]
              [CLEAR(clear_condition)]
              [CLEAR_PRESET_VAR1(value)]
              [CLEAR_PRESET_VAR2(value)]
              )
```

Context

A TEST_REGISTER statement can appear only at the CELL level.

Description

Describes the behavior of a flip flop in *non-scan* mode (normal behavior).

Arguments

See the arguments for REGISTER.

Example

See the example for REGISTER. See also the example for Scan Cell Modeling on page 409.

Related Statements

LATCH, REGISTER, TEST_LATCH

TLF_VERSION

Syntax

`TLF_VERSION(`*`tlf_version`*`)`

Context

A `TLF_VERSION` statement can appear in the `HEADER` statement.

Description

Specifies the version of the TLF language used to create this file. This statement is for reference purposes only.

Arguments

`tlf_version`

Specifies the version of the TLF language used as a string of characters surrounded by double-quotes (for example, `"4.1"`).

Example

```
Header(
    Library("cmos5")
    TLF_Version("4.1")
    )
```

Related Statement

`HEADER`

TOTAL_ENERGY

Syntax

TOTAL_ENERGY(*value*)

or

TOTAL_ENERGY(RISE(value) FALL(value))

or

TOTAL_ENERGY(value COND(cond))

or

TOTAL_ENERGY(RISE(value) FALL(value) COND(cond))

Note: The COND option applies only at the PIN level.

Context

A TOTAL_ENERGY statement can appear within the CELL, PATH, BUS, and PIN statements.

Description

Specifies the total dynamic energy consumption which is the sum of the INTERNAL_ENERGY and the energy consumed due to wire load and next stage input pin load.

See the description of SUPPLY_CURRENT for more information.

A single value is used for both rising and falling signals. The RISE and FALL statements can be used to specify different values for rising and falling signals.

Arguments

value

float | *min*::*max* | *min*:*typ*:*max* | *modelName*

modelName	Specifies the name of a table model. See Spline or Table Models on page 46.

Examples

```
TOTAL_ENERGY(2.0)
TOTAL_ENERGY(0.112 COND(~d[1]))
TOTAL_ENERGY(totalEnergyModel)
```

Related Statements

GROUND_CURRENT, INTERNAL_ENERGY, SC_ENERGY, SUPPLY_CURRENT

TRACK_AREA

Syntax

TRACK_AREA(*value*)

Context

A TRACK_AREA statement can appear within a ROUTING_PROPS statement. A ROUTING_PROPS statement can appear at the CELL level.

Description

Specifies the total track area for routing on a layer and for a cell.

Arguments

value
float

Example

```
TRACK_AREA(0.5)
```

See also the example for Routing on page 393.

Related Statements

MIN_POROSITY, ROUTING_LAYER, ROUTING_PROPS

*usage*_MODEL

Syntax

```
usage_MODEL(modelName [referenceModel]
            (algorithm
                 { [parameter] (value)
                 | (cond [parameter] (value))}...
             )
         )
```

Context

A *usage*_MODEL statement can appear either at the library level before a PROPERTIES statement or within a CELL statement before properties are defined.

Description

Defines the internal timing and power behavior of one or more cells or timing behavior of an interconnect. The *usage* variable indicates the usage where the model can be used. For example, FLUENCE_MODEL specifies that the model is only going to be used in FLUENCE statements.

*usage*_MODEL statements also include the following features:

- Support to three-dimensional table
- Additional axis parameters
- Support to two-dimensional Wire Delay tables

Arguments

usage

```
                    NET_CAP | NET_RES | FLUENCE | ENERGY |
                    TRANSIENT_RES | CURRENT | TIMING | PROCESS_MULT
                    | VOLTAGE_MULT | TEMPERATURE_MULT
```

3D Table

Syntax of the 3D table:

```
usage_Model(modelName [referenceModel]
        (axisParameter indexValues)
        (axisParameter indexValues)
```

```
(axisParameter indexValues) // Third Dimension
(dataValues)...
)
```

axisParameter

Specifies the names of axis variables. The following names are allowed and depend on the usage model:

axis	Axis is general term that can be used in place of other axis (e.g. input_slew_axis, load_axis etc.), however, in this case, there will not be any scaling for the axis values.
`clock_slew_axis`	Specifies the slew on the clock pin.
`input_slew_axis`	Specifies the slew on an input pin.
`load_axis`	Specifies the axis value.
`load2_axis`	In a table reference, specifies the third dimension using `OTHER_PINS`. For example, to refer a 3D table for delay, use the `OTHER_PINS` statement in `PATH`. Values for this pin are specified on the axis mentioned as `load2_axis`.
`output_slew_axis`	Specifies the slew on an output pin. Used for modelling slew degradation tables and two-dimensional wire delay tables.
rc_product_axis	Used for modeling Wire Delay tables.
slew_axis	This is used as input_slew_axis.
`substrate_current_axis`	Specifies the substrate current. It is used in fluence calculation.
`temperature_axis`	Specifies the temperature.
`transition_count_axis`	Specifies the total number of transitions on a pin. It is used in fluence calculation.
`voltage_axis`	Specifies the voltage.

`wire_delay_axis` Specifies the delay across wire as the signal traverses from the output pin to the input pin of next stage. It is used only in slew degradation.

`indexValues`

Specifies the list of axis parameter values to index the table. Indexing to table is performed by running the third dimension faster than the second and similarly by running the second dimension faster than first. Thus the first set of data values corresponds to the first index of the first and second dimensions. While the second set of data values correspond to the first index of the first dimension and the second index of the second dimension.

`dataValues`

Specifies the output values.

The following tables show the allowable axis names for the usage models.

Table 6-1 One-Dimensional *usage*_MODEL Axis Name

***usage*_MODEL**	**Supported Axis Name**
NET_CAP_MODEL, NET_RES_MODEL, TIMING_MODEL, ENERGY_MODEL, TRANSIENT_RES_MODEL, CURRENT_MODEL	axis clock_slew_axis input_slew_axis load_axis

Table 6-2 Two-Dimensional *usage*_MODEL Axis Name

*usage*_MODEL	Supported Axis Name
NET_CAP_MODEL, NET_RES_MODEL, TIMING_MODEL, ENERGY_MODEL, CURRENT_MODEL, FLUENCE_MODEL	axis clock_slew_axis load_axis input_slew_axis output_slew_axis rc_product_axis wire_delay_axis output_slew_axis

Table 6-3 Three-Dimensional *usage*_MODEL Axis Name

*usage*_MODEL	Supported Axis Name
TIMING_MODEL, ENERGY_MODEL	load_axis load2_axis input_slew_axis

Example

Following is an example of a 3D power table. The table models power as the total energy per transition which is a function of load on pin B, load on pin D, and slew on pin A.

```
ENERGY_MODEL(pin_B_power
spline
(load_axis 1.2 2.3)
(load2_axis 2.4 1.3)
(input_slew_axis 1.0 2.0)
((0.43 0.44) (0.48 0.51)
(0.56 0.59) (0.61 0.64))
PATH(A=>B OTHER_PINS(D) 01 10 TOTAL_ENERGY(pin_B_power))
```

See also the examples for Power Modeling with 1D, 2D and 3D Tables on page 373.

Related Statement

WAVEFORM_TAIL_RES, WIRE_DELAY

VENDOR

Syntax

VENDOR(*vendor*)

Context

A VENDOR statement can appear in the HEADER statement.

Description

Identifies the supplier of the models or devices described in the TLF file. This statement is for reference purposes only.

Arguments

vendor

Specifies the name of the supplier. Use a string — a sequence of characters surrounded by double-quotes (").

Example

```
Header(
        Library("cmos500k")
        Vendor("Cadence")
    )
```

VERSION

Syntax

VERSION(*version*)

Context

A VERSION statement can appear in the HEADER statement.

Description

Allows the creator of the TLF file to keep track of different releases of the same library. This statement is for reference purposes only.

Arguments

version

Specifies the version of the TLF file. Use a string — a sequence of characters surrounded by double-quotes (").

Example

```
Header(
        Library("cmos500k")
        Version("5")
    )
```

WARN

Syntax

WARN(*warnValue*)

Context

A WARN statement can appear within the following limit statements at any level in which they appear:

- FANOUT_LIMIT, FANOUT_MIN
- FLUENCE_LIMIT
- LOAD_LIMIT, LOAD_MIN
- SLEW_LIMIT, SLEW_MIN
- VDROP_LIMIT

Description

Specifies a limit. A delay calculator can compare this value with the actual values to determine when it must generate a warning message.

Arguments

warnValue

Specifies the boundary value. You can use a float value, a “min::max” pair, or a “min:typ:max” triplet.

Note: No checking will be done if you either specify a tilde (~) as the value or omit the WARN statement.

WAVETABLE

Syntax

WAVETABLE

Context

The WAVETABLE statement can appear in a usage_MODEL statement after the model name and optional model reference.

Description

The WAVETABLE algorithm is used to describe the current waveform as a set of current and time coordinates for each value of input slew and output loads. It is used for current-based dynamic power analysis. See SUPPLY_CURRENT for more information about dynamic power modeling.

A WAVETABLE is simply an extension of the existing TLF table algorithm (see Spline or Table Models on page 46). In the case of a wave table, each table data value describes a waveform as a set of current and time coordinates.

Note: Although table data values can be of type *float*, *min::max* or *min:typ:max*, the wave table data values (sets of x and y coordinates above) can be of type float only.

Example

```
CURRENT_MODEL(vdd_1
    (WAVETABLE
        (INPUT_SLEW_AXIS 0.1 0.3 0.9 2.7)
        (LOAD_AXIS 0.01 0.03 0.09 0.27)
        (
        // waveform for slew 0.1ns and loadcap 0.01pf
        ((-6.25 0)(-5.61 0)(3.99 0.000167772)
        (6.25 0.000239228)(22.9653 9.70859e-05)
        (40.2198 1.34283e-05)(62.8778 0))
        // waveform for slew 0.1ns and loadcap 0.03pf
        ((-5.61 0)(3.99 0.000167772)(6.25 0.00024718)
        (40.2393 0.000111914)(74.7681 1.69049e-05)
        (98.75 4.97199e-06)(128.75 0))
        // waveform for slew 0.1ns and loadcap 0.09pf (with
        // time as x-axis and current as y-axis)
        // ( (x1 y1) (x2 y2) ....)
```

```
        ...
        )
    )
    ...)
```

See also the example for Wavetable on page 404 for more information.

Related Statements

GROUND_CURRENT, SUPPLY_CURRENT, usage_MODEL

7

Logic Block Templates

This chapter contains the following information:

Introduction

This chapter lists some typical cells used by designers and library builders. For each cell you are given a simple schematic representation and a skeleton of the cell description, including relevant pin and path information.

The cells are categorized in two groups:

- Combinational Logic Blocks
- Sequential Logic Blocks

Important

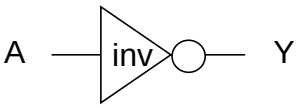
The following conventions are used in the examples:

- Three dots inside parentheses (...) or three dots that do not follow an opening parenthesis “(” indicate that you must supply data.
- Three dots that follow a closing parenthesis “)” indicate that you can repeat the previous construct.

Combinational Logic Blocks

This section describes the following combinational cells:

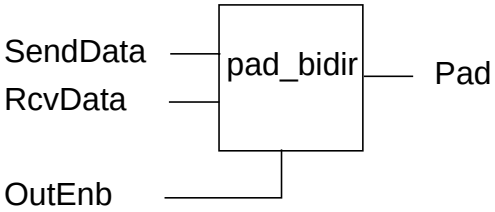
Inverter



TLF Template

```
Cell(inv
    Timing_Model(...)...

    Pin(A Pintype(input)
        Capacitance(...))
    Pin(Y Pintype(output) Function(!A)
        Capacitance(...))

    Path(A => Y 01 10 Delay(...) Slew(...))
    Path(A => Y 10 01 Delay(...) Slew(...))
    )
```

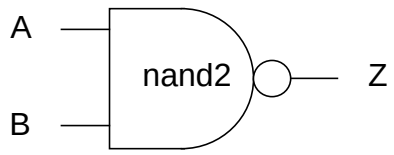
Pad Driver/Receiver



TLF Template

```
Cell(pad_bidir
    Timing_Model(...)...
    Pin(SendData Pintype(input) Capacitance(...))
    Pin(RcvData Pintype(output)
        Function((OutEnb && SendData) || (!OutEnb && Pad))
        Capacitance(...))
    Pin(OutEnb Pintype(input)
        Capacitance(...))
    Pin(Pad Pintype(bidir)
        Function(SendData) Enable(OutEnb)
        Capacitance(...))

    Path(OutEnb => Pad 01 Z0 Delay(...) Slew(...))
    Path(OutEnb => Pad 01 Z1 Delay(...) Slew(...))
    Path(OutEnb => Pad 10 0Z Delay(...) Slew(...))
    Path(OutEnb => Pad 10 1Z Delay(...) Slew(...))
    Path(SendData => Pad 01 01 Delay(...) Slew(...))
    Path(SendData => Pad 10 10 Delay(...) Slew(...))
    Path(Pad => RcvData 01 01 Delay(...) Slew(...))
    Path(Pad => RcvData 10 10 Delay(...) Slew(...))
    Path(SendData => RcvData 01 01 Delay(...) Slew(...))
    Path(SendData => RcvData 10 10 Delay(...) Slew(...))
    )
```

Two-Input NAND

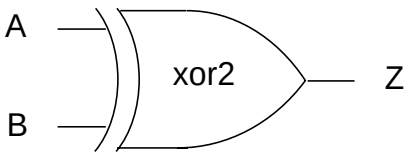


TLF Template

```
Cell(nand2
    Timing_Model(...)...

    Pin(A Pintype(input) Capacitance(...))
    Pin(B Pintype(input) Capacitance(...))
    Pin(Z Pintype(output) Function(!(A && B) Capacitance(...))
```

```
    Path(A => Z 01 10 Delay(...) Slew(...))
    Path(A => Z 10 01 Delay(...) Slew(...))
    Path(B => Z 01 10 Delay(...) Slew(...))
    Path(B => Z 10 01 Delay(...) Slew(...))
    )
```

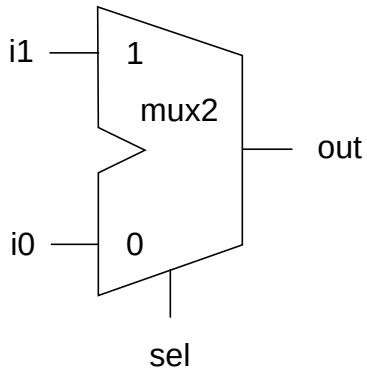
Two-Input XOR



TLF Template

```
Cell(xor2
    Timing_Model(...)...

    Pin(A Pintype(input) Capacitance(...))
    Pin(B Pintype(input) Capacitance(...))
    Pin(Z Pintype(output) Function(A ^ B) Capacitance(...))

    Path(A => Z Cond(B) SDF_Cond(B==1'b1) 01 10 Delay(...)Slew(...))
    Path(A => Z Cond(B) SDF_Cond(B==1'b1) 10 01 Delay(...)Slew(...))
    Path(A => Z Cond(!B) SDF_Cond(B==1'b0) 01 01 Delay(...)Slew(...))
    Path(A => Z Cond(!B) SDF_Cond(B==1'b0) 10 10 Delay(...)Slew(...))
    Path(B => Z Cond(A) SDF_Cond(A==1'b1) 01 10 Delay(...)Slew(...))
    Path(B => Z Cond(A) SDF_Cond(A==1'b1) 10 01 Delay(...)Slew(...))
    Path(B => Z Cond(!A) SDF_Cond(A==1'b0) 01 01 Delay(...)Slew(...))
    Path(B => Z Cond(!A) SDF_Cond(A==1'b0) 10 10 Delay(...)Slew(...))
    )
```

Two-Input Multiplexer

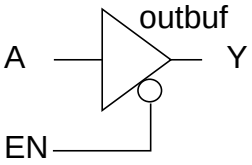


TLF Template

```
Cell(mux2
    Timing_Model(...)...

    Pin(i0 Pintype(input) Capacitance(...))
    Pin(i1 Pintype(input)Capacitance(...))
    Pin(sel Pintype(input) Capacitance(...))
    Pin(out Pintype(output)
        Function((~sel && i0)||(sel && i1))
        Capacitance(...))

    Path(i0 => out 01 01 Delay(...) Slew(...))
    Path(i0 => out 10 10 Delay(...) Slew(...))
    Path(i1 => out 01 01 Delay(...) Slew(...))
    Path(i1 => out 10 10 Delay(...) Slew(...))


    Path(sel => out Cond(~i0) SDF_Cond(i0==1'b0) 01 01
            Delay(...) Slew(...))
    Path(sel => out Cond(~i0) SDF_Cond(i0==1'b0) 01 10
            Delay(...) Slew(...))
    Path(sel => out Cond(~i1) SDF_Cond(i1==1'b0) 10 10
            Delay(...) Slew(...))
    Path(sel => out Cond(~i1) SDF_Cond(i1==1'b0) 10 01
            Delay(...) Slew(...))
    )
```

Tristate Driver

Buffer with active low enable



TLF Template

```
Cell(outbuf
   Timing_Model(...)...

    Pin(A Pintype(input) Capacitance(...))
    Pin(EN Pintype(input) Capacitance(...))
    Pin(Y Pintype(output) Function(A)
        Enable(!EN) Capacitance(...))

    Path(A => Y 01 01 Delay(...) Slew(...))
    Path(A => Y 10 10 Delay(...) Slew(...))
    Path(EN => Y 01 0Z Delay(...) Slew(...))
    Path(EN => Y 10 Z1 Delay(...) Slew(...))
    Path(EN => Y 10 Z0 Delay(...) Slew(...))
    Path(EN => Y 01 1Z Delay(...) Slew(...))
    )
```

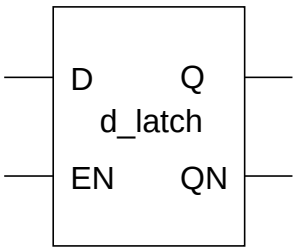
Sequential Logic Blocks

This section describes the following sequential cells:

- Latch
- D Flip-Flop with Buffered Outputs
- D Flip-Flop with Unbuffered Outputs
- Random Access Memory

Latch

Buffered, complementary output latch with gated hold



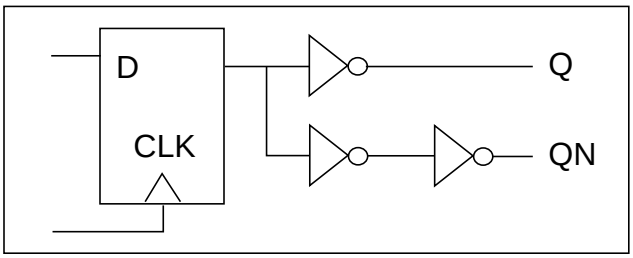
TLF Template

```
Cell(d_latch
    Timing_Model(...)...

