**Draft 0.3.7**

# Draft Standard for the Design Constraints Description Language (DCDL)

Sponsor

**Accellera**

Last modified: 7/25/00

# Version History

| Version | Details |
|---|---|
| 0.1.0 | Initial draft |
| 0.1.1 | Version history added, changes to “Basic Language Features”, *operating_** commands, & *derived_waveform*. Merging of *operating_variation* information into *operating_** commands. |
| 0.1.2 | Alphabetized commands and changes due to review of the introduction and basic language features. |
| 0.1.3 | Changes due to review of basic language features & addition of information for script integrators and implementors. Change of syntax for commands using {} for quoting to use “ and added subsection on strings as arguments. |
| 0.1.4 | Changes to design object and command value discussions in basic language features. Changed *name_space*, to *design_name_space*. |
| 0.1.5 | Changes due to the constraint value discussions and changes to syntax due to new keywords for defaults and reset/unset. Added placeholder section on basic terminology in the introduction section. |
| 0.1.6 | Addition of precedence rules and comment incorporation for constraint value material. |
| 0.1.7 | Addition of rule about comments and \, review comment incorporation for inheritance, unset and reset, added constraint scoping info., and cleaned upextension lang. sec. |
| 0.1.8 | Created a new annex to house rejected functionality and review comment incorporation for serveral universal commands and features (including BNF adjustments). |
| 0.1.9 | Changes to the *include* and *design_name_space* commands (& changes to BNF annex), additional script writers issues added, added section on messaging. |
| 0.2.0 | Modifications to *include* and *design_name_space* commands (& changes to BNF annex), additional script writers issues added, and information about file scope added. |
| 0.2.1 | Modifications to the design_name_space command (& changes to BNF), additional information about escapes in the strings section, and a reference to POSIX in the biblio. |
| 0.2.2 | Adjustments to reserved character section, changes to design_name_space, units, & version commands, & moved tool_domain to rejected annex. |
| 0.2.3 | Changes to the *disable* command, *current_instance* and *current_module* became *current_scope* (with other changes), and the addition of the *functional_mode* command. |
| 0.2.4 | Additional information on *current_scope*, changes to *false_path*, *clock*, *disable*, *design_name_space*, & *waveform*, and added information on pathnames. |
| 0.2.5 | Changes to *operating_range*, *current_scope*, *functional_mode*, & *design_name_space*, added material to OLA interaction section, added OLA mapping table annex, created scoping theory section. |
| 0.2.6 | Additions to OLA annex; added new command *operating_point*; adjustments to operating condition commands; changes to *multi_cycle_path*, *clock*, and *disable;* added mode semantics; moved any notes out of one-line descriptions. |