    Pin(D Pintype(input) Capacitance(...))
    Pin(EN Clock_pin Pintype(input) Capacitance(...))
    Pin(Q Pintype(output) Capacitance(...))
    Pin(QN Pintype(output) Capacitance(...))
    Latch(
        Clock(EN)
        Input(D)
        Output(Q)
        Inverted_Output(QN)
   )

    Path(D => Q 01 01 Delay(...) Slew(...))
    Path(D => Q 10 10 Delay(...) Slew(...))
    Path(D => QN 01 10 Delay(...) Slew(...))
    Path(D => QN 10 01 Delay(...) Slew(...))
    Path(EN => Q 01 01 Delay(...) Slew(...))
    Path(EN => Q 01 10 Delay(...) Slew(...))
    Path(EN => QN 01 01 Delay(...) Slew(...))
    Path(EN => QN 01 10 Delay(...) Slew(...))

    Setup(D => EN 01 high ...)
    Setup(D => EN 10 high ...)
    Hold(D => EN 01 high ...)
    Hold(D => EN 10 high ...)
    )
```

D Flip-Flop with Buffered Outputs

Positive edge-triggered D flip-flop with buffered complementary outputs

dff_buf



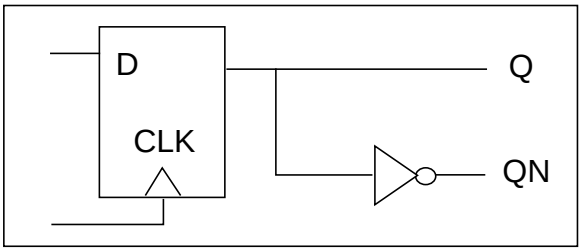
TLF Template

```
Cell(dff_buf
    Timing_Model(...)...

    Pin(D Pintype(input) Capacitance(...))
    Pin(CLK Clock_pin Pintype(input) Capacitance(...))
    Pin(Q Pintype(output)  Capacitance(...))
    Pin(QN Pintype(output) Capacitance(...))

    Register(
        Input(D)
        Clock(CLK)
        Output(Q)
        Inverted_Output(QN)
    )

    Path(CLK => Q 01 01 Delay(...) Slew(...))
    Path(CLK => Q 01 10 Delay(...) Slew(...))
    Path(CLK => QN 01 10 Delay(...) Slew(...))
    Path(CLK => QN 01 01 Delay(...) Slew(...))

    Setup(D => CLK 01 posEdge ...)
    Setup(D => CLK 10 posEdge ...)
    Hold(D => CLK 01 posEdge ...)
    Hold(D => CLK 10 posEdge ...)
    )
```

D Flip-Flop with Unbuffered Outputs

Positive edge-triggered D flip-flop with complementary outputs

dff

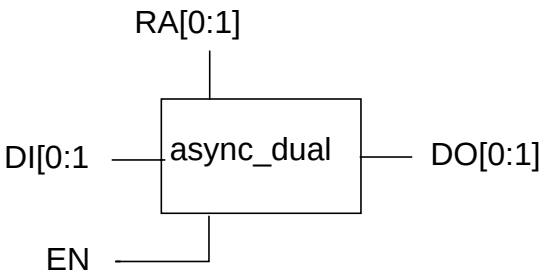


TLF Template

```
Cell(dff
    Timing_Model(...)...

    Pin(D Pintype(input) Capacitance(...))
    Pin(CLK Clock_pin Pintype(input) Capacitance(...))
    Pin(Q Pintype(output) Capacitance(...))
    Pin(QN Pintype(output) Function(~Q) Capacitance(...))
    Register(
        Input(D)
        Clock(CLK)
        Output(Q)
    )
    Path(CLK => Q 01 01 Delay(...) Slew(...))
    Path(CLK => Q 01 10 Delay(...) Slew(...))

    Path_Extension(Q => QN 01 10 Delay(...) Slew(...))
    Path_Extension(Q => QN 10 01 Delay(...) Slew(...))

    Setup(D => CLK 01 posEdge ...)
    Setup(D => CLK 10 posEdge ...)
    Hold(D => CLK 01 posEdge ...)
    Hold(D => CLK 10 posEdge ...)
    )
```

Random Access Memory

Synchronous 2x2 random access memory



TLF Template

```
Cell(async_dual
     Timing_Model(...)...
     )
     Pin(EN Pintype(Input) Clock_pin Capacitance(0.280000))
     Bus(RA[0:1]
          Pintype(Input)
          Capacitance(0.241000)
     )
     Bus(DI[0:1]
          Pintype(Input)
          Capacitance(0.144000)
     )
     Bus(DO[0:1]
          Pintype(Output)
          Capacitance(0.097000)
     )
     Memory_Props(
          Memory_type(RAM)
          Memory_opr(Synchronous)
          Address_Width(2)
          Data_Width(2)
     )
     Memory_Bus(DI
               Busmode(write)
               Address_Bus(RA)
               Clock(~(EN))
          )
     Memory_Bus(DO
               Busmode(Read)
               Address_Bus(RA)
          )
```

```
Path(RA[0:1] *> DO[0:1]  01 01 Delay(ioDelayRiseModel0)
    Slew(SlopeRiseModel0))
Path(RA[0:1] *> DO[0:1]  10 01 Delay(ioDelayRiseModel0)
    Slew(SlopeRiseModel0))
Path(RA[0:1] *> DO[0:1]  01 10 Delay(ioDelayFallModel0)
    Slew(SlopeFallModel0))
Path(RA[0:1] *> DO[0:1]  10 10 Delay(ioDelayFallModel0)
    Slew(SlopeFallModel0))
Setup(RA[0:1] *> EN  Rise Negedge (Const(1.7)))
Setup(RA[0:1] *> EN  Fall Negedge (Const(4.2)))
Setup(DI[0:1] *> EN  Rise Posedge (Const(1.7)))
Setup(DI[0:1] *> EN  Fall Posedge (Const(1.1)))
)
```

8

Examples

This chapter contains the both the Synopsis and TLF equivalent description for the following constructs as applicable:

3D Table and Design Rule Checks

The example is presented as “Example 1: Synopsys Description” and “Example 1: TLF Equivalent” on page 371.

Example 1: Synopsys Description

```
/* For Operating conditions, 3D tables, statetable, designrulecheck
*/
library ( PVT_3D_stateTable_designRuleCheck )
    {
    technology ( cmos ) ;
    delay_model : table_lookup ;

    lu_table_template(templateThreeD)
    {
    variable_1 : related_pin_transition;
    index_1 ("0.0300, 0.8100");
    variable_2 : constrained_pin_transition;
    index_2 ("0.0300, 0.8100");
    variable_3 : related_out_total_output_net_capacitance;
    index_3 ("0.0080, 0.0160");
    }

    operating_conditions("T125_V4_P1")
    {
    process : 1;
    voltage : 4 ;
    temperature : 125 ;
    }

    operating_conditions("T70_V4_P05")
    {
    process : 0.5;
    voltage : 4 ;
    temperature : 70 ;
    }

    default_max_fanout : 10;
    default_fanout_load : 10;
    default_operating_conditions : T125_V4_P1;

    cell ( LD1S2QA )
    {
```

```
statetable("SCK2 SI  CP  D SCK1 ", " Q1   Q2")
{
table : "L  -  -  -  - :  -  - :  N  - , /* master inactive*/ \
       H  L  -  -  - :  -  - :  L  - , /* master loads SI*/ \
       -  -  H  L  L :  -  - :  L  L/H , /* slave loads D */\
       -  -  H  -  H :  -  - :  L  X , /* illegal clocking*/ \
       -  -  L  -  H :  H  - :  L  L  " /* slave loads master*/ ;
}

pin ( Q )
{
direction : output;
capacitance : 0.000000;
min_fanout : 2;
min_transition : 1.0;
min_capacitance : 1.0;
internal_node : "Q1";
input_map : "SCK2 SI CP D SCK1";
timing ( )
    {
    ...
    }
}

pin ( D )
{
direction : input;
capacitance : 0.017000;
fanout_load : 1.0;
timing ( )
        {
        related_pin : "CP" ;
        related_output_pin : "Q";
        timing_type : setup_falling;
        rise_constraint (templateThreeD)
            {
            values ("0.9525, 1.2839", "0.9528, 1.2850", \
            "0.7396, 1.0710", "0.9623, 1.4876")
            }
        fall_constraint (templateThreeD)
            {
            values ("0.9366, 1.2142", "0.9567, 1.2567",\
             "0.7237, 1.0013", "0.9876, 1.5643")
            }
        }
timing ( )
        {
```

```
            related_pin : "CP" ;
            related_output_pin : "Q";
            timing_type : hold_falling;
            rise_constraint (templateThreeD)
                {
                values ("0.0641, -0.2257", "0.0723, -0.123", \
                "0.2771, -0.0128", "0.4530, -0.0112")
                }
            fall_constraint (templateThreeD)
                {
                values ("0.0598, -0.1792", "0.0612, -0.1812", \
                "0.2727, 0.0337", "0.2936, 0.0412")
                }
            }
    }

    pin ( CP )
        {
        direction : input;
        capacitance : 0.020000;
        clock : true ;
        }
    }
```

Example 1: TLF Equivalent

```
Header(
    Library("PVT_3D_stateTable_designRuleCheck")
    Technology("cmos")
    TLF_Version("4.1")
    )

PROPERTIES(
    For_Pin(OutputFanout_Limit(10.000000))
    For_pin(Input Input_Fanload(10.000000))
    PVT_Conds(T125_V4_P1 Proc_Var(1) Voltage(4) Temperature(125))
    PVT_Conds(T70_V4_P05 Proc_Var(0.5) Voltage(4) Temperature(70))
    Default_PVT_Cond(T125_V4_P1)
    )

Cell(LD1S2QA

    Timing_Model(templateThreeDDCCellMod
        (Spline
            (Clock_Slew_Axis 0.030000 0.810000)
            (Input_Slew_Axis 0.030000 0.810000)
```

```
            (Output_Load_Axis 0.00800 0.016000)
            data()
        )
    )



Timing_Model(TchkRiseModel0 templateThreeDDCCellMod
        (Spline
            data(
                ((0.952500, 1.283900) (0.9528, 1.2850))
                ((0.739600, 1.071000) (0.9623, 1.4876))
            )
        )
    )

    Timing_Model(TchkFallModel0 templateThreeDDCCellMod
        (Spline
            data(
                ((0.936600, 1.214200) (0.9567, 1.2567))
                ((0.723700, 1.001300) (0.9876, 1.5643))
            )
        )
    )

    Timing_Model(TchkRiseModel1 templateThreeDDCCellMod
        (Spline
            data(
                ((0.064100, -0.225700) (0.0723, -0.123))
                ((0.277100, -0.012800) (0.4530, -0.0112))
            )
        )
    )

    Timing_Model(TchkFallModel1 templateThreeDDCCellMod
        (Spline
            data(
                ((0.059800, -0.179200) (0.0612, -0.1812))
                ((0.272700, 0.033700) (0.2936, 0.0412))
            )
        )
    )

State_Table(stTab
        (SCK2 SI  CP  D SCK1 : Q1 Q2)
        ((0 - - - - : - -: N -)
```

```
          (1 0 - - - : - -: 0 -)
          (- - 1 0 0 : - -: 0 01)
          (- - 1 - 1 : - -: 1 X)
          (- - 0 - 1 : 1 -: 0 0)
        )
    )


Pin(D Pintype(Input) Capacitance(0.017000) Input_fanout(1.000000)))
    Pin(CP Pintype(Input) Clock_pin Capacitance(0.020000))
    Pin(SCK1 Pintype(Input) Clock_pin Capacitance(0.027000)))
    Pin(SCK2 Pintype(Input) Clock_pin Capacitance(0.025000)))
    Pin(SI Pintype(Input) Capacitance(0.018000)))
Pin(Q Pintype(Output)
Map_to_stpin(stTab (SCK2 SI CP D SCK1 : Q -))
Capacitance(0.000000) Fanout_Min(1.000000) Load_Min(1.000000)
Slew_Min(1.000000))

Setup(D => CP Other_Pin(Q) Rise Negedge TchkRiseModel0)
    Setup(D => CP Other_Pin(Q) Fall Negedge TchkFallModel0)
    Hold(D => CP Other_Pin(Q) Rise Negedge TchkRiseModel1)
    Hold(D => CP Other_Pin(Q) Fall Negedge TchkFallModel1)
)
```

Power Modeling with 1D, 2D and 3D Tables

The example is presented as "Example 2: Synopsys Description" on page 373 and "Example 2: TLF Equivalent" on page 376.

Example 2: Synopsys Description

```
/* For Power Modelling using 1D,2D and 3D tables */
library(power) {
    leakage_power_unit : 1nW;
    default_cell_leakage_power : 0.1;
    k_volt_cell_leakage_power : 1.000000;
    k_volt_internal_power : 2.000000;

    power_lut_template(output_by_cap1_cap2_and_trans) {
        variable_1 : total_output1_net_capacitance ;
        variable_2 : total_output2_net_capacitance ;
        variable_3 : input_transition_time ;
        index_1 ("0.0, 5.0 ") ;
        index_2 ("0.0, 5.0 ") ;
```

```
        index_3 ("0.0, 1.0 ") ;
    }
    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");
        index_2 ("0.0, 1.00");
    }
    power_lut_template(output_by_cap) {
        variable_1 : total_output_net_capacitance ;
        index_1("0.0, 5.00") ;
    }
    power_lut_template(input_by_trans) {
        variable_1 : input_transition_time ;
        index_1 ("0.0, 1.00") ;
    }
    cell(AN2) {
         cell_leakage_power : 0.2;
         leakage_power () {
            when : "A" ;
            values ("2.0") ;
         }
         pin(Z) {
            direction : output ;
            internal_power {
                power(output_by_cap_and_trans) {
                    values ("2.2, 3.73", "1.7, 2.15");
                }
                related_pin : "A B" ;
            }
            timing() {
...
            }
        }
pin(A) {
            direction : input ;
        }
        pin(B) {
            direction : input ;
        }

    }
    cell(FLIPFLOP1) {
        pin(CP) {
            direction : input ;
            internal_power() {
                power(input_by_trans) {
```

```
                    values("2.2, 3.7");
                }
            }
        }
        pin(D) {
            direction : input ;
        }
        pin(Q) {
            direction : output ;
            internal_power() {
                power(output_by_cap) {
                    values("2.2, 3.7")
                }
            }
        }
    }
cell(FLIPFLOP2) {
        pin(CP) {
            direction : input ;
        }
        pin(D) {
            direction : input ;
        }
        pin(Q) {
            direction : output ;
            internal_power() {
                rise_power(output_by_cap1_cap2_and_trans) {
                    values("2.2, 3.7", "1.7, 2.15", \
                    "2.1, 3.62", "1.6, 2.04")
                }
                fall_power(output_by_cap1_cap2_and_trans) {
                    values("2.2, 3.7", "1.7, 2.15", \
                    "2.1, 3.62", "1.6, 2.04")
                }
                equal_or_opposite_output : "QN" ;
                related_pin : "CP" ;
}
        }
        pin(QN) {
            direction : output ;
        }
    }
}
```

Example 2: TLF Equivalent

```
Header(
    Library("power")
    TLF_Version("4.1")
    Generated_By("UMAM")
)

Energy_Model(output_by_cap1_cap2_and_trans
    (Spine
        (Load_Axis 0.000000 5.000000)
        (Load2_Axis 0.000000 5.000000)
        (Input_Slew_Axis 0.000000 1.000000)
    )
)

Energy_Model(output_by_cap_and_trans
    (Spine
        (Load_Axis 0.000000 5.000000)
        (Input_Slew_Axis 0.000000 1.000000)
    )
)

Energy_Model(output_by_cap
    (Spine
        (Load_Axis 0.000000 5.000000)
    )
)

Energy_Model(input_by_trans
    (Spine
        (Input_Slew_Axis 0.000000 1.000000)
    )
)

PROPERTIES(
    Unit(
        Power_Unit(1nW)
    )
    Cell_SPower(0.1)
    Volt_Mult_CSPower(1.0)
    Volt_Mult_Ienergy(2.0)
)
Cell(AN2

    Energy_Model(2DTable output_by_cap_and_trans
        (Spline
```

```
            data(
                (2.2 3.73)
                (1.7 2.15)
            )
        )
    )

        Cell_SPower(0.2)
        Cell_SPower(2.0 COND(A))


    Pin(Z Pintype(Output))
    Pin(A Pintype(Input))
    Pin(B Pintype(Input))

    Path((A B) *> Z 01 01 Delay(...) Slew(...)
INTERNAL_ENERGY(AVE(2DTable)))
    Path((A B) *> Z 10 10 Delay(...) Slew(...)
INTERNAL_ENERGY(AVE((2DTable)))
    Path((A B) *> Z 10 10 COND(...) SDF_COND(...) Delay(...)
Slew(...) INTERNAL_ENERGY(2DTable))
)

Cell(FLIPFLOP1

    Energy_Model(1DTable_1 input_by_trans
        (Spline
            data(
                (2.2 3.7)
            )
        )
    )

    Energy_Model(1DTable_2 output_by_cap
        (Spline
            data(
                (2.2 3.7)
            )
        )
    )

    Pin(CP Pintype(Input) INTERNAL_ENERGY(AVE(1DTable_1)))
    Pin(D Pintype(Input))
    Pin(Q Pintype(Output) INTERNAL_ENERGY(AVE(1DTable_2)))
)
Cell(FLIPFLOP2
```

```
    Energy_Model(3DTable_rise output_by_cap1_cap2_and_trans
        (Spline
            data(
                ((2.2, 3.7) (1.7 2.15))
                ((2.1 3.62) (1.6 2.04))
            )
        )
     )

    Energy_Model(3DTable_fall output_by_cap1_cap2_and_trans
        (Spline
            data(
                ((2.2 3.7) (1.7 2.15))
                ((2.1 3.62) (1.6 2.04))
            )
        )
     )

    Pin(CP Pintype(Input))
    Pin(D Pintype(Input))
    Pin(Q Pintype(Output))
    Pin(QN Pintype(Output))

    Path(CP => Q 01 01 Other_Pin(QN)
INTERNAL_ENERGY(AVE(3DTable_rise)))
    Path(CP => Q 01 10 Other_Pin(QN)
INTERNAL_ENERGY(AVE(3DTable_fall)))
)
```

Memory (Asynch 2x2 RAM with Dual Port)

This example is presented as “Example 3: Synopsys Description” and “Example 3: TLF Equivalent” on page 381.

Example 3: Synopsys Description

```
/* Async 2X2 RAM with dual port */
library (RAM) {
    type(bus2) {
        base_type : array;
        data_type : bit;
        bit_width : 2;
        bit_from : 0;
        bit_to : 1;
```

```
        downto : false;
    }

  cell(async_dual) {
     memory() { /* Indicates this is a memory cell */
        type : ram;
        address_width : 2;
        word_width : 2;
    }
     bus (RA) {
        bus_type  : "bus2";
        direction : input;
        capacitance : 0.241;
        timing () { /*Address setup time */
            timing_type : setup_falling;
            intrinsic_rise : 1.7;
            intrinsic_fall : 4.2;
            related_pin : "OE";
          }
         timing () { /* Address hold time */
            timing_type : hold_rising;
            intrinsic_rise : 0.11;
            intrinsic_fall : 0.23;
            related_pin : "OE";
          }
    }
     bus (WA) {
        bus_type  : "bus2";
        direction : input;
        capacitance : 0.241;
        timing () { /*Address setup time */
            timing_type : setup_falling;
            intrinsic_rise : 1.7;
            intrinsic_fall : 4.2;
            related_pin : "WR";
          }
         timing () { /* Address hold time */
timing_type : hold_rising;
            intrinsic_rise : 0.11;
            intrinsic_fall : 0.23;
            related_pin : "WR";
          }
     }
bus (DI) {
         bus_type: "bus2"
         direction : input;
         capacitance : 0.144;
```

```
          memory_write() { /* Indicate this is a write port */
             address : WA;
             enable : WR;
           }
          timing () { /* Input data setup time */
             timing_type : setup_rising;
             intrinsic_rise : 1.7;
             intrinsic_fall : 1.1;
             related_pin : "WR";
           }
           timing () { /* Input Data hold time */
              timing_type : hold_rising;
              intrinsic_rise : 0.11;
              intrinsic_fall : 0.12;
              related_pin : "WR";
           }
    }
     pin (WR) {
        direction : input;
        capacitance : 0.28;
        clock : true; /* WR pulse is modeled as a clock */
    }
     pin (OE) {
        direction : input;
        capacitance : 0.28;
    }
     bus(DO){
        bus_type  : "bus2";
        direction : output;
        capacitance : 0.097;
        memory_read() { /* Indicates that this is a read port */
            address : RA;
        }
        three_state : "OE";
        timing () {
            timing_sense : non_unate; /* combinational delay */
            intrinsic_rise : 5.15;
            intrinsic_fall : 4.51;
            rise_resistance : 0.020;
            fall_resistance : 0.017;
            related_bus_pins : "RA";
        }
}
  }
}
```

Example 3: TLF Equivalent

```
Header(
    Library("RAM")
    Date("Fri Feb 13 10:16:02 1998")
    TLF_Version("4.1")
    Generated_By("UMAM")
)

Cell(async_dual

    Timing_Model(ioDelayRiseModel0
        (Spline
            (Load_Axis 0 1.000000)
            (Input_Slew_Axis 0 1.000000)
            data(
                (5.150000 5.150000)
                (5.170000 5.170000)
            )
        )
    )

    Timing_Model(ioDelayFallModel0
        (Spline
            (Load_Axis 0 1.000000)
            (Input_Slew_Axis 0 1.000000)
            data(
                (4.510000 4.510000)
                (4.527000 4.527000)
            )
        )
    )

    Timing_Model(SlopeRiseModel0
        (Spline
            (Load_Axis 0 1.000000)
            data(
                (0 0.020000)
            )
        )
    )

    Timing_Model(SlopeFallModel0
        (Spline
            (Load_Axis 0 1.000000)
            data(
                (0 0.017000)
```

```
            )
        )
    )
Pin(WR Pintype(Input) Clock_pin Capacitance(0.280000))
    Pin(OE Pintype(Input) Capacitance(0.280000))
    Bus(RA[0:1]
        Bustype(Input)
Capacitance(0.241000)
    )
    Bus(WA[0:1]
        Bustype(Input)
            Capacitance(0.241000)
    )
    Bus(DI[0:1]
        Bustype(Input)
Capacitance(0.144000)
    )
    Bus(DO[0:1]
        Bustype(Output) Enable(~(OE))
            Capacitance(0.097000)
    )

    Memory_Props(
        Memory_type(RAM)
        Memory_opr(Asynchronous)
        Address_Width(2)
        Data_Width(2)
    )

    Memory_Bus(DO
            Busmode(Read)
            Address_Bus(RA)
        )
    Memory_Bus(DI
            Busmode(Write)
            Address_Bus(WA)
            Enable(WR)
        )


Path(RA[0:1] *> DO[0:1]  01 01 Delay(ioDelayRiseModel0) Slew(SlopeR
iseModel0))

Path(RA[0:1] *> DO[0:1]  10 01 Delay(ioDelayRiseModel0) Slew(SlopeR
iseModel0))
```

```
Path(RA[0:1] *> DO[0:1]  01 10 Delay(ioDelayFallModel0) Slew(SlopeF
allModel0))

Path(RA[0:1] *> DO[0:1]  10 10 Delay(ioDelayFallModel0) Slew(SlopeF
allModel0))
Setup(RA[0:1] *> OE  Rise Negedge (Const(1.7)))
    Setup(RA[0:1] *> OE  Fall Negedge (Const(4.2)))
    Hold(RA[0:1] *> OE  Rise Posedge (Const(0.11)))
    Hold(RA[0:1] *> OE  Fall Posedge (Const(0.23)))
    Setup(WA[0:1] *> WR  Rise Negedge (Const(1.7)))
    Setup(WA[0:1] *> WR  Fall Negedge (Const(4.2)))
    Hold(WA[0:1] *> WR  Rise Posedge (Const(0.11)))
Hold(WA[0:1] *> WR  Fall Posedge (Const(0.23)))
    Setup(DI[0:1] *> WR  Rise Posedge (Const(1.7)))
    Setup(DI[0:1] *> WR  Fall Posedge (Const(1.1)))
    Hold(DI[0:1] *> WR  Rise Posedge (Const(0.11)))
    Hold(DI[0:1] *> WR  Fall Posedge (Const(0.12)))
)
```

Memory (ROM cell)

This example is presented as “Example 4: Synopsys Description” and “Example 4: TLF Equivalent” on page 384.

Example 4: Synopsys Description

```
library (read_only_memory) {
    type(bus4) {
      base_type : array;
      data_type : bit;
      bit_width : 4;
      bit_from : 0;
      bit_to : 3;
      downto : false;
    }

    cell (rom) {
        memory() {
        type : rom;
        address_width : 4;
        word_width : 4;
      }
          bus (ADDR) {
```

```
                bus_type  : "bus4";
                direction : input;
                capacitance : 1.46;
                timing () {
                    timing_type : setup_falling;
                    intrinsic_rise : 3.20;
                    intrinsic_fall : 3.20;
                    related_pin : "CLK";
                }
            }
            pin (CLK) {
                direction : input;
                capacitance : 1.13;
                clock : true;
            }
            bus(QO){
                bus_type  : "bus4";
                direction : output;
                memory_read() {
                     address : ADDR;
                }
                timing () {
                        timing_sense : non_unate;
                intrinsic_rise : 5.25;
                rise_resistance : 0.020;
                intrinsic_fall : 5.50;
                fall_resistance : 0.017;
                related_bus_pins : "ADDR";
                }
        }
     }
}
```

Example 4: TLF Equivalent

```
Header(
    Library("read_only_memory")
    TLF_Version("4.1")
    Generated_By("UMAM")
)

Cell(rom

    Timing_Model(ioDelayRiseModel0
        (Spline
            (Load_Axis 0 1.000000)
```

```
            (Input_Slew_Axis 0 1.000000)
            data(
                (5.250000 5.250000)
                (5.270000 5.270000)
            )
        )
    )

    Timing_Model(ioDelayFallModel0
        (Spline
            (Load_Axis 0 1.000000)
            (Input_Slew_Axis 0 1.000000)
            data(
                (5.500000 5.500000)
                (5.517000 5.517000)
            )
        )
    )

    Timing_Model(SlopeRiseModel0
        (Spline
            (Load_Axis 0 1.000000)
            data(
                (0 0.020000)
            )
        )
    )

    Timing_Model(SlopeFallModel0
        (Spline
            (Load_Axis 0 1.000000)
            data(
                (0 0.017000)
            )
        )
    )

    Pin(CLK Pintype(Input) Clock_pin Capacitance(1.130000))
    Bus(ADDR[0:3]
        Bustype(Input)
Capacitance(1.460000)
    )
    Bus(QO[0:3]
Bustype(Output)
    )

    Memory_Props(
```

```
        Memory_type(ROM)
        Address_Width(4)
        Data_Width(4)
    )


    Memory_Bus(Q0
            Busmode(Read)
            Address_Bus(ADDR)
        )



Path(ADDR[0:3] *> QO[0:3]  01 01 Delay(ioDelayRiseModel0) Slew(Slop
eRiseModel0))

Path(ADDR[0:3] *> QO[0:3]  10 01 Delay(ioDelayRiseModel0) Slew(Slop
eRiseModel0))

Path(ADDR[0:3] *> QO[0:3]  01 10 Delay(ioDelayFallModel0) Slew(Slop
eFallModel0))

Path(ADDR[0:3] *> QO[0:3]  10 10 Delay(ioDelayFallModel0) Slew(Slop
eFallModel0))

    Setup(ADDR[0:3] *> CLK  Rise Negedge (Const(3.2)))
    Setup(ADDR[0:3] *> CLK  Fall Negedge (Const(3.2)))
)
```

Memory (Synch 2x2 RAM with Single Port)

This example is presented as “Example 5: Synopsys Description” and “Example 5: TLF Equivalent” on page 388.

Example 5: Synopsys Description

```
library (RAM) {
    type(bus2) {
        base_type : array;
        data_type : bit;
        bit_width : 2;
        bit_from : 0;
        bit_to : 1;
        downto : false;
```

```
    }
    cell(async_dual) {
        memory() { /* Indicates this is a memory cell */
            type : ram;
            address_width : 2;
            word_width : 2;
        }
    bus (RA) {
        bus_type  : "bus2";
        direction : input;
        capacitance : 0.241;
        timing () { /*Address setup time */
            timing_type : setup_falling;
            intrinsic_rise : 1.7;
            intrinsic_fall : 4.2;
            related_pin : "EN";
         }
    }
    bus (DI) {
         bus_type: "bus2"
         direction : input;
         capacitance : 0.144;
         memory_write() { /* Indicate this is a write port */
            address : RA;
            clocked_on : "EN'";
         }
         timing () { /* Input data setup time */
            timing_type : setup_rising;
            intrinsic_rise : 1.7;
            intrinsic_fall : 1.1;
            related_pin : "EN";
          }
    }
    pin (EN) {
        direction : input;
        capacitance : 0.28;
        clock : true; /* EN pulse is modeled as a clock */
    }
    bus(DO){
        bus_type  : "bus2";
        direction : output;
        capacitance : 0.097;
        memory_read() { /* Indicates that this is a read port */
            address : RA;
}
        timing () {
            timing_sense : non_unate; /* combinational delay */
```

```
            intrinsic_rise : 5.15;
            intrinsic_fall : 4.51;
            rise_resistance : 0.020;
            fall_resistance : 0.017;
            related_bus_pins : "RA";
        }
    }
 }
}
```

Example 5: TLF Equivalent

```
Header(
    Library("RAM")
    TLF_Version("4.1")
    Generated_By("UMAM")
)
Cell(async_dual
    Timing_Model(ioDelayRiseModel0
        (Spline
            (Load_Axis 0 1.000000)
            (Input_Slew_Axis 0 1.000000)
            data(
                (5.150000 5.150000)
                (5.170000 5.170000)
            )
        )
    )
    Timing_Model(ioDelayFallModel0
        (Spline
            (Load_Axis 0 1.000000)
            (Input_Slew_Axis 0 1.000000)
            data(
                (4.510000 4.510000)
                (4.527000 4.527000)
            )
        )
    )
    Timing_Model(SlopeRiseModel0
        (Spline
        (Load_Axis 0 1.000000)
            data(
                (0 0.020000)
            )
        )
    )
```

```
    Timing_Model(SlopeFallModel0
        (Spline
            (Load_Axis 0 1.000000)
            data(
                (0 0.017000)
            )
        )
    )
    Pin(EN Pintype(Input) Clock_pin Capacitance(0.280000))
    Bus(RA[0:1]
        Pintype(Input)
        Capacitance(0.241000)
    )
    Bus(DI[0:1]
        Pintype(Input)
        Capacitance(0.144000)
    )
    Bus(DO[0:1]
        Pintype(Output)
        Capacitance(0.097000)
    )
    Memory_Props(
        Memory_type(RAM)
        Memory_opr(Synchronous)
        Address_Width(2)
        Data_Width(2)
    )
    Memory_Bus(DI
            Busmode(write)
            Address_Bus(RA)
            Clock(~(EN))
        )
    Memory_Bus(DO
            Busmode(Read)
            Address_Bus(RA)
        )

    Path(RA[0:1] *> DO[0:1]  01 01 Delay(ioDelayRiseModel0)
        Slew(SlopeRiseModel0))
Path(RA[0:1] *> DO[0:1]  10 01 Delay(ioDelayRiseModel0)
        Slew(SlopeRiseModel0))
    Path(RA[0:1] *> DO[0:1]  01 10 Delay(ioDelayFallModel0)
        Slew(SlopeFallModel0))
    Path(RA[0:1] *> DO[0:1]  10 10 Delay(ioDelayFallModel0)
        Slew(SlopeFallModel0))
    Setup(RA[0:1] *> EN  Rise Negedge (Const(1.7)))
    Setup(RA[0:1] *> EN  Fall Negedge (Const(4.2)))
```

```
    Setup(DI[0:1] *> EN  Rise Posedge (Const(1.7)))
    Setup(DI[0:1] *> EN  Fall Posedge (Const(1.1)))
)
```

Units and Pad Modeling

This example is presented as “Example 6: Synopsys Description” and “Example 6: TLF Equivalent” on page 391.

Example 6: Synopsys Description

```
/* For Units, Pad Modelling,*/
library (example1) {
    time_unit : "1ns";
    voltage_unit : "1V";
    current_unit : "1mA";
    pulling_resistance_unit : "1kohm";
    capacitive_load_unit( 1,pf );
    input_voltage(CMOS) {
      vil : 1.5;
      vih : 3.5;
      vimin : -0.3;
      vimax : VDD + 0.3;
    }
    input_voltage(CMOS_SCHMITT) {
      vil : 1.0;
      vih : 4.0;
      vimin : -0.3;
      vimax : VDD + 0.3;
    }
    output_voltage(GENERAL) {
      vol : 0.4;
      voh : 2.4;
      vomin : -0.3;
      vomax : VDD + 0.3;
    }
    k_process_drive_current: 0.105;

/***** BIDIRECTIONAL PAD *****/
cell(BIBUF) {
    pad_cell : true;
    pin(E ) {
        direction : input;
        capacitance : 1.800000;
```

```
    }
    pin(Y ) {
        direction : output;
        function : "PAD";
        driver_type : "open_source pull_up";
        pulling_resistance : 10000;
    }
pin(PAD ) {
        direction : inout;
        is_pad : true;
        drive_current : 2.0;
        output_voltage : GENERAL;
        input_voltage : CMOS;
slew_control : high;
        rise_current_slope_before_threshold : 0.18;
        rise_time_before_threshold : 0.8;
        rise_current_slope_after_threshold : -0.09;
        rise_time_after_threshold : 2.4;
        fall_current_slope_before_threshold : -0.14;
        fall_time_before_threshold : 0.55;
        fall_current_slope_after_threshold : 0.07;
        fall_time_after_threshold : 1.8;
        three_state : "E";
    }
}
/*INPUT PAD WITH HYSTERESIS*/
cell(INBUFH) {
    pad_cell : true;
    pin(PAD ) {
        direction : input;
        is_pad : true;
        hysteresis : true;
        input_voltage : CMOS_SCHMITT;
    }
    pin(Y ) {
        direction : output;
        function : "PAD";
    }
 }
}
```

Example 6: TLF Equivalent

```
Header(
    Library("example1")
    TLF_Version("4.1")
```

```
    Generated_By("UMAM")
)

Process_Mult_Model(k_process_drive_current
    (Linear
        value(-~:~:0.895000:0.105000)
    )
)

PROPERTIES(
Input_voltage(CMOS
            Volt_Low_Threshold(1.5)
            Volt_High_Threshold(3.5)
            Volt_Min(-0.3)
            Volt_Max(0.3)
)

Input_Voltage(CMOS_SCHMITT
            Volt_Low_Threshold(1.0)
            Volt_High_Threshold(4.0)
            Volt_Min(-0.3)
Volt_Max(0.3)
)

Output_voltage(GENERAL
            Volt_Low_Threshold(0.4)
            Volt_High_Threshold(2.4)
            Volt_Min(-0.3)
            Volt_Max(0.3)
)

    Unit(
        Cap_Unit(1pf)
        Current_Unit(1mA)
        Res_Unit(1kohm)
        Time_Unit(1ns)
        Volt_Unit(1V)
    )
    Proc_Mult_Dcurrent(k_process_drive_current)
)
Cell(BIBUF
Pad_cell
    Pin(E Pintype(Input) Capacitance(0.000180))
    Pin(Y
        Pintype(Output)
        Function(PAD)
        Drivetype(open_source)
```

```
        Pull(Up)
        Pull_Resistance(10000)
    )
    Pin(PAD
        Pintype(Bidir)
    Pad_pin
        Enable(~(E))
        Pad_Props(
         Input_Voltage(CMOS)
         Output_Voltage(GENERAL)
         Csat(Rise(-0.09) Fall(0.07))
         Csbt(Rise(0.8) Fall(-0.14))
         Cttat(Rise(2.4) Fall(1.8))
         Cttbt(Rise(0.8) Fall(0.55))
         Dcurrent(2.0)
        )
    )
)

Cell(INBUFH
Pad_cell

    Pin(PAD
        Pintype(Input)
    Pad_pin
Pad_Props(
            Input_Voltage(CMOS_SCHMITT)
            Hyseteresis
        )
    )
    Pin(Y Pintype(Output) Function(PAD))
)
```

Routing

This example is presented as “Example 7: Synopsys Description” on page 393 and “Example 7: TLF Equivalent” on page 394.

Example 7: Synopsys Description

```
/* Routing information */
library(routing) {
    default_min_porosity : 15.0;
    routing_layers("metal2", "metal3");
```

```
    cell("ND2P") {
    routing_track(metal2) {
        tracks : 2;
        total_track_area : 0.2;
    }
    routing_track(metal3) {
        tracks : 4;
        total_track_area : 0.4;
    }
    pin (Y) {
        capacitance : 0.000000;
        direction : output;
        function : "((A1 & A0))'" ;
    }
    pin (A0) {
        capacitance : 17.770000;
        direction : input;
    }
    pin (A1) {
        capacitance : 18.600000;
        direction : input;
    }
    }
}
```

Example 7: TLF Equivalent

```
Header(
    Library("routing")
    TLF_Version("4.1")
    Generated_By("UMAM")
)

PROPERTIES(
    Routing_Layer(metal2 metal3)
    Min_Porosity(15.0)
)

Cell(ND2P

    Routing_Props(metal2
        Available_Track(2)
Track_Area(0.2)
    )

    Routing_Props(metal3
```

```
        Available_Track(4)
        Track_Area(0.4)
    )


Pin(Y Pintype(Output)) Function(~((A1&A0))) Capacitance(0.000000))
Pin(A0 Pintype(Input) Capacitance(17.770000))
Pin(A1 Pintype(Input) Capacitance(18.600000))
)
```

Mixed Threshold Setting

This example is presented as “Example 8: Synopsys Description” and “Example 8: TLF Equivalent” on page 397.

Example 8: Synopsys Description

```
/*
This example demonstrates setting cell level thresholds, for a libr
ary which
contains intermixing of both cell_rise/fall and rise/
fall_propagation attributes.
*/

library(mixedThreshold) {

  date : "Feb. 11, 1998";
  revision : 1.0;

  delay_model : table_lookup;

  lu_table_template(rise_x1){
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    index_1 ("0.1093, 0.6000");
    index_2 ("0.0, 3.85");
  }
  lu_table_template(fall_x1){
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    index_1 ("0.1153, 0.4085");
    index_2 ("0.0, 3.85");
  }
```

```
  k_process_cell_rise   : 1.0;
  k_process_cell_fall   : 1.0;

  k_process_rise_propagation   : 1.0;
  k_process_fall_propagation   : 1.0;

  nom_process : 1.0;
  nom_voltage : 5.0;
  nom_temperature : 25.0;

cell(BUF)  {
  area : 2;
  pin(O)  {
    direction : output;
    max_fanout : 61;
    max_capacitance : 61;
    function : "I1";
    timing()  {
      related_pin : "I1";
cell_rise(rise_x1) {
        values ("0.1663,0.2416", \
                "0.4669,0.5467");
      }
      rise_transition(rise_x1) {
        values ("0.0929,0.2364", \
                "0.1618,0.2985");
      }
      cell_fall(fall_x1) {
values ("0.1513,0.2249", \
                "0.2321,0.3176");
      }
      fall_transition(fall_x1) {
        values ("0.0632,0.1468", \
                "0.1285,0.2070");
      }
    }
  }
  pin(I1)  {
    direction : input;
    capacitance : 2;
    fanout_load : 2;
  }
}/* cell (BUF) */
cell(BUF2)  {
  area : 2;
  pin(O)  {
    direction : output;
```

```
    max_fanout : 107;
    max_capacitance : 107;
    function : "I1";
    timing()  {
      related_pin : "I1";
      rise_propagation(rise_x1) {
        values ("0.2876,0.3785", \
                "0.7210,0.8155");
      }
      rise_transition(rise_x1) {
        values ("0.1454,0.3065", \
                "0.2237,0.3654");
      }
      fall_propagation(fall_x1) {
        values ("0.2438,0.3365", \
                "0.3627,0.4633");
      }
      fall_transition(fall_x1) {
        values ("0.1117,0.2017", \
                "0.1840,0.2720");
      }
    }
  }
  pin(I1)  {
direction : input;
     capacitance : 1;
     fanout_load : 1;
  }
 }/* cell BUF2 */
}/* library */
```

Example 8: TLF Equivalent

```
/*
This example demonstrates setting cell level thresholds, for a libr
ary which
contains intermixing of both cell_rise/fall and rise/
fall_propagation attributes.
*/

Header(
    Library("mixedThreshold")
    Date("Feb. 11, 1998")
    Version("1")
    TLF_Version("4.1")
)
```

```
Timing_Model(rise_x1Mod
    (Spline
        (Input_Slew_Axis 0.109300 0.600000)
        (Load_Axis 0.000000 3.850000)
        data()
    )
)

Timing_Model(fall_x1Mod
    (Spline
        (Input_Slew_Axis 0.115300 0.408500)
        (Load_Axis 0.000000 3.850000)
        data()
    )
)

PROCESS_MULT_MODEL(k_process_cell_fallMod
    (Linear
        value(-~:~:0.000000:1.000000)
    )
)

PROCESS_MULT_MODEL(k_process_cell_riseMod
    (Linear
        value(-~:~:0.000000:1.000000)
    )
)

PROCESS_MULT_MODEL(k_process_fall_propagationMod
    (Linear
value(-~:~:0.000000:1.000000)
    )
)

PROCESS_MULT_MODEL(k_process_rise_propagationMod
    (Linear
        value(-~:~:0.000000:1.000000)
    )
)

PROPERTIES(
Proc_Var(1.000000)
    Voltage(5.000000)
    Temperature(25.000000)
)
```

```
Cell(BUF

    Timing_Model(ioDelayRiseModel0 rise_x1Mod
        (Spline
            data(
                (0.166300 0.241600)
                (0.466900 0.546700)
            )
        )
    )

    Timing_Model(ioDelayFallModel0 fall_x1Mod
        (Spline
            data(
                (0.151300 0.224900)
                (0.232100 0.317600)
            )
        )
    )

    Timing_Model(SlopeRiseModel0 rise_x1Mod
        (Spline
            data(
                (0.092900 0.236400)
                (0.161800 0.298500)
            )
        )
    )

    Timing_Model(SlopeFallModel0 fall_x1Mod
        (Spline
            data(
                (0.063200 0.146800)
                (0.128500 0.207000)
            )
        )
)

        Area(2.000000)

Proc_Mult_Propagation(Rise(k_process_cell_riseMod) Fall(k_process_c
ell_fallMod))
        Table_Input_Threshold(0.500000)
        Table_Output_Threshold(0.500000)
        Table_Transition_Start(0.100000)
        Table_Transition_End(0.500000)
```

```
Pin(O Pintype(Output) Function(I1) Load_Limit(1.586000))
    Pin(I1 Pintype(Input) Capacitance(0.052000))


Path(I1 => O  01 01 Delay(ioDelayRiseModel0) Slew(SlopeRiseModel0))

Path(I1 => O  10 10 Delay(ioDelayFallModel0) Slew(SlopeFallModel0))

)

Cell(BUF2

    Timing_Model(ioDelayRiseModel0 rise_x1Mod
        (Spline
            data(
                (0.287600 0.378500)
                (0.721000 0.815500)
            )
        )
    )

    Timing_Model(ioDelayFallModel0 fall_x1Mod
        (Spline
            data(
                (0.243800 0.336500)
                (0.362700 0.463300)
            )
        )
    )

    Timing_Model(SlopeRiseModel0 rise_x1Mod
        (Spline
            data(
                (0.145400 0.306500)
                (0.223700 0.365400)
            )
        )
    )

    Timing_Model(SlopeFallModel0 fall_x1Mod
        (Spline
            data(
                (0.111700 0.201700)
(0.184000 0.272000)
            )
        )
    )
```

```
        Area(2.000000)
        Proc_Mult_Propagation(Rise(k_process_rise_propagationMod)
                Fall(k_process_fall_propagationMod))
        Table_Input_Threshold(0.500000)
        Table_Output_Threshold(0.100000)
Table_Transition_Start(0.100000)
        Table_Transition_End(0.500000)


Pin(O Pintype(Output) Function(I1) Load_Limit(2.782000))
Pin(I1 Pintype(Input) Capacitance(0.026000))

    Path(I1 => O  01 01 Delay(ioDelayRiseModel0)
            Slew(SlopeRiseModel0))
    Path(I1 => O  10 10 Delay(ioDelayFallModel0)
            Slew(SlopeFallModel0))
)
```

Slew Degradation

This example is presented as “Example 9: Synopsys Description” and “Example 9: TLF Equivalent” on page 402.

Example 9: Synopsys Description

```
library (Slew_degradation) {
    delay_model : table_lookup;
    lu_table_template(trans_deg) {
            variable_1 : output_pin_transition
            index_1("0, 1");
            variable_2 : connect_delay
            index_2("0, 1");
    }

    /* Slew Degradation Tables */
    rise_transition_degradation(trans_deg) {
        values("0.0, 0.6", \
            "1.0, 1.6");
    }
    fall_transition_degradation(trans_deg) {
        values("0.0, 0.8", \
            "1.0, 1.8");
```

```
    }
}
```

Example 9: TLF Equivalent

```
Header(
    Library("Slew_degradation")
    TLF_Version("4.1")
    Generated_By("UMAM")
)

Slew_Degrade_Model(trans_deg
    (Spline
        (Output_Slew_Axis 0.000000 1.000000)
        (Wire_delay 0.000000 1.000000)
    )
)

Slew_Degrade_Model(rise_transition_degradation trans_deg
    (Spline
        data(
            (0.0 0.6)
            (1.0 1.6)
        )
     )
)

Slew_Degrade_Model(fall_transition_degradation trans_deg
    (Spline
data(
            (0.0 0.8)
            (1.0 1,8)
         )
     )
)

PROPERTIES(
    Slew_Degradation(Rise(rise_transition_degradation)
             Fall(fall_transition_degradation))
)
```

Internal Pin

This example is presented as “Example 10: Synopsys Description” and “Example 10: TLF Equivalent” on page 404.

Example 10: Synopsys Description

```
cell (REG_AND) {
ff(IQ, IQN) {
             clocked_on : "CLK" ;
             next_state : "B" ;
         }
         pin (Y) {
             direction : output;
             function : "IQ * A";
             timing () {
               timing_sense : postive_unate;
               related_pin : "A";
                  ........
             }
             timing () {
               related_pin : "CLK";
             }
         }
         pin (A) {
             direction : input ;
         }
         pin (B) {
             direction : input;
             timing () {
               timing_type : setup_rising ;
               related_pin : "CLK";
             }
             timing () {
               timing_type : hold_rising ;
               related_pin : "CLK";
             }
}
         pin (CLK) {
             direction : input ;
             clock : true ;
         }
     }
```

Example 10: TLF Equivalent

```
Cell(REG_AND
    Pin(Y Pintype(Output) Function(IQ&&A))
    Pin(A Pintype(Input))
Pin(B PinType(Input))
    Pin(CLK Clock_pin PinType(Input))
    Pin(IQ PinType(Internal))

    Register(
        Output(IQ)
        Input(B)
        Clock(CLK)
    )

    Path(CLK => Y  01 01 Delay(...) Slew(...))
    Path(CLK => Y  01 10 Delay(...) Slew(...))
    Path(A => Y  01 01 Delay(...) Slew(...))
    Path(A => Y  10 10 Delay(...) Slew(...))

    Setup(B => CLK 01 Posedge (...))
    Setup(B => CLK 10 Posedge (...))
    Hold(B => CLK 01 Posedge (...))
    Hold(B => CLK 10 Posedge (...))
)
```

Wavetable

This example is presented as “Example 11: TLF Description”. There is no Synopsys description because Synopsys has no corresponding constructs.

Example 11: TLF Description

```
Header(
    Library("wavetable")
    TLF_Version("4.1")
    Generated_By("UMAM")
)

Current_Model(output_by_cap1_cap2_and_trans
    (Wavetable
        (Load_Axis 0.000000 5.000000)
        (Load2_Axis 0.000000 5.000000)
        (Input_Slew_Axis 0.000000 1.000000)
    )
```

```
)

Current_Model(output_by_cap_and_trans
    (Wavetable
        (Load_Axis 0.000000 5.000000)
        (Input_Slew_Axis 0.000000 1.000000)
    )
)


PROPERTIES(
    Unit(
    Current_Unit(1mA)
    )
    Volt_Mult_SUPPC(1.0)
    Volt_Mult_GNDC(2.0)
)

Cell(AN2

    Current_Model(2DTable1 output_by_cap_and_trans
        (Wavetable
            data(
        ((-6.25 0)(-5.61 0)(3.99 0.000167772)
            (6.25 0.000239228)(22.9653 9.70859e-05)
            (40.2198 1.34283e-05)(62.8778 0))
        ((-5.61 0)(3.99 0.000167772)(6.25 0.00024718)
            (40.2393 0.000111914)(74.7681 1.69049e-05)
            (98.75 4.97199e-06)(128.75 0))
        ((-1.25 0)(3.99 0.000167772) (6.25 0.000239228)
            (22.9653 9.70859e-05)(40.2198 1.34283e-05)
            (62.8778 0)(128.75 0))
        ((-0.61 0)(3.99 0.000167772)(6.25 0.00024718)
            (40.2393 0.000111914)(74.7681 1.69049e-05)
            (98.75 4.97199e-06)(128.75 0))
            )
        )
    )
    Current_Model(2DTable2 output_by_cap_and_trans
        (Wavetable
            data(
        ((-8.25 0)(-5.61 0)(3.99 0.000167772)
            (6.25 0.000239228)(22.9653 9.70859e-05)
            (40.2198 1.34283e-05)(62.8778 0))
```

```
        ((-3.61 0)(3.99 0.000167772)(6.25 0.00024718)
            (40.2393 0.000111914)(74.7681 1.69049e-05)
            (98.75 4.97199e-06)(128.75 0))
        ((-0.25 0)(3.99 0.000167772) (6.25 0.000239228)
            (22.9653 9.70859e-05)(40.2198 1.34283e-05)
            (62.8778 0)(128.75 0))
        ((-0.1 0)(3.99 0.000167772)(6.25 0.00024718)
            (40.2393 0.000111914)(74.7681 1.69049e-05)
            (98.75 4.97199e-06)(128.75 0))
        )
    )
)
    Pin(Z Pintype(Output))
    Pin(A Pintype(Input))
    Pin(B Pintype(Input))


Path((A B) *> Z 01 01 Delay(...) Slew(...) SUPPLY_CURRENT(2DTable1)
 GROUND_CURRENT(2DTable2))

Path((A B) *> Z 10 10 Delay(...) Slew(...) SUPPLY_CURRENT(2DTable1)
 GROUND_CURRENT(2DTable2))

Path((A B) *> Z 10 10 COND(...) SDF_COND(...) Delay(...) Slew(...)
SUPPLY_CURRENT(2DTable1))
)

Cell(FLIPFLOP2

    Current_Model(3DTable output_by_cap1_cap2_and_trans
        (Wavetable
            data(
        ((-6.25 0)(-
5.61 0)(3.99 0.000167772) (6.25 0.000239228)(22.9653 9.70859e-
05) (40.2198 1.34283e-05)(62.8778 0))
        ((-
5.61 0)(3.99 0.000167772)(6.25 0.00024718) (40.2393 0.000111914)(74
.7681 1.69049e-05) (98.75 4.97199e-06)(128.75 0))
        ((-
1.25 0)(3.99 0.000167772) (6.25 0.000239228)(22.9653 9.70859e-
05) (40.2198 1.34283e-05)(62.8778 0)(128.75 0))
        ((-
0.61 0)(3.99 0.000167772)(6.25 0.00024718) (40.2393 0.000111914)(74
.7681 1.69049e-05) (98.75 4.97199e-06)(128.75 0))
        ((-8.25 0)(-
5.61 0)(3.99 0.000167772) (6.25 0.000239228)(22.9653 9.70859e-
05) (40.2198 1.34283e-05)(62.8778 0))
```

```
        ((-
3.61 0)(3.99 0.000167772)(6.25 0.00024718) (40.2393 0.000111914)(74
.7681 1.69049e-05) (98.75 4.97199e-06)(128.75 0))
        ((-
0.25 0)(3.99 0.000167772) (6.25 0.000239228)(22.9653 9.70859e-
05) (40.2198 1.34283e-05)(62.8778 0)(128.75 0))
        ((-
0.1 0)(3.99 0.000167772)(6.25 0.00024718) (40.2393 0.000111914)(74.
7681 1.69049e-05) (98.75 4.97199e-06)(128.75 0))

)
        )
     )


    Pin(CP Pintype(Input))
    Pin(D Pintype(Input))
    Pin(Q Pintype(Output))
    Pin(QN Pintype(Output))

    Path(CP => Q 01 01 Other_Pin(QN) SUPPLY_CURRENT(3DTable))
    Pin(CP => Q 01 10 Other_Pin(QN) GROUND_CURRENT(3DTable))
)
```

Wireload and Synthesis Constructs

This example is presented as “Example 12: Synopsys Description” and “Example 12: TLF Equivalent” on page 408.

Example 12: Synopsys Description

```
library(syn_related) {
  date : "Feb. 2, 1999";

  delay_model : table_lookup;

    wire_load_selection (“metal_2”) {
        wire_load_from_area(0,42770,"STD1");
        wire_load_from_area(42770,85540,"STD2");
    }
    wire_load_selection (“metal_3”) {
        wire_load_from_area(85540,128310,"STD3");
        wire_load_from_area(128310,213850,"STD5");
        wire_load_from_area(213850,342160,"STD8");
```

```
    }
    default_wire_load_selection : “metal_3”;
    default_wire_load_mode : top;
    operating_conditions("T125_V4_P1") {
        process : 1;
        voltage : 4 ;
        temperature : 125 ;
        tree_type : balanced_tree ;
  }
cell(buffer) {
    dont_use : true;
    dont_touch : true;
    ...
}
}
```

Example 12: TLF Equivalent

```
Header(
    Library("syn_related")
    TLF_Version("4.1")
    Generated_By("UMAM")
)

    PROPERTIES(
        WireLoad_By_Area(metal_2
            (STD1 42770.000000 Net_Cap(...) Net_Res(...))
            (STD2 85540.000000 Net_Cap(...) Net_Res(...))
        )

        WireLoad_By_Area(metal_3
            (STD3 128310.000000 Net_Cap(...) Net_Res(...))
            (STD5 213850.000000 Net_Cap(...) Net_Res(...))
            (STD8 342160.000000 Net_Cap(...) Net_Res(...))
         )
        Default_Wireload_Group(metal_3)
        Default_Wireload_Mode(top)

        PVT_Conds(T125_V4_P1
            Proc_Var(1)
            Voltage(4)
            Temperature(125)
            Tree_Type(balanced_tree)
        )
    )
```

```
Cell(BUFFER

    DONT_USE
    DONT_TOUCH

    ...
)
```

Scan Cell Modeling

Scan cell modelling is shown in two examples:

- TEST_REGISTER
- TEST_LATCH

A TEST_REGISTER example is presented as “Example 13: Synopsys Description” and “Example 13: TLF Equivalent” on page 410.
A TEST_LATCH example is presented as “Example 14: Synopsys Description” on page 411 and “Example 14: TLF Equivalent” on page 412.

Example 13: Synopsys Description

```
library(scan_related) {
  date : "Feb. 2, 1999";
  delay_model : table_lookup;
  cell ( scan_cell ) {
    ff ("IQ", "IQN") {
        next_state : "((D & TE') + (TI & TE))";   // scan behavior
        clocked_on : "CP";
    }
    pin ( Q ) {
        direction : output;
        function : "IQ";
    }
    pin ( D ) { direction : input; }
    pin ( CP ) { direction : input; }
    pin ( TI ) { direction : input; }
    pin ( TE ) { direction : input; }
    test_cell () {
        ff ("IQ", "IQN") {
            next_state : "D";   // normal behavior
            clocked_on : "CP";
        }
        pin (D) { direction : input; }
```

```
        pin (CP) { direction : input; }
        pin (TI) {
            direction : input;
            signal_type : test_scan_in;   // scan input pin
        }
        pin (TE) {
            direction : input;
            signal_type : test_scan_enable;   // scan enable pin
        }
        pin (Q) {
            direction : output;
            function : "IQ";
            signal_type : test_scan_out;  // scan output pin
        }
    }
  }
}
```

Example 13: TLF Equivalent

```
Header(
    Library("scan_related")
    TLF_Version("4.1")
    Generated_By("Maruti")
)
Cell(scan_cell
    Register(
        Output(Q)
        Input((D & !TE) | (TI & TE))  // scan behavior
        Clock(CP)
    )
    Pin(Q Pintype(Output) SCAN_PINTYPE(Output))
    Pin(D Pintype(Input))
    Pin(CP Pintype(Input) Clock_Pin)
    Pin(TI Pintype(Input) SCAN_PINTYPE(Input))
    Pin(TE Pintype(Input) SCAN_PINTYPE(Enable))
    TEST_REGISTER(   // normal behavior
        Output(Q)
        Input(D)
        Clock(CP)
    )
)
```

Example 14: Synopsys Description

```
library(scan_related) {

    date : "Feb. 2, 1999";
        delay_model : table_lookup;

        cell ( scan_cell ) {
        latch ("IQ", "IQN") {
        data_in : "((D & TE') + (TI & TE))";    // scan behavior
        enable : "Clk";
    }
    pin ( Q ) {
        direction : output;
        function : "IQ";
    }
    pin ( D ) {
        direction : input;
    }
    pin ( Clk ) {
        direction : input;
        clock : true;

    }
    pin ( TI ) {
        direction : input;
    }
    pin ( TE ) {
        direction : input;
    }
    test_cell () {
        latch ("IQ", "IQN") {
            data_in : "D";    // normal behavior
            enable : "Clk";
        }
        pin (D) {
            direction : input;
        }
        pin (Clk) {
            direction : input;
            clock : true;
        }
        pin (TI) {
            direction : input;
            signal_type : test_scan_in;    // scan input pin
        }
        pin (TE) {
```

```
            direction : input;
            signal_type : test_scan_enable;    // scan enable pin
        }
        pin (Q) {
            direction : output;
            function : "IQ";
            signal_type : test_scan_out;    // scan output pin
        }
    }
test_cell () {
        latch ("IQ", "IQN") {
            data_in : "D'";    // normal behavior
            enable : "Clk";
        }
        pin (D) {
            direction : input;
        }
        pin (Clk) {
            direction : input;
            clock : true;
        }
        pin (TI) {
            direction : input;
            signal_type : test_scan_in;    // scan input pin
        }
        pin (TE) {
            direction : input;
            signal_type : test_scan_enable;    // scan enable pin
        }
        pin (Q) {
            direction : output;
            function : "IQ";
            signal_type : test_scan_out;    // scan output pin
        }
    }

}
}
```

Example 14: TLF Equivalent

```
Header(
    Library("scan_related")
    TLF_Version("4.1")
    Generated_By("Maruti")
)
```

```
Cell(scan_cell
    LATCH(
        Output(Q)
        Input((D & !TE) | (TI & TE))  // scan behavior
        Clock(Clk)
    )
    Pin(Q Pintype(Output) SCAN_PINTYPE(Output))
    Pin(D Pintype(Input))
    Pin(Clk Pintype(Input) Clock_Pin)
    Pin(TI Pintype(Input) SCAN_PINTYPE(Input))
    Pin(TE Pintype(Input) SCAN_PINTYPE(Enable))
    TEST_LATCH(   // normal behavior
        Output(Q)
        Input(D)
        Clock(Clk)
)
        TEST_LATCH(   // normal behavior
        Output(Q)
        Input(~D)
        Clock(Clk)
    )
)
```

FLUENCE and FLUENCE_LIMIT

“Example 15: TLF Description” shows the use of

- `FLUENCE` at the cell and path levels
- `FLUENCE_LIMIT` at the cell and pin levels

There is no Synopsys description because Synopsys has no corresponding construct.

Example 15: TLF Description

```
Header(
    Library("fluence_lib")
    TLF_Version("4.1")
    Generated_By("Maruti")
)

Cell(myCell
    Fluence_Model(fluenceMod1
        (Spline
```

```
            (Input_Slew_Axis 0.1 0.3)
            (Load_Axis 0.01 0.03)
            data(
                // fluence for slew 0.1ns
                (0.00024718 0.000167772)
                // fluence for slew 0.3ns
                (0.000111914 0.000016904)
            )
        )
    )

    Fluence_Model(defFluenceMod
        (Const (0.001) )
    )

    Fluence_Model(fluenceMod2
        (Const (0.01:0.02:0.03) )
    )

    FLUENCE(defFluenceMod) // Cell level default fluence
    FLUENCE_LIMIT(0.1:0.2:0.3) // FLUENCE_LIMIT at cell level
    Pin(Y Pintype(Output) Function(~((A0 & A1) & A2))
        Capacitance(1.000000)
        // FLUENCE_LIMIT at pin level
        FLUENCE_LIMIT(WARN(0.205) ERROR(0.315))
    )

    Pin(A0 Pintype(Input) Capacitance(17.770000))
    Pin(A1 Pintype(Input) Capacitance(18.600000))
    Pin(A2 Pintype(Input) Capacitance(16.600000))

    PATH(A0 => Y 01 10 DELAY(..) SLEW(..) FLUENCE(fluenceMod1))
    PATH(A0 => Y 10 01 DELAY(..) SLEW(..) FLUENCE(fluenceMod1))
    PATH(A1 => Y 01 10 DELAY(..) SLEW(..) FLUENCE(fluenceMod1))
    PATH(A1 => Y 10 01 DELAY(..) SLEW(..) FLUENCE(fluenceMod1))

    // Here, by default, cell level default fluence, defFluenceMod
    // will be used for fluence
    PATH(A2 => Y 01 10 DELAY(..) SLEW(..))

    PATH(A2 => Y 10 01 DELAY(..) SLEW(..) FLUENCE(fluenceMod2))
)
```

PROPAGATION_DELAY_TABLE

This example is presented as “Example 16: Synopsys Description” and “Example 16: TLF Equivalent” on page 415.

Example 16: Synopsys Description

```
library (example1) {
    delay_model : table_lookup ;
    cell (XOR2) {
        pin (Y) {
            direction : output;
            function : "A0" ;
            timing () {
                related_pin : "A0" ;
                rise_propagation(inrise_template) {
                    ...
                    }
                rise_transition(inrise_template) {
                    ...
                    }
                fall_propagation(infall_template) {
                    ...
                    }
                fall_transition(infall_template) {
                    ...
                    }
                }
            }
        pin (A0) {
            direction : input;
            }
        }
    }
```

Example 16: TLF Equivalent

```
HEADER(
    LIBRARY("example1")
    VENDOR("Cadence")
    TLF_VERSION("4.3")
    )
// model section
TIMING_Model(inrise_templateMod
            ...
```

```
            )
TIMING_Model(infall_templateMod
            ...
            )
// properties section
PROPERTIES(
        PROPAGATION_DELAY_TABLE
        ...
        )
...
CELL(XOR2
    ...
    )
```

TEST_FUNCTION

This example is presented as “Example 17: Synopsys Description” and “Example 17: TLF Equivalent” on page 418. This example shows the use of:

- `TEST_FUNCTION` statement
- User defined attributes: `STATE_VARIABLE`, `state_variable_map`, `STATE_VARIABLE_INVERTED`

Example 17: Synopsys Description

```
library (scanlib) {
    cell(scan){
        ff ("IQ", "IQN") {
        next_state : "((D & TE’) + (TI & TE))";
        clocked_on : "CP";
        }
        test_cell () {
        ff ("IQ", "IQN") {
            next_state : "(D)";
            clocked_on : "CP";
            }
        pin (D) {
            direction : input;
            }
        pin (CP) {
            direction : input;
            }
        pin (TI) {
            direction : input;
```

```
        signal_type : test_scan_in;
        }
    pin (TE) {
        direction : input;
        signal_type : test_scan_enable;
        }
    pin (Q) {
        direction : output;
        function : "IQ";
        }
    pin (QN) {
        direction : output;
        function : "IQN";
        }
    pin (SQ) {
        direction : output;
        function : "IQ";
        signal_type : test_scan_out;
        }
    pin (SQN) {
        direction : output;
        signal_type : test_scan_out;
        }
    }
    pin ( Q ) {
        direction : output;
        capacitance : 0.400000;
        }
    pin ( QN ) {
        direction : output;
        capacitance : 0.400000;
        function : "IQN";
        }
    pin ( SQ ) {
        direction : output;
        capacitance : 0.400000;
        }
    pin ( SQN ) {
        direction : output;
        capacitance : 0.400000;
        function : "IQN";
        }
    pin ( D ) {
        direction : input;
        capacitance : 0.027000;
        }
    pin ( CP ) {
```

```
            direction : input;
            capacitance : 0.028000;
            clock : true ;
            }
        pin ( TI ) {
            direction : input;
            capacitance : 0.030000;
            }
        pin ( TE ) {
            direction : input;
            capacitance : 0.059000;
            }
        }
    }
```

Example 17: TLF Equivalent

```
HEADER(
    LIBRARY("scanlib")
    VENDOR("Cadence")
    Environment("Nominal")
    TLF_VERSION("4.3")
    )
// User properties section
DEFINE_ATTRIBUTE(state_variable_map (PIN) (STRING))
DEFINE_ATTRIBUTE(STATE_VARIABLE (PIN) (BOOLEAN))
DEFINE_ATTRIBUTE(STATE_VARIABLE_INVERTED (PIN) (BOOLEAN))
// properties section
PROPERTIES(
    Proc_Mult(1.000000)
    Volt_Mult(1.000000)
    Temp_Mult(1.000000)
    )
CELL(scan
    // model section
    PIN(Q
    PINTYPE(OUTPUT )
    state_variable_map("IQ")
    TEST_FUNCTION( IQ)
    // properties section
    Capacitance(0.400000)
    )
    PIN(QN
    PINTYPE(OUTPUT )
    state_variable_map("IQN")
    FUNCTION( IQN)
```

```
TEST_FUNCTION( IQN)
// properties section
Capacitance(0.400000)
)
PIN(SQ
    PINTYPE(OUTPUT )
    state_variable_map("IQ")
    TEST_FUNCTION( IQ)
    SCAN_PINTYPE(Output )
    // properties section
    Capacitance(0.400000)
    )
PIN(SQN
    PINTYPE(OUTPUT )
    state_variable_map("IQN")
    FUNCTION( IQN)
    SCAN_PINTYPE(Output )
    // properties section
    Capacitance(0.400000)
    )
PIN(D
    PINTYPE(INPUT )
    // properties section
    Capacitance(0.027000)
    )
PIN(CP
    PINTYPE(INPUT )
    CLOCK_PIN
    // properties section
    Capacitance(0.028000)
    )
PIN(TI
    PINTYPE(INPUT )
    SCAN_PINTYPE(Input )
    // properties section
    Capacitance(0.030000)
    )
PIN(TE
    PINTYPE(INPUT )
    SCAN_PINTYPE(Enable )
    // properties section
    Capacitance(0.059000)
    )
PIN(IQ
    PINTYPE(INTERNAL )
    STATE_VARIABLE(TRUE)
    )
```

```
    PIN(IQN
        PINTYPE(INTERNAL )
        STATE_VARIABLE_INVERTED(TRUE)
        )
    REGISTER(
        CLOCK(CP)
        INPUT(((D & ~TE) | (TI & TE)))
        OUTPUT(IQ)
        INVERTED_OUTPUT(IQN)
        )
    TEST_REGISTER(
        CLOCK(CP)
        INPUT((D))
        OUTPUT(IQ)
        INVERTED_OUTPUT(IQN)
        )
    )
```

STATE_FUNCTION

This example is presented as “Example 18: Synopsys Description” and “Example 18: TLF Equivalent” on page 422. This example shows the use of:

- `STATE_FUNCTION` statement
- User defined attribute: `internal_node`

Example 18: Synopsys Description

```
library (simple) {
    technology (cmos);
    delay_model : lsi_cmde;
    time_unit : "1ns";
    voltage_unit : "1V";
    current_unit : "1mA";
    pulling_resistance_unit : "1kohm";
    capacitive_load_unit (1, pf);
    nom_process : 1.000000;
    nom_temperature : 25.000000;
    nom_voltage : 5.000000;
cell ( LD1S2A ) {
    area : 28.000 ;
    scan_group : "A";
    statetable("SCK2 SI  CP  D SCK1 ", " Q1 Q2") {
    table : "  L - - - - :  - - : N - , /* master inactive */\
```

```
H  L/H  - - - : - - : L/H - , /* master loads SI */\
- - H L/H L : - - : - L/H, /* slave loads D */\
- - H - H : - - : - X , /* illegal clocking */\
- - L - L :  - - : - N ,  /* slave inactive */\
- - L - H : L/H - : - L/H " /* slave loads master */;
}
pin (Q1) {
    direction : internal ;
    internal_node : "Q1";
    }
pin ( Q ) {
    direction : output;
    capacitance : 0.000000;
    internal_node : "Q2";
    max_capacitance :  1.51386 ;
    min_capacitance :  0.03000 ;
    }
pin ( Q2 ) {
    direction : output;
    capacitance : 0.000000;
    internal_node : "Q2";
    max_capacitance :  1.52158 ;
    min_capacitance :  0.03000 ;
    }
pin(SQ) {
    direction : output ;
    state_function : "Q1";
}
pin ( D ) {
    direction : input;
    capacitance : 0.028000;
    }
pin ( CP ) {
    direction : input;
    capacitance : 0.020000;
    clock : true ;
    min_pulse_width_high : 0.5213 ;
    }
pin ( SCK1 ) {
    direction : input;
    capacitance : 0.015000;
    clock : true ;
    min_pulse_width_high : 0.3287 ;
    }
pin ( SCK2 ) {
    direction : input;
    capacitance : 0.034000;
```

```
        clock : true ;
        min_pulse_width_high : 0.5805 ;
        }
    pin ( SI ) {
        direction : input;
        capacitance : 0.022000;
        }
    }
}
```

Example 18: TLF Equivalent

```
HEADER(
    LIBRARY("simple")
    VENDOR("Cadence")
    Environment("Nominal")
    TECHNOLOGY("cmos")
    TLF_VERSION("4.3")
    GENERATED_BY("Syn2tlf4.1-s001")
    )
// User properties section
DEFINE_ATTRIBUTE(internal_node (PIN) (STRING))
// properties section
PROPERTIES(
    Proc_Var(1.000000)
    Voltage(5.000000)
    Temperature(25.000000)
    Proc_Mult(1.000000)
    Volt_Mult(1.000000)
    Temp_Mult(1.000000)
    // WireLoad  Models
    )
CELL(LD1S2A
    // model section
    // state table definitions
    STATE_TABLE(StTable
        (SCK2 SI CP D SCK1 : Q1 Q2)
        ((0 - - - - : - -: N -)
        (1 01 - - - : - -: 01 -)
        (- - 1 01 0 : - -: - 01)
        (- - 1 - 1 : - -: - X)
        (- - 0 - 0 : - -: - N)
        (- - 0 - 1 : 01 -: - 01)
        )
    )
// properties section
```

```
Area(28.000000)
PIN(Q1
    PINTYPE(INTERNAL )
    internal_node("Q1")
    MAP_TO_STPIN( StTable ( SCK2 SI CP D SCK1 : Q1 Q2))
    )
PIN(Q
    PINTYPE(OUTPUT )
    internal_node("Q2")
    // properties section
    Capacitance(0.000000)
    Load_Limit(1.513860)
    LOAD_MIN(0.030000)
    MAP_TO_STPIN( StTable ( SCK2 SI CP D SCK1 : Q1 Q))
    )
PIN(Q2
    PINTYPE(OUTPUT )
    internal_node("Q2")
    // properties section
    Capacitance(0.000000)
    Load_Limit(1.521580)
    LOAD_MIN(0.030000)
    MAP_TO_STPIN( StTable ( SCK2 SI CP D SCK1 : Q1 Q2))
    )
PIN(SQ
    PINTYPE(OUTPUT )
    STATE_FUNCTION( Q1)
    )
PIN(D
    PINTYPE(INPUT )
    // properties section
    Capacitance(0.028000)
    )
PIN(CP
    PINTYPE(INPUT )
    CLOCK_PIN
    // properties section
    Capacitance(0.020000)
    )
PIN(SCK1
    PINTYPE(INPUT )
    CLOCK_PIN
    // properties section
    Capacitance(0.015000)
    )
PIN(SCK2
    PINTYPE(INPUT )
```

```
    CLOCK_PIN
    // properties section
    Capacitance(0.034000)
    )
PIN(SI
    PINTYPE(INPUT )
    // properties section
    Capacitance(0.022000)
    )
// pinrels
MPWH( CP (Const(0.521300)) )
MPWH( SCK1 (Const(0.328700)) )
MPWH( SCK2 (Const(0.580500)) )
)
```

WIRE_DELAY

This example is presented as “Example 19: Synopsys Description” and “Example 19: TLF Equivalent” on page 426. This example shows the use of:

- `WIRE_DELAY` statement
- User defined attribute: `in_place_swap_mode`, `scale_slew_times`, `cell_footprint`, `dont_fault`, `auxiliary_pad_cell`, `pad_type`, `complementary_pin`, `x_function`, `fault_model`, `test_output_only`

Example 19: Synopsys Description

```
library(simple) {
    delay_model : table_lookup;
    in_place_swap_mode : match_footprint;
    scale_slew_times : false;
    capacitive_load_unit(1,pf);
    nom_process     :  1.00;
    nom_temperature :  25.0;
    nom_voltage     :  3.3;
    lu_table_template(net_delay_trans_deg) {
        variable_1 : output_transition;
        variable_2 : rc_product;
        index_1 ("0.02, 0.1, 0.3, 0.6, 1.0, 2.0, 3.0");
        index_2 ("0.0, 0.5, 1.0, 3.0, 6.0, 7.0");
        }
    rise_net_delay(net_delay_trans_deg) {
    index_1 ("0.02, 0.1, 0.3, 0.6, 1.0, 2.0, 3.0");
    index_2 ("0.00, 0.5, 1.0, 3.0, 6.0, 7.0");
```

```
  values("0.000000,0.355769, 0.705882, 2.105960, 4.205980,
   4.905983" \
  "0.000000, 0.375000, 0.727273, 2.129032, 4.229508, 4.929577" \
  "0.000000, 0.406250, 0.769231, 2.181818, 4.285714, 4.986301" \
  "0.000000, 0.431818, 0.812500, 2.250000, 4.363636, 5.065790" \
  "0.000000, 0.450000, 0.850000, 2.325000, 4.457143, 5.162500" \
  "0.000000, 0.470000, 0.900000, 2.460000, 4.650000, 5.366667" \
  "0.000000, 0.478571, 0.925000, 2.550000, 4.800000, 5.530000" ;
  }
  rise_transition_degradation(net_delay_trans_deg) {
 values("0.020000, 1.077692, 2.176863, 6.576292, 13.176147,
  15.376125"  \
  "0.100000, 1.016667, 2.100000, 6.487097, 13.083607, 15.283098" \
  "0.300000, 0.987500, 1.992308, 6.300000, 12.871428, 15.067123" \
  "0.600000, 1.100000, 1.975000, 6.100000, 12.600000, 14.784210" \
  "1.000000, 1.366667, 2.100000, 5.950000, 12.314285, 14.475000" \
  "2.000000, 2.220000, 2.733333, 5.960000, 11.900000, 13.977777" \
  "3.000000, 3.157143, 3.550000, 6.300000, 11.800000, 13.780000");
  }
  fall_net_delay(net_delay_trans_deg) {
  index_1 ("0.02, 0.1, 0.3, 0.6, 1.0, 2.0, 3.0");
  index_2 ("0.0, 0.5, 1.0, 3.0, 6.0, 7.0");
  values("0.000000, 0.355769, 0.705882, 2.105960, 4.205980,
   4.905983"  \
  "0.000000, 0.375000, 0.727273, 2.129032, 4.229508, 4.929577" \
  "0.000000, 0.406250, 0.769231, 2.181818, 4.285714, 4.986301" \
  "0.000000, 0.431818, 0.812500, 2.250000, 4.363636, 5.065790" \
  "0.000000, 0.450000, 0.850000, 2.325000, 4.457143, 5.162500" \
  "0.000000, 0.470000, 0.900000, 2.460000, 4.650000, 5.366667" \
  "0.000000, 0.478571, 0.925000, 2.550000, 4.800000, 5.530000");
  }
  fall_transition_degradation(net_delay_trans_deg) {
  values("0.020000, 1.077692, 2.176863, 6.576292, 13.176147,
   15.376125" \
  "0.100000, 1.016667, 2.100000, 6.487097, 13.083607, 15.283098" \
  "0.300000, 0.987500, 1.992308, 6.300000, 12.871428, 15.067123" \
  "0.600000, 1.100000, 1.975000, 6.100000, 12.600000, 14.784210" \
  "1.000000, 1.366667, 2.100000, 5.950000, 12.314285, 14.475000" \
  "2.000000, 2.220000, 2.733333, 5.960000, 11.900000, 13.977777" \
  "3.000000, 3.157143, 3.550000, 6.300000, 11.800000, 13.780000");
  }
  cell(abc){
  cell_footprint: "5mil";
  auxiliary_pad_cell: false;
  pad_type: clock ;
  dont_fault: sa0 ;
  pin(a){
```

```
        direction: output;
        dont_fault: sa01 ;
        complementary_pin: b ;
        test_output_only: false ;
        x_function: b ;
        fault_model: none ;
        }
    pin(b)
        {
        direction: input;
        }
    pin(o)
        {
        direction: output;
        }
    }
}
```

Example 19: TLF Equivalent

```
HEADER(
    LIBRARY("simple")
    VENDOR("Cadence")
    Environment("Nominal")
    TLF_VERSION("4.3")
    GENERATED_BY("Syn2tlf4.1-s001")
    )
// User properties section
DEFINE_ATTRIBUTE(in_place_swap_mode (LIBRARY) (STRING))
DEFINE_ATTRIBUTE(scale_slew_times (LIBRARY) (BOOLEAN))
DEFINE_ATTRIBUTE(cell_footprint (CELL) (STRING))
DEFINE_ATTRIBUTE(dont_fault (CELL|PIN) (STRING))
DEFINE_ATTRIBUTE(auxiliary_pad_cell (CELL) (BOOLEAN))
DEFINE_ATTRIBUTE(pad_type (CELL) (STRING))
DEFINE_ATTRIBUTE(complementary_pin (PIN) (STRING))
DEFINE_ATTRIBUTE(x_function (PIN) (STRING))
DEFINE_ATTRIBUTE(fault_model (PIN) (STRING))
DEFINE_ATTRIBUTE(test_output_only (PIN) (BOOLEAN))
in_place_swap_mode("match_footprint")
scale_slew_times(FALSE)
// model section
TIMING_Model(net_delay_trans_degMod
    (Spline
        (OUTPUT_SLEW_AXIS 0.020000 0.100000 0.300000 0.600000
                1.000000 2.000000 3.000000)
        (RC_PRODUCT_AXIS 0.000000 0.500000 1.000000 3.000000
```

```
                6.000000 7.000000)
        data()
    )
    )
TIMING_Model(rise_transition_degradation net_delay_trans_degMod
    (Spline
        data
        (
        (0.020000 1.077692 2.176863 6.576292 13.176147 15.376125)
        (0.100000 1.016667 2.100000 6.487097 13.083607 15.283098)
        (0.300000 0.987500 1.992308 6.300000 12.871428 15.067123)
        (0.600000 1.100000 1.975000 6.100000 12.600000 14.784210)
        (1.000000 1.366667 2.100000 5.950000 12.314285 14.475000)
        (2.000000 2.220000 2.733333 5.960000 11.900000 13.977777)
        (3.000000 3.157143 3.550000 6.300000 11.800000 13.780000)
        )
    )
    )
TIMING_Model(fall_transition_degradation net_delay_trans_degMod
    (Spline
        data
        (
        (0.020000 1.077692 2.176863 6.576292 13.176147 15.376125)
        (0.100000 1.016667 2.100000 6.487097 13.083607 15.283098)
        (0.300000 0.987500 1.992308 6.300000 12.871428 15.067123)
        (0.600000 1.100000 1.975000 6.100000 12.600000 14.784210)
        (1.000000 1.366667 2.100000 5.950000 12.314285 14.475000)
        (2.000000 2.220000 2.733333 5.960000 11.900000 13.977777)
        (3.000000 3.157143 3.550000 6.300000 11.800000 13.780000)
        )
    )
    )
TIMING_Model(wire_delay_rise
    (Spline
        (OUTPUT_SLEW_AXIS 0.020000 0.100000 0.300000 0.600000
            1.000000 2.000000 3.000000)
        (RC_PRODUCT_AXIS 0.000000 0.500000 1.000000 3.000000
            6.000000 7.000000)
        data
        (
        (0.000000 0.355769 0.705882 2.105960 4.205980 4.905983)
        (0.000000 0.375000 0.727273 2.129032 4.229508 4.929577)
        (0.000000 0.406250 0.769231 2.181818 4.285714 4.986301)
        (0.000000 0.431818 0.812500 2.250000 4.363636 5.065790)
        (0.000000 0.450000 0.850000 2.325000 4.457143 5.162500)
        (0.000000 0.470000 0.900000 2.460000 4.650000 5.366667)
        (0.000000 0.478571 0.925000 2.550000 4.800000 5.530000)
```

```
        )
    )
    )
TIMING_Model(wire_delay_fall
    (Spline
        (OUTPUT_SLEW_AXIS 0.020000 0.100000 0.300000 0.600000
            1.000000 2.000000 3.000000)
        (RC_PRODUCT_AXIS 0.000000 0.500000 1.000000 3.000000
            6.000000 7.000000)
        data
        (
        (0.000000 0.355769 0.705882 2.105960 4.205980 4.905983)
        (0.000000 0.375000 0.727273 2.129032 4.229508 4.929577)
        (0.000000 0.406250 0.769231 2.181818 4.285714 4.986301)
        (0.000000 0.431818 0.812500 2.250000 4.363636 5.065790)
        (0.000000 0.450000 0.850000 2.325000 4.457143 5.162500)
        (0.000000 0.470000 0.900000 2.460000 4.650000 5.366667)
        (0.000000 0.478571 0.925000 2.550000 4.800000 5.530000)
        )
    )
    )
// properties section
PROPERTIES(
    Proc_Var(1.000000)
    Voltage(3.300000)
    Temperature(25.000000)
    Proc_Mult(1.000000)
    Volt_Mult(1.000000)
    Temp_Mult(1.000000)
    SLEW_DEGRADATION(RISE(rise_transition_degradation)
        FALL(fall_transition_degradation))
    Wire_Delay(RISE(wire_delay_rise) FALL(wire_delay_fall))
    // WireLoad  Models
    )
CELL(abc
    cell_footprint("5mil")
    dont_fault("sa0")
    auxiliary_pad_cell(FALSE)
    pad_type("clock")
    // model section
    PIN(a
        PINTYPE(OUTPUT )
        complementary_pin("b")
        x_function("b")
        fault_model("none")
        dont_fault("sa01")
        test_output_only(FALSE)
```

```
    )
PIN(b
    PINTYPE(INPUT )
    )
PIN(o
    PINTYPE(OUTPUT )
    )
)
```

9

TLF File Utilities

The timing library format (TLF) contains several utilities that you can use to encrypt, merge, convert, and generate TLF files. The TLF files can be generated from the Synopsys .lib format.

You can find the TLF utilities in your installation hierarchy in the `tools` directory.

```
.../tools/tlfUtil/bin/tlfMerge
.../tools/tlfUtil/bin/tlfConvert
.../tools/tlfUtil/bin/tlfEncrypt
.../tools/tlfUtil/bin/syn2tlf
```

This chapter contains information about the following encryption and conversion utilities:

tlfEncrypt

Syntax

```
tlfEncrypt in_file out_file
            [ -help] [ -version ]
```

Description

Encrypts an ASCII TLF file. This can be done to protect proprietary vendor data. The utility does syntax and semantic/data validation checks on the input library before encrypting it. The syntax issues are displayed while reading the library file. If the input TLF file has semantic or data integrity issues, a log file, libcheck.log, is generated. The various data integrity checks performed on the input library file are listed below:

1. Absence of clock pin in sequential cell.
2. Unspecified/zero nominal conditions (PVT values).
3. Cells with non-increasing table data.
4. Delay characterization (threshold) points unspecified.
5. Cells with zero/unspecified input/inout pin capacitance.
6. Combinational cells with no functional description for output/inout pins.
7. Sequential cells with no sequential block (Register/Latch) description.
8. Cells with zero slew table data for one or more non_Z transitions (with Z1/Z0/01/10 transitions).

Note: This utility can be used on any ASCII TLF file, regardless of version.

Arguments

in_file
: Specifies the ASCII TLF file to encrypt.

out_file
: Specifies the name of the encrypted TLF file to create.

–help
: Prints the `tlfEncrypt` command line usage information.

`-version`

Prints the `tlfEncrypt` version information.

tlfMerge

Syntax

```
tlfMerge timing_in_file power_in_file merged_out_file
            [ -help] [ -version ]
```

Description

Merges TLF `4.1` timing and power library data into a single TLF `4.1/4.3` file. This utility can also be used to merge TLF *3.X* timing library data with TLF `4.1` power library data into a single TLF `4.1/4.3` file.

If the input timing library is in TLF`4.1` format, `tlfMerge` generates merged output TLF file in default TLF units. However, if the input timing library is in TLF`4.3` format, the merged output library is in unit defined in the input timing library.

Note: If the units specified in the power library and input timing library are different, power library units are scaled to become similar to the input timing library units before merging is done.

Arguments

`timing_in_file`
: Specifies the name of the input TLF file to merge.

`power_in_file`
: Specifies the name of the input power library file to merge.

`merge_out_file`
: Specifies the name of the merged TLF file to create.

`–help`
: Prints the `tlfMerge` command line usage information.

`–version`
: Prints the `tlfMerge` version information.

tlfConvert

Syntax

```
tlfConvert tlf3.X_file tlf4.1_file
            [ –help] [ –version | –v] [ –precision ]
```

Description

The tlfConvert utility converts CTLF or TLF 3.*X* library data into encrypted or ASCII TLF `4.1` library data.

TLF `4.1` does not require library compilation; and clear-text or encrypted-text TLF (ETLF) libraries can be read directly into all the applications.

Note: TLF `4.1` libraries that are created from TLF 3.*X* libraries via `tlfConvert` will not contain the necessary information to fully support the Pompeii (version 3.4) signal and design integrity flow.

Arguments

`tlf3.X_file`
: Specifies the name of the CTFL/TLF 3.*X* file to convert.

`tlf4.1_file`
: Specifies the name of the encrypted/ASCII TLF 4.1 file to create.

`–help`
: Prints the `tlfConvert` command line usage information.

`–version | –v`
: Prints the tlfConvert version information.

`–precision`
: Sets the precision of the generated values.

syn2tlf

Syntax

`syn2tlf <arguments> input_file`

Description

Converts Synopsys `.lib` to either TLF 3.1, 4.1, 4.2 or 4.3.

Note: All the syn2tlf options are case insensitive.

Arguments

`-h`

Prints the `syn2tlf` command line usage information.

`-h message.id`

Prints a description of the message(s) specified as *`message.id`*.

`-version`

Prints the `syn2tlf` version information.

`-o outputfile`

Specifies the name of the output TLF file.
Default: Synopsys filename root with *.tlf* extension

`-l logfile`

Specifies the name of the log file to be generated by `syn2tlf`.
Default: `syn2tlf.log`

`-p precision`

Sets the precision of the generated values. The argument type is integer, and it can have values from 1 to 10. This option is required if Synopsys time/capacitance units are very small compared to CDC default units.
Default: 6

`-e env_type`

Specifies the Environment Corner for the library. Any string can

be specified as the argument value. Typical argument values are: `commercial`, `military`, `industrial`.

`-n` *`vendor`*

Specifies the name of the ASIC vendor owning the Synopsys file. This is used to set the Vendor() field in the generated TLF file. The argument type is string.

`-cost` *`cost_type`*

Interprets the cost parameter in the Synopsys library. The argument values can be one of the three valid cost-type strings: `area`, `trans_count` and `gate_count`. These values are used to determine whether to translate the cost parameter as `area`, `transistor_count` or `gate_count` respectively.
Default: `area`

`-i` *`tbl_inp_thresh`*

Specifies the `Table_Input_Threshold` parameter value explained in Appendix A.
Type: Float
Value: Between -1 and 1

`–d` *`tbl_outp_thresh`*

Specifies the `Table_Output_Threshold` parameter value explained in Appendix A.
Type: Float
Value: Between -1 and 1

`–s` *`tbl_trans_start`*

Specifies the `Table_Transition_Start` parameter value explained in Appendix A.
Type: Float
Value: Between -1 and 1

`–t` *`tbl_trans_end`*

Specifies the `Table_transition_End` parameter value explained in Appendix A.
Type: Float
Value: Between -1 and 1

`–ir` *`input_threshold_pct_rise`*

Specifies the `input_threshold_pct_rise` value.
Value: Between 0.0 and 100.0

`-if` *`input_threshold_pct_fall`*
Specifies the `input_threshold_pct_fall` value.
Value: Between 0.0 and 100.0

`-dr` *`output_threshold_pct_rise`*
Specifies the `output_threshold_pct_rise` value.
Value: Between 0.0 and 100.0

`-df` *`output_threshold_pct_fall`*
Specifies the `output_threshold_pct_fall` value.
Value: Between 0.0 and 100.0

`-sr` *`slew_lower_threshold_pct_rise`*
Specifies the `slew_lower_threshold_pct_rise` value.
Value: Between 0.0 and 100.0.

`-sf` *`slew_lower_threshold_pct_fall`*
Specifies the `slew_lower_threshold_pct_fall` value.
Value: Between 0.0 and 100.0

`-tr` *`slew_upper_threshold_pct_rise`*
Specifies the `slew_upper_threshold_pct_rise` value.
Value: Between 0.0 and 100.0

`-tf` *`slew_upper_threshold_pct_fall`*
Specifies the `slew_upper_threshold_pct_fall` value.
Value: Between 0.0 and 100.0

Note: Specify all the threshold points on the command line. The tool does not accept incomplete threshold values.

`-slew_measure_lower_rise` *`float value`*
Specifies the rise value for TLF construct `SLEW_MEASURE_LOWER_THRESHOLD_PCT`. This option can be specified for TLF4.2 and higher versions of TLF.
Value: Between 0.0 and 100.0

`-slew_measure_lower_fall` *`value`*
Specifies the fall value for TLF construct `SLEW_MEASURE_LOWER_THRESHOLD_PCT`. This option can be specified for TLF4.2 and higher versions of TLF.
Type: Float
Value: Between 0.0 and 100.0

`-slew_measure_upper_rise` *value*
: Specify the rise value for TLF construct `SLEW_MEASURE_UPPER_THRESHOLD_PCT`. This option can be specified for TLF4.2 and higher versions of TLF.
Type: Float
Value: Between 0.0 and 100.0

`-slew_measure_upper_fall` *value*
: Specifies the fall value for TLF construct `SLEW_MEASURE_UPPER_THRESHOLD_PCT`. This option can be specified for TLF4.2 and higher versions of TLF.
Type: Float
Value: Between 0.0 and 100.0

`-format 3.1 | 4.1 | 4.2 | 4.3`
: Specifies the output TLF file format as either TLF 3.1, 4.1, 4.2 or 4.3.
Default: TLF 4.1

`-verbose`
: Reports all error and warning messages. If this option is not specified, only critical error and warning messages will be reported.

`-process` *value*
: Specifies the process value at which the timing information in the library should be characterized. Use this option only if the `delay_model` is `generic_cmos`.
Type: Float

`-voltage` *value*
: Specifies the voltage value at which the timing information in the library should be characterized. Use this option only if the `delay_model` is `generic_cmos`.
Type: Float

`-temperature` *value*
: Specifies the temperature value at which the timing information in the library should be characterized. Use this option only if the `delay_model` is `generic_cmos`.
Type: Float

input_file

Specifies the input Synopsys library filename.

Note: If warnings and/or error messages are encountered during syn2tlf invocation, the translator creates a log file for storing the messages. However, the error messages (fatal and internal) are also displayed as a standard error.

Example:

```
syn2tlf input.lib
syn2tlf -e com -n TI -o outfile.tlf input.li
syn2tlf -i -0.5 -d 0.5 -s 0.2 -t -0.8 table.lib
syn2tlf -l test.log -cost trans_count test.lib
```

A

Units in TLF

The following table shows the default units in TLF. You can enter values in the TLF file using the default units or you can use the corresponding user defined property to change the default units for the TLF file:

Parameter	User Defined Property	Units	Default
area	AREA_UNIT	square micrometer	1squ
capacitance	CAP_UNIT	picofarad	1pF
conductance	CONDUCTANCE_UNIT	millisiemen	1mS
current	CURRENT_UNIT	milliampere	1mA
inductance	INDUCTANCE_UNIT	picohenry	1pH
power	POWER_UNIT	milliwatt	1mW
resistance	RES_UNIT	kiloOhm	1kohm
temperature	TEMPERATURE_UNIT	degree Celcius	1C
time	TIME_UNIT	nanosecond	1ns
voltage	VOLT_UNIT	volt	1V

B

Gate Ensemble and Silicon Ensemble TLF Library Requirements

This chapter contains the following information:

Overview

To support the timing-driven design methodology and reduce the duplication of data, all ASIC timing information is stored in the Cadence Timing Library Format (TLF). TLF is an ASCII representation of the timing models for the cells in a library. This eliminates the unnecessary task of duplicating the same timing information in multiple application libraries. Besides reading the Library Exchange Format (LEF), Gate Ensemble and Silicon Ensemble also have the capability to retrieve the nonlinear data from the CTLF libraries using the procedural access routines.

This appendix describes the required TLF data for the Table algorithm that Gate Ensemble and Silicon Ensemble will use to calculate delays.

This appendix is intended primarily for semiconductor vendors who wish to start developing TLF libraries to support version 4.5 and up of the Gate and Silicon Ensemble tools. Knowledge of LEF and TLF syntax are assumed. This appendix does not cover how the Table algorithm and the TLF data are used in the Gate and Silicon Ensemble tools.

Requirements

The following TLF information is used by Gate Ensemble and Silicon Ensemble to compute the component (device) delay that includes the intrinsic delay and the Extra Source Gate Delay through the component (device), due to the load on the output net.

- **Spline** delay models

- Propagation delay tables for rise and fall transitions are two-dimensional tables which are the function of output load and input edge rate. Output load is the effective load capacitance, and the input edge rate depends on the timing arc polarity and the signal direction.
- Output transition time tables for rise and fall transitions can be one- or two-dimensional tables. They contain the output edge rates which are the function of output load and/or input edge rate. Different PVT derating factors can be specified for the propagation and transition time models. *However, the recommended method is to enter the prederated delay values for all environment corners in the TLF files.* Refer to the *Delay Calculation Algorithm Guide* for more information about the TLF Table algorithm.

- **Polarity** of each timing arc
- **Current** and **Energy** models

  To identify voltage drop and electromigration problems, the place-and-route power analyzer requires that you use one of these power models:

  - Dynamic power consumption model, where supply and ground current are modeled as waveforms. The software supports these waveforms: average, triplet, piecewise linear, and wavetable.
  - Energy consumption per transition model, where you use the energy of a given power arc and associated parameters to generate a lookup table. This model does not model dynamic power (peaks and dips), which you need to identify voltage drop problems. There are three energy models: short circuit energy consumption, internal energy of the cell, and total energy.

- **Wavetable** Current models

  The wavetable is the most accurate way to model current waveforms for dynamic power consumption. The wavetable is a lookup table of current waveforms, where each waveform can have its own shape.

- Power pin types (**Supply**, **Ground**)

  Supply current is the power consumed in charging the load and includes its associated short circuit current. Ground current is the power consumed in discharging the load, and includes its associated short circuit current.

- **Crosstalk**, **VDROP** limits

  Crosstalk limits (`CT_TOLERANCE`) on pins, cells, or the library, set positive and negative limits. The place-and-route tools use these limits to identify areas where there could be crosstalk-induced logic glitches or delay problems. The voltage drop limit

(VDROP_LIMIT) is a property set on pins or cells, which specifies how much the supply can vary before there is a voltage drop problem.

- **Fluence** and Fluence limits

  Fluence is a measure, over the life of the chip, of the flux of injected hot electrons. You set a fluence per transition value (FLUENCE) and a transition limit for the cell (FLUENCE_LIMIT). The place-and-route tools use these values to identify hot electron effects.

Example

The following example shows CT_TOLERANCE and FLUENCE limits for a 2-input AND gate (ssad2).

```
...
Area(4.000000)
VDROP_LIMIT(0.1)
    PIN(A
        PINTYPE(INPUT )
        CT_TOLERANCE(Positive(0.73105) Negative(0.51895))
        Capacitance(0.010841)
    )
    PIN(B
        PINTYPE(INPUT )
        CT_TOLERANCE(Positive(0.76274) Negative(0.48726))
        Capacitance(0.009429)
    )
    PIN(Y
        PINTYPE(OUTPUT )
        FUNCTION( (A & B) )
        // 1 year @ 10 MHz
        FLUENCE_Limit ( 1.73448000E+12 )
        Capacitance(0.000000)
    )
...
```

C

Developing TLF Libraries for CT-Gen and the Ultra Router

This chapter contains the following information:

Overview

To support the timing-driven design methodology and reduce the duplication of data, all ASIC timing information is stored in the Cadence Timing Library Format (TLF). TLF is an ASCII representation of the timing models for the cells in a library. The TLF file can be optionally encrypted by the semiconductor vendors. TLF (either encrypted or not) is used by all Cadence products in the timing-driven flow. This eliminates the unnecessary task of duplicating the same timing information in multiple application libraries. The Envisia™ ultra router and Envisia clock tree generator (CT-Gen) retrieve the nonlinear delay data from the TLF libraries.

This appendix describes the required TLF data for the nonlinear delay model that is used by the ultra router and CT-Gen products.

This appendix is intended primarily for semiconductor vendors who wish to start developing TLF libraries to support the ultra router and CT-Gen. This appendix does not cover how TLF data is used in the ultra router and CT-Gen.

Requirements

Improved integrated circuit processes have increased the demand for more accurate delay modeling to account for nonlinear effects at the submicron level. Both the ultra router and CT-Gen support the nonlinear delay model for accurate delay calculation.

The ultra router and CT-Gen expect certain timing data to be specified in the TLF library to have successful ultra router and CT-Gen runs. This appendix lists the required TLF timing data.

Pin-to-Pin Delays

Propagation delay tables for rise and fall transitions are two-dimensional tables which are a function of the output load and input edge rate. The output load is the effective load capacitance, and the input edge rate depends on the timing arc polarity and the signal direction.

Output transition time tables for rise and fall transitions can be one- or two-dimensional tables. They contain the output edge rates which are the function of output load and/or input edge rate.

Note: If the paths have any state dependencies, you must specify them as conditional expressions in the PATH statements.

Setup and Hold Timing Checks

The timing checks can be specified as constant, one-dimensional, or two-dimensional tables. For two-dimensional tables, these timing checks are the function of input transition of the data and the clock pins. You can specify them as one-dimensional tables when the setup/hold depends only on the input transition of the data or the clock.

Note: If the timing checks have any state dependencies, you must specify them as conditional expressions in the timing check statements.

Insertion Delays

For large macros with embedded clocks, CT-Gen needs to know the minimum and maximum insertion delays from the clock pin to the internal clock target pins (clock pins on flip-flops and latches). The insertion delay includes delay through both wires and cells, such as buffers and inverters. Normally there will be different paths from the clock pin to each of the clock target pins. A minimum and a maximum insertion delay should be specified so the ultra router and CT-Gen will know the range of the delays along these paths.

The insertion delay can be calculated from the model information included in the INSERTION_DELAY statement. For each pin driving an internal clock tree, you must specify two Insertion_Delay statements (a FAST and a SLOW statement) for each input transition—internal transition pair for which there is an insertion delay.

Load Limit

The load limit is the total capacitive load allowed for an output pin. Vendors should provide maximum load limits on all output pins. The TLF syntax is:

```
Load_Limit(Warn(value) Error(value))
```

Slew Limit

The slew limit is the maximum transition on all input and output pins. Vendors should provide maximum transition limits on all input pins and/or all output pins. The TLF syntax is:

```
Slew_Limit(Warn(value) Error(value))
```

Pin Type and Direction

All the pin types must be correctly specified, see PINTYPE statement. All clock pins must be identified with the CLOCK_PIN statement.

Pin Function

All buffer and inverter cells must have the correct output pin function declaration for the ultra router and CT-Gen to run. You must include the expression with the output pins using a FUNCTION statement.

For example, if *A* is the input pin of a buffer and *Y* is the output pin, the TLF description for pin *Y* is similar to:

```
Pin(Y Pintype(output) Function(A))
```

If the cell was an inverter, the description for the output pin would be:

```
Pin(Y Pintype(output) Function(!A))
```

Note: The ultra router uses the output pin function declarations to determine the cells that are logically equivalent and thus can be substituted during resizing.

PVT Derating

Different process, voltage, and temperature (PVT) derating factors can be specified for the table models. *However, the recommended method is to enter the prederated delay values for all environment corners in the TLF files.*

D

TLF Property Index

This appendix indexes all TLF properties and statements. The same information is provided in two ways:

- Alphabetic Index of Properties
- Index of Properties Grouped by Purpose

Both tables show the sections in which the property can be used, and provide notes on the usage. Click on the property name to see the description page. Some properties (library level and lower) are described in Chapter 5, “Properties,” other properties and keywords are described in Chapter 6, “TLF Statements.”

Alphabetic Index of Properties

The following table shows all properties and keywords in alphabetical order.

Table 4-1 TLF Properties and Keywords in Alphabetical Order

Predefined Property Name	Purpose	Context	Notes
ADDRESS_BUS	memory	MEMORY_BUS, CELL	New 4.1
ADDRESS_WIDTH	memory	MEMORY_PROPS, CELL	New 4.1
AREA	cost function	PROPERTIES, CELL	
AREA_UNIT	units	UNIT, PROPERTIES	New 4.1
AVAILABLE_TRACK	routing	ROUTING_PROPS, CELL	New 4.1
BUS	bus specification	CELL	New 4.1
BUSMODE	memory	MEMORY_BUS, CELL	
BUSTYPE	bus specification	BUS, CELL	New 4.1
CAPACITANCE	bus specification	PROPERTIES, CELL, PIN, BUS	New 4.1
CAP_UNIT	units	UNIT, PROPERTIES	New 4.1
CELL	TLF structure	Library	
CT_RES_LOW	crosstalk resistance	lIBRARY, CELL, PIN	New 4.2
CT_RES_HIGH	crosstalk resistance	lIBRARY, CELL, PIN	New 4.2
CELL_SPOWER	power	PROPERTIES, CELL	New 4.1
CELLTYPE	obsolete		Replaced. See IGNORE_CELL and PAD_CELL.
CLEAR	pin data	LATCH, REGISTER	

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
CLEAR_PRESET_VAR1	pin data	LATCH, REGISTER, TEST_REGISTER, TEST_LATCH, CELL	New 4.1
CLEAR_PRESET_VAR2	pin data	LATCH, REGISTER, TEST_REGISTER, TEST_LATCH, CELL	New 4.1
CLOCK	memory	MEMORY_BUS, LATCH, REGISTER, CELL	Modified 4.1. See also CLOCK_PIN.
CLOCK_PIN	TLF structure	PIN	New 4.1
COND	delay timing check, conditional	PATH, PATH_EXTENSION, timing_check, CELL	
COND_END	timing check, conditional	SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE, timing_check, CELL	
COND_START	timing check, conditional	SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE, timing_check, CELL	
CONDUCTANCE_UNIT	units	UNIT, PROPERTIES	New 4.1
CSAT	threshold	PAD_PROPS, BUS, PIN	New 4.1
CSBT	threshold	PAD_PROPS, BUS, PIN	New 4.1
CTTAT	threshold	PAD_PROPS, BUS, PIN	New 4.1
CTTBT	threshold	PAD_PROPS, BUS, PIN	New 4.1
CT_TOLERANCE	signal/design integrity	PROPERTIES, CELL, PIN, BUS	New 4.1

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
CURRENT_UNIT	units	UNIT, PROPERTIES	New 4.1
DATA_WIDTH	memory	MEMORY_PROPS, CELL	New 4.1
DATE	TLF structure	HEADER	
DCURRENT	pad modelling	PAD_PROPS, BUS, PIN	New 4.1
DEFAULT_LOAD	default val	PROPERTIES, CELL, PIN	
DEFAULT_PVT_COND	custom operating condition	PROPERTIES	New 4.1
DEFAULT_SLEW	default val	PROPERTIES, CELL, PIN	
DEFAULT_WIRELOAD_GROUP	wire load	PROPERTIES	New 4.1
DEFAULT_WIRELOAD_MODE	wire load	PROPERTIES	New 4.1
DELAY	delay time	PATH, PATH_EXTENSION, CELL	
DONT_TOUCH	synthesis	CELL	New 4.1
DONT_USE	synthesis	CELL	New 4.1
DRIVETYPE	pin specification	BUS, PIN	New 4.1
ENABLE	memory	MEMORY_BUS, CELL, BUS, PIN	Modified 4.1
ENVIRONMENT	TLF structure.	HEADER	
EQ_CELLS	cell specification	Library	
EQ_PINS	pin specification	CELL	

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
ERROR	error limit	LOAD_LIMIT, SLEW_LIMIT, FLUENCE_LIMIT, FANOUT_LIMIT, VDROP_LIMIT	
FALL	edge indicators	DEFAULT_SLEW, SLEW_LIMIT, PROC_MULT, VOLT_MULT, TEMP_MULT	
FANOUT_LIMIT	design rule check	PROPERTIES, CELL, BUS, PIN	New 4.1
FANOUT_MIN	design rule check	PROPERTIES, CELL, BUS, PIN	New 4.1
FOR_CELL	obsolete		Removed 4.1
FOR_PIN	TLF structure	PROPERTIES, CELL	Modified 4.1
FLUENCE	signal/design integrity	CELL, PATH	New 4.1
FLUENCE_LIMIT	signal/design integrity	CELL, PIN	New 4.1
FUNCTION	pin specification	PIN, BUS	
GATE_COUNT	cost function	PROPERTIES, CELL	
GENERATED_BY	TLF structure	HEADER	
GROUND_CURRENT	power	CELL, PATH, BUS, PIN	New 4.1
HEADER	TLF structure	beginning of file	
HOLD	timing check	NO_CHANGE, CELL	
HYSTERESIS	pad modelling	PAD_PROPS, BUS, PIN	New 4.1
IGNORE_CELL	TLF structure	CELL	New 4.1
INDUCTANCE_UNIT	units	UNIT, PROPERTIES	New 4.1

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
INPUT	pin specification	LATCH, REGISTER, PINTYPE, CELL, PIN	
INPUT_FANLOAD	design rule check	PROPERTIES, CELL, BUS, PIN	New 4.1
INPUT_THRESHOLD_PCT		PROPERTIES, CELL	New 4.2
INPUT_VOLTAGE	pad modelling	PROPERTIES, PIN	New 4.1
INSERTION_DELAY	pin specification	CELL	
INTERNAL_ENERGY	power	CELL, PATH, BUS, PIN	New 4.1
INVERTED_OUTPUT	pin data	LATCH, REGISTER, CELL	
LATCH	cell specification	CELL	
LIBRARY	TLF structure	HEADER	
LOAD_LIMIT	error limit	PROPERTIES, CELL, PIN	
LOAD_MIN	design rule check	PROPERTIES, CELL, BUS, PIN	New 4.1
MAP_TO_STPIN	state table	PIN	New 4.1
MEMORY_BUS	memory	CELL	New 4.1
MEMORY_OPR	memory	MEMORY_PROPS, PROPERTIES, CELL	New 4.1
MEMORY_PROPS	memory	PROPERTIES, CELL	New 4.1
MEMORY_TYPE	memory	MEMORY_PROPS, PROPERTIES, CELL	New 4.1
MIN_POROSITY	routing	PROPERTIES	New 4.1
MOBILITY_LIMIT	signal/design integrity	CELL, BUS, PIN	New 4.1
MODEL	obsolete		Obsolete 4.1 See usage_MODEL

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
MPWH	timing check	CELL	
MPWL	timing check	CELL	
NEGATIVE	signal/design integrity	CT_TOLERANCE	New 4.1
NET_CAP	wire load	PROPERTIES, WIRELOAD, WIRELOAD_BY_XXX	
NET_RES	wire load	PROPERTIES, WIRELOAD, WIRELOAD_BY_XXX	
NO_CHANGE	timing check	CELL	
OTHER_PINS	timing check	PATH, PATH_EXTENSION, MPWH, MPWL, PERIOD	
OUTPUT	pin specification	LATCH, REGISTER, PINTYPE, CELL, PIN	
OUTPUT_THRESHOLD_PCT		CELL, PROPERTIES	New 4.2
OUTPUT_VOLTAGE	pad modelling	PROPERTIES, PIN	New 4.1
PAD_CELL	TLF structure	CELL	New 4.1
PAD_PIN	TLF structure	BUS, PIN	New 4.1
PAD_PROPS	pad modelling	BUS, PIN	New 4.1
PATH	delay specification	timing_check, CELL	
PATH_EXTENSION	delay specification	timing_check, CELL	
PERIOD	timing check	CELL	
PIN	cell or bus specification	CELL, BUS	
PIN_CAP	obsolete		Renamed. See CAPACITANCE

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
PIN_SPOWER	power	PROPERTIES, CELL, BUS, PIN	New 4.1
PINDIR	obsolete		Renamed. See PINTYPE
PINTYPE	pin specification	PIN	Modified
POSITIVE	signal/design integrity	CT_TOLERANCE	New 4.1
POWER_ESTIMATE	cost function	PROPERTIES, CELL	
POWER_UNIT	units	UNIT, PROPERTIES	New 4.1
PROC_MULT PROC_MULT_modeltype	custom operating condition	PROPERTIES, CELL	modeltypes updated in 4.1
PROC_VAR	environment	PROPERTIES	
PROPERTIES	TLF structure	LIBRARY	New 4.1
PULL	pin specification	BUS, PIN	New 4.1
PULL_CURRENT	pin specification	BUS, PIN	New 4.1
PULL_RESISTANCE	pin specification	BUS, PIN	New 4.1
PVT_CONDS	custom operating condition	PROPERTIES	New 4.1
RECOVERY	timing check	CELL	
REGISTER	cell specification	CELL	
REMOVAL	timing check	CELL	
RES_UNIT	units	UNIT, PROPERTIES	New 4.1

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
RISE	edge indicators	DEFAULT_SLEW, SC_ENERGY, CSAT, CSBT, CTTAT, CTTBT, SLEW_LIMIT, PROC_MULT, VOLT_MULT, TEMP_MULT	
ROUTING_LAYER	routing	Library	New 4.1
ROUTING_PROPS	routing	CELL	New 4.1
SC_ENERGY	power	CELL, PATH, BUS, PIN	New 4.1
SCAN_EQUIVALENT	scan cell modeling	CELL	New 4.1
SCAN_PINTYPE	scan cell modeling	PIN	New 4.1
SDF_COND	delay condition	PATH, PATH_EXTENSION, timing_check, CELL	
SDF_COND_END	timing check, conditional	SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE, CELL	
SDF_COND_START	timing check, conditional	SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE, CELL	
SET	pin data	LATCH, REGISTER, CELL	
SETUP	timing check	timing_check, NO_CHANGE, CELL	
SKEW	timing check	CELL	
SLAVE_CLOCK	pin data	LATCH, REGISTER, CELL	

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
SLEW	delay specification	PATH, PATH_EXTENSION, CELL	
SLEW_DEGRADATION	delay specification	PROPERTIES	
SLEW_LIMIT	error limit	PROPERTIES, CELL, PIN, BUS	
SLEW_LOWER_THRESHOLD_PCT		CELL, PROPERTIES	New 4.2
SLEW_MEASURE_LOWER_THRESHOLD_PCT		CELL, PROPERTIES	New 4.2
SLEW_MEASURE_UPPER_THRESHOLD_PCT		CELL, PROPERTIES	New 4.2
SLEW_MIN	design rule check	PROPERTIES, CELL, PIN, BUS	New 4.1
SLEW_UPPER_THRESHOLD_PCT		CELL, PROPERTIES	New4.2
STATE_TABLE	state table	CELL	New 4.1
SUPPLY_CURRENT	power	CELL, PATH, BUS, PIN	New 4.1
TABLE_INPUT_THRESHOLD	threshold	PROPERTIES, CELL	Modified 4.1
TABLE_OUTPUT_THRESHOLD	threshold	PROPERTIES, CELL	Modified 4.1
TABLE_TRANSITION_START	threshold	PROPERTIES, CELL	Modified 4.1
TABLE_TRANSITION_END	threshold	PROPERTIES, CELL	Modified 4.1
TECHNOLOGY	TLF structure	HEADER	
TEMPERATURE	environment	PROPERTIES	
TEMPERATURE_UNIT	units	UNIT, PROPERTIES	New 4.1
TEMP_MULT TEMP_MULT_modeltype	custom operating condition	PROPERTIES, CELL	modeltypes updated in 4.1
TEST_LATCH	scan cell modeling	CELL	New 4.1

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
TEST_REGISTER	scan cell modeling	CELL	New 4.1
THRESHOLD_LIMIT	threshold	CELL, BUS, PIN	New 4.1
TIME_UNIT	units	UNIT, PROPERTIES	New 4.1
TIMING_PROPS	obsolete		Renamed. See PROPERTIES
TLF_VERSION	TLF structure	HEADER	
TOTAL_ENERGY	power	CELL, PATH, BUS, PIN	New 4.1
TRACK_AREA	routing	ROUTING_PROPS, CELL	
TRANSISTOR_COUNT	cost function	PROPERTIES, CELL	3.0
TREE_TYPE	custom operating condition	PVT_CONDS, PROPERTIES	New 4.1
UNIT	units	PROPERTIES	New 4.1
usage_MODEL	pad modelling	PROPERTIES, CELL	Modified 4.1
VENDOR	TLF structure	HEADER	
VERSION	TLF structure	HEADER	
VDROP_LIMIT	signal/design integrity	CELL, PIN	New 4.1
VOLTAGE	environment	PROPERTIES	
VOLT_HIGH_THRESHOLD	pad modelling	INPUT_VOLTAGE, OUTPUT_VOLTAGE, PROPERTIES, PIN	New 4.1
VOLT_LOW_THRESHOLD	pad modelling	INPUT_VOLTAGE, OUTPUT_VOLTAGE, PROPERTIES, PIN	New 4.1
VOLT_MAX	pad modelling	INPUT_VOLTAGE, OUTPUT_VOLTAGE, PROPERTIES, PIN	New 4.1

Table 4-1 TLF Properties and Keywords in Alphabetical Order, *continued*

Predefined Property Name	Purpose	Context	Notes
VOLT_MIN	pad modelling	INPUT_VOLTAGE, OUTPUT_VOLTAGE, PROPERTIES, PIN	New 4.1
VOLT_MULT VOLT_MULT_modeltype	custom operating condition	PROPERTIES, CELL	modeltypes updated in 4.1
VOLT_UNIT	units	UNIT, PROPERTIES	New 4.1
WARN	error limit	LOAD_LIMIT, SLEW_LIMIT, FLUENCE_LIMIT, FANOUT_LIMIT, VDROP_LIMIT	
WAVEFORM_TAIL_RES	transient driver resistance	CELL, PATH, PIN	New 4.2
WAVETABLE	current	usage_MODEL	New 4.1
WIRELOAD	wire load	PROPERTIES	
WIRELOAD_BY_XXX	wire load	PROPERTIES	

Index of Properties Grouped by Purpose

The following table contains the same data shown in Table 4-1 on page 449. The data is organized by purpose in this table.

Table 4-2 TLF Properties and Keywords Grouped by Purpose

Predefined Property Name	Context	Notes
Bus Specification		
BUS	CELL	New 4.1
BUSTYPE	BUS, CELL	New 4.1
CAPACITANCE	PROPERTIES, CELL, PIN, BUS	New 4.1
PIN	CELL, BUS	
Cell Specification		
EQ_CELLS	Library	
LATCH	CELL	
PIN	CELL, BUS	
REGISTER	CELL	
Cost Function		
AREA	PROPERTIES, CELL	
GATE_COUNT	PROPERTIES, CELL	
POWER_ESTIMATE	PROPERTIES, CELL	
TRANSISTOR_COUNT	PROPERTIES, CELL	3.0
Current		
WAVETABLE	usage_MODEL	New 4.1
Custom Operating Condition		
PROC_MULT PROC_MULT_modeltype	PROPERTIES, CELL	modeltypes updated in 4.1
PVT_CONDS	PROPERTIES	New 4.1

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
TEMP_MULT TEMP_MULT_modeltype	PROPERTIES, CELL	modeltypes updated in 4.1
TREE_TYPE	PVT_CONDS, PROPERTIES	New 4.1
VOLT_MULT VOLT_MULT_modeltype	PROPERTIES, CELL	modeltypes updated in 4.1
	PROPERTIES, CELL	
DEFAULT_PVT_COND	PROPERTIES	New 4.1
Default Values		
DEFAULT_LOAD	PROPERTIES, CELL, PIN	
DEFAULT_SLEW	PROPERTIES, CELL, PIN	
Delay Conditions		
SDF_COND	PATH, PATH_EXTENSION, timing_check, CELL	
Delay Specification		
PATH	timing_check, CELL	
PATH_EXTENSION	timing_check, CELL	
SLEW	PATH, PATH_EXTENSION, CELL	
SLEW_DEGRADATION	PROPERTIES	
DELAY	PATH, PATH_EXTENSION, CELL	
COND	PATH, PATH_EXTENSION, timing_check, CELL	
Design Rule Check		
FANOUT_LIMIT	PROPERTIES, CELL, BUS, PIN	New 4.1
FANOUT_MIN	PROPERTIES, CELL, BUS, PIN	New 4.1

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
INPUT_FANLOAD	PROPERTIES, CELL, BUS, PIN	New 4.1
LOAD_MIN	PROPERTIES, CELL, BUS, PIN	New 4.1
SLEW_MIN	PROPERTIES, CELL, PIN, BUS	New 4.1
Edge Indicators		
FALL	DEFAULT_SLEW, SC_ENERGY, CSAT, CSBT, CTTAT, CTTBT, SLEW_LIMIT, PROC_MULT, VOLT_MULT, TEMP_MULT	
RISE	DEFAULT_SLEW, SC_ENERGY, CSAT, CSBT, CTTAT, CTTBT, SLEW_LIMIT, PROC_MULT, VOLT_MULT, TEMP_MULT	
Environment		
PROC_VAR	PROPERTIES	
TEMPERATURE	PROPERTIES	
VOLTAGE	PROPERTIES	
Error Limits		
ERROR	LOAD_LIMIT, SLEW_LIMIT, FLUENCE_LIMIT, FANOUT_LIMIT, VDROP_LIMIT	
LOAD_LIMIT	PROPERTIES, CELL, PIN	
SLEW_LIMIT	PROPERTIES, CELL, PIN, BUS	

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
WARN	LOAD_LIMIT, SLEW_LIMIT, FLUENCE_LIMIT, FANOUT_LIMIT, VDROP_LIMIT	
Memory		
ADDRESS_BUS	MEMORY_BUS, CELL	New 4.1
ADDRESS_WIDTH	MEMORY_PROPS, CELL	New 4.1
BUSMODE	MEMORY_BUS, CELL	
CLOCK	MEMORY_BUS, LATCH, REGISTER, CELL	Modified 4.1. See also CLOCK_PIN.
DATA_WIDTH	MEMORY_PROPS, CELL	New 4.1
ENABLE	MEMORY_BUS, CELL, BUS, PIN	Modified 4.1
MEMORY_BUS	CELL	New 4.1
MEMORY_OPR	MEMORY_PROPS, PROPERTIES, CELL	New 4.1
MEMORY_PROPS	PROPERTIES, CELL	New 4.1
MEMORY_TYPE	MEMORY_PROPS, PROPERTIES, CELL	New 4.1
Obsolete		
CELLTYPE		Replaced. See IGNORE_CELL and PAD_CELL.
FOR_CELL		Removed 4.1
MODEL		Obsolete 4.1 See usage MODEL
PIN_CAP		Renamed. See CAPACITANCE
PINDIR		Renamed. See PINTYPE

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
TIMING_PROPS		Renamed. See PROPERTIES
Pad Modeling		
DCURRENT	PAD_PROPS, BUS, PIN	New 4.1
HYSTERESIS	PAD_PROPS, BUS, PIN	New 4.1
INPUT_VOLTAGE	PROPERTIES, PIN	New 4.1
OUTPUT_VOLTAGE	PROPERTIES, PIN	New 4.1
PAD_PROPS	BUS, PIN	New 4.1
usage_MODEL	PROPERTIES, CELL	Modified 4.1
VOLT_HIGH_THRESHOLD	INPUT_VOLTAGE, OUTPUT_VOLTAGE, PROPERTIES, PIN	New 4.1
VOLT_LOW_THRESHOLD	INPUT_VOLTAGE, OUTPUT_VOLTAGE, PROPERTIES, PIN	New 4.1
VOLT_MAX	INPUT_VOLTAGE, OUTPUT_VOLTAGE, PROPERTIES, PIN	New 4.1
VOLT_MIN	INPUT_VOLTAGE, OUTPUT_VOLTAGE, PROPERTIES, PIN	New 4.1
Pin Data		
CLEAR	LATCH, REGISTER	
CLEAR_PRESET_VAR1	LATCH, REGISTER, TEST_REGISTER, TEST_LATCH, CELL	New 4.1
CLEAR_PRESET_VAR2	LATCH, REGISTER, TEST_REGISTER, TEST_LATCH, CELL	New 4.1
INVERTED_OUTPUT	LATCH, REGISTER, CELL	

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
SET	LATCH, REGISTER, CELL	
SLAVE_CLOCK	LATCH, REGISTER, CELL	
Pin Specification		
DRIVETYPE	BUS, PIN	New 4.1
EQ_PINS	CELL	
FUNCTION	PIN, BUS	
INPUT	LATCH, REGISTER, PINTYPE, CELL, PIN	
INSERTION_DELAY	Library	
OUTPUT	LATCH, REGISTER, PINTYPE, CELL, PIN	
PINTYPE	PIN	Modified
PULL	BUS, PIN	New 4.1
PULL_CURRENT	BUS, PIN	New 4.1
PULL_RESISTANCE	BUS, PIN	New 4.1
Power		
CELL_SPOWER	PROPERTIES, CELL	New 4.1
GROUND_CURRENT	CELL, PATH, BUS, PIN	New 4.1
INTERNAL_ENERGY	CELL, PATH, BUS, PIN	New 4.1
PIN_SPOWER	PROPERTIES, CELL, BUS, PIN	New 4.1
SC_ENERGY	CELL, PATH, BUS, PIN	New 4.1
SUPPLY_CURRENT	CELL, PATH, BUS, PIN	New 4.1
TOTAL_ENERGY	CELL, PATH, BUS, PIN	New 4.1
Routing		
AVAILABLE_TRACK	ROUTING_PROPS, CELL	New 4.1
MIN_POROSITY	PROPERTIES	New 4.1

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
ROUTING_LAYER	Library	New 4.1
ROUTING_PROPS	CELL	New 4.1
TRACK_AREA	ROUTING_PROPS, CELL	
Scan Cell Modeling		
SCAN_EQUIVALENT	CELL	New 4.1
SCAN_PINTYPE	PIN	New 4.1
TEST_LATCH	CELL	New 4.1
TEST_REGISTER	CELL	New 4.1
Signal and Design Integrity		
CT_TOLERANCE	PROPERTIES, CELL, PIN, BUS	New 4.1
FLUENCE	CELL, PATH	New 4.1
FLUENCE_LIMIT	CELL, PIN	New 4.1
NEGATIVE	CT_TOLERANCE	New 4.1
POSITIVE	CT_TOLERANCE	New 4.1
VDROP_LIMIT	CELL, PIN	New 4.1
MOBILITY_LIMIT	CELL, BUS, PIN	New 4.1
State Table		
MAP_TO_STPIN	PIN	New 4.1
STATE_TABLE	CELL	New 4.1
Synthesis		
DONT_TOUCH	CELL	New 4.1
DONT_USE	CELL	New 4.1
Thresholds		
CSAT	PAD_PROPS, BUS, PIN	New 4.1

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
CSBT	PAD_PROPS, BUS, PIN	New 4.1
CTTAT	PAD_PROPS, BUS, PIN	New 4.1
CTTBT	PAD_PROPS, BUS, PIN	New 4.1
TABLE_INPUT_THRESHOLD	PROPERTIES, CELL	Modified 4.1
TABLE_OUTPUT_THRESHOLD	PROPERTIES, CELL	Modified 4.1
TABLE_TRANSITION_END	PROPERTIES, CELL	Modified 4.1
TABLE_TRANSITION_START	PROPERTIES, CELL	Modified 4.1
THRESHOLD_LIMIT	CELL, BUS, PIN	New 4.1
Timing Checks		
HOLD	NO_CHANGE, CELL	
MPWH	CELL	
MPWL	CELL	
NO_CHANGE	CELL	
OTHER_PINS	PATH, PATH_EXTENSION, MPWH, MPWL, PERIOD	
PERIOD	CELL	
RECOVERY	CELL	
REMOVAL	CELL	
SETUP	timing_check, NO_CHANGE, CELL	
SKEW	CELL	
Timing Checks, Conditional		
COND	PATH, PATH_EXTENSION, timing_check, CELL	
COND_END	SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE, timing_check, CELL	

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
COND_START	SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE, timing_check, CELL	
SDF_COND_END	SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE, CELL	
SDF_COND_START	SETUP, HOLD, RECOVERY, REMOVAL, SKEW, NO_CHANGE, CELL	
TLF File Structure		
CELL	Library	
CLOCK_PIN	PIN	New 4.1
DATE	HEADER	
FOR_PIN	PROPERTIES, CELL	Modified 4.1
GENERATED_BY	HEADER	
HEADER	beginning of file	
IGNORE_CELL	CELL	New 4.1
LIBRARY	HEADER	
PAD_CELL	CELL	New 4.1
PAD_PIN	BUS, PIN	New 4.1
PROPERTIES	Library	New 4.1
TECHNOLOGY	HEADER	
TLF_VERSION	HEADER	
VENDOR	HEADER	
VERSION	HEADER	
ENVIRONMENT	HEADER	
Units		

Table 4-2 TLF Properties and Keywords Grouped by Purpose, *continued*

Predefined Property Name	Context	Notes
AREA_UNIT	UNIT, PROPERTIES	New 4.1
CAP_UNIT	UNIT, PROPERTIES	New 4.1
CONDUCTANCE_UNIT	UNIT, PROPERTIES	New 4.1
CURRENT_UNIT	UNIT, PROPERTIES	New 4.1
INDUCTANCE_UNIT	UNIT, PROPERTIES	New 4.1
POWER_UNIT	UNIT, PROPERTIES	New 4.1
RES_UNIT	UNIT, PROPERTIES	New 4.1
TEMPERATURE_UNIT	UNIT, PROPERTIES	New 4.1
TIME_UNIT	UNIT, PROPERTIES	New 4.1
UNIT	PROPERTIES	New 4.1
VOLT_UNIT	UNIT, PROPERTIES	New 4.1
Wire Load		
DEFAULT_WIRELOAD_GROUP	PROPERTIES	New 4.1
DEFAULT_WIRELOAD_MODE	PROPERTIES	New 4.1
NET_CAP	PROPERTIES, WIRELOAD, WIRELOAD_BY_XXX	
NET_RES	PROPERTIES, WIRELOAD, WIRELOAD_BY_XXX	
WIRELOAD	PROPERTIES	
WIRELOAD_BY_XXX	PROPERTIES	

Glossary

A

algorithm

A set of equations used to perform calculations, in this case timing and power calculations, in a systematic fashion

C

cell

A design object that can be used any number of times to build a chip or system

CTLF

Compiled Timing Library Format

This file format is available with TLF 3.1 and prior releases. It contains the same timing data as in the ASCII TLF file, but in a binary format. A `ctlf` file is generated from a TLF file with the TLF 3.1 program `tlfc`.

Note: TLF 4.1 files can be stored in ASCII format or encrypted format.

D

DSPF

Detailed Standard Parasitics Format

This file format describes complete interconnect parasitic information in the form of RC trees, as computed by an extraction program, such as DRACULA.

E

effective capacitance

The fraction of the total interconnect capacitance to which a driving output pin reacts for a given input slew and cell I/O path model

Remaining capacitance is shielded from the driver by resistance in the interconnect.

H

hold time

For a synchronous cell, the time after the clock edge that the data input must remain stable

I

instance cell

A specific occurrence of a cell in a chip or system design

instance pin

A single-bit input or output connection of an instance cell

interconnect delay

The delay across a wire starting at a driver output pin and ending at a receiver input pin

intrinsic delay

The delay through a cell given a reference load which is connected to the output (and a reference input slew that is applied to the input)

L

load dependent delay

An extra delay factor in the source gate due to net loading

M

model

A set of data that provides parameters to an algorithm for evaluating a specific timing or power relationship

N

net

A logical signal connection between a set of pins on different instances

After routing, a net consists of routed wires on the routing layers.

P

parasitic

An unintended linear element resulting from the physical structure of a circuit

pin relationship

See *timing relationship*

polarity

An element which indicates how a signal propagation path translates an input transition (inverting, noninverting, or either)

Starting in TLF 3.0, polarity inforrmation is represented as input and output transitions in the PATH elements.

power relationship

A current or energy specification between one or more pins within a cell

primary input

An input connection accessible from outside the design

primary output

An output connection accessible from outside the design

R

receiver

Any device connected to a net at input; the input of a cell

recovery time

A timing check

This is the minimum time an asynchronous pin (usually a reset or clear pin) must be deactivated before the reference pin (usually a clock) reaches its prescribed condition.

RSPF

Reduced Standard Parasitic File

This file contains interconnect parasitics information in an approximately electrically equivalent network. This type of SPF file is used to describe approximated interconnect parasitics using a PI model to model the driver load, and Elmore delays to model the interconnect delays.

S

SDF

Standard Delay Format

This file is an ASCII format file used to express delay information and to pass the delay data between different EDA tools.

setup time

A timing check

On a synchronous cell, the time before the clock edge that the data input must be stable

skew

The difference in delays a signal suffers as it passes through two different paths

slew

The time for a signal to make a transition rising or falling

SPF

Standard Parasitic Format

This file format is used to represent extracted parasitic information describing the interconnect of a circuit.

spline

A method of defining a function that divides the domain of the function into several regions

The value of the function for each region is defined by a very simple function, such as a first or second order polynomial. The exact definition of each component function is chosen so that regions join neighboring regions smoothly.

Spline algorithm

See *Table algorithm*

spline model

See *table model*

T

table

A data format that lists the value of a function in rows and columns

Used in spline models (table models).

Table algorithm

The particular method of choosing output values from a set of input parameters and table model data

In the delay calculator, the table functions are simple linear (or, for two-dimensional functions, bilinear) polynomials chosen to make the edges between regions continuous and to exactly interpolate the values at the region corners. The separate functions are chosen to make the overall function continuous.

table model

The data used by the Table algorithm to define the input range partitions and data values at each partition endpoint (corner)

This requires a set of input values and a corresponding set of output values, either as two vectors (for a one-dimensional function) or as two vectors and a matrix (for a two-dimensional function). Interpolation order (a parameter to the Table algorithm) can also be included.

timing check

A limit between two signal transitions, for example, setup time, hold time, minimum pulse width, period, skew, and recovery time

timing path

A signal propagation path within a cell

A timing path has the attributes of polarity, a model for delay calculation, and a model for output slew calculation.

timing relationship

A delay (timing path) or timing check between one or more pins within a cell

timing view

The set of timing relationships and associated properties for a cell, or for all cells, in a library

The timing view format has changed from DFII timing views to TLF.

TLF

Timing Library Format

This is the ASCII format file used to supply timing information to various Cadence tools.

W

wavetable

A wavetable is a lookup table of current waveforms, where each waveform can have its own shape. The wavetable is the most accurate way to model current waveforms for dynamic power consumption.

Index

N

O

P

R

S

T

U

V

W