**Table 1: Version History**

| Version | Details |
|---|---|
| 0.2.7 | Additions to the scoping theory material and *units* command; minor syntax changes to most operating condition commands; started adding operating condition theory information; changed the *power_regime* command to *voltage_regime*; additions to the OLA annex |
| 0.2.8 | Additions due to selective inheritance, operating condition precedence and inheritance, new timing domain theory outline, change to *target_based_uncertainty.* |
| 0.2.9 | Changed the *instance_*,*pin_*, *port_*, and *cell_identifier* constructs to *_list*, changed *-cell* and *-instance* to *-cells* and *-instances* (as appropriate); changed the syntax of *disable*, *clock_arrival_time*, *waveform*, and *driver_specification*; moved tags discussion to the rejected annex for now. |
| 0.3.0 | Modified *clock_arrival_time*, *clock_delay*, *clock_mode*, *disable*, *multi_cycle_path*; *driver_specification* was changed to *driver_cell* & the syntax was altered; *external_load* was changed to *port_capacitance* & the syntax was altered. |
| 0.3.1 | Change from OVI to Accellera; changes to *driver_cell*, *clock_delay*, *clock_mode*, *port_capacitance* commands; *external_load_limit* renamed to *port_capacitance_limit*. |
| 0.3.2 | Added status convention for commands; changes to *port_capacitance*, *waveform*, *derived_waveform*, *clock_mode*, & *clock_required_time*. The “Constraint Inheritance” section was changed to a more general concept of selected inheritance. |
| 0.3.3 | Changes to the *waveform*, *derived_waveform*, *clock_arrival_time*, *functional_mode* & *clock_required_time*; added introduction material on terminology - constraint vs assertion vs annotation vs directive. |
| 0.3.4 | Deleted the *derived_clock* and added this abilility to the *clock* command; changes to *clock*, *clock_uncertainty*, *derived_waveform*, *clock_arrival_time*, & *clock_required_time* |
| 0.3.5 | Changes to *slew_time*, *clock*, *clock_delay* & *clock_skew*; *clock_slew* was eliminated - the functionality added to *slew_time*. The syntax of the following commands was altered due to adding the object type to keywords: *clock_mode*, *disable*, *false_path*, *multi_cycle_path*, *common_insertion_delay* & *tree_delay.* |
| 0.3.6 | Added default values to *waveform* and *derived_waveform* options; changes to *false_path*, *multi_cycle_path*, *waveform*, *target_uncertainty*, *common_insertion_delay*; eliminated *clock_domain* command - functionality covered in *waveform*; changed *target_based_uncertainty* to *target_uncertainty.* |
| 0.3.7 | Changed *derived_waveform*, *target_uncertainty*, *data_arrival_time*, *false_path*, *multi_cycle_path*; moved *tree_delay* and *tree_mode* to the timing exception section. |

**Table 1: Version History**

All changes between 2 consecutive versions are indicated with change bars in the left margin (with the exception of global changes, such as header and footer dates and versions). All specification source documents are archived for comparison purposes.

# Acronyms and Abbreviations

This section lists the acronyms and abbreviations used in this standard.

| | |
|---|---|
| API | Application Programming Interface (see also PI) |
| ASIC | Application Specific Integrated Circuit |
| BNF | Backus-Naur Form |
| CAE | Computer-Aided Engineering (the term EDA is preferred) |
| DCDL | Design Constraints Description Language |
| DCL | Delay Calculation Language |
| DC-WG | Design Constraints-Working Group |
| DEF | Design Exchange Format |
| EDA | Electronic Design Automation |
| FPGA | Field Programmable Gate Array |
| HDL | Hardware Description Language |
| IC | Integrated Circuit |
| IP | Intellectual Property |
| LEF | Library Exchange Format |
| MCM | Multi Chip Module |
| OLA | Open Library API |
| OVI | Open Verilog International |
| PCB | Printed Circuit Board |
| PDEF | Physical Design Exchange Format |
| PI | Procedural Interface |
| PVT | Process/Voltage/Temperature |
| RC | Resistance multiplied by Capacitance |
| SDF | Standard Delay Format |
| SLDL | System Level Design Language |
| SPEF | Standard Parasitic Exchange Format |
| SPF | Standard Parasitic Format |
| SPICE | Simulation Program with Integrated Circuit Emphasis |
| Tcl | Tool command language |
| VHDL | VHSIC Hardware Description Language |
| VHSIC | Very High Speed Integrated Circuit |
| VI | VHDL International |
| VITAL | VHDL Initiative Towards ASIC Libraries |
| VLSI | Very Large Scale Integration |
| VSIA | Virtual Socket Interface Alliance |

SECTION PAGE

# List of Figures

# List of Tables

# 1. Introduction

[Informative]

The objective of the Design Constraints Description Language (DCDL) is to make it possible for designers to *consistently* specify, apply, and reuse constraint descriptions in order to describe design intent for EDA tools. This objective is accomplished by specifying a command syntax, appropriate semantic descriptions, and related conceptual material.

## Background

Libraries and design descriptions are currently addressed using IEEE, emerging, and de-facto standards. Figure 1-1 shows a simple example of the standards world in the context of a high-level design to physical implementation design flow.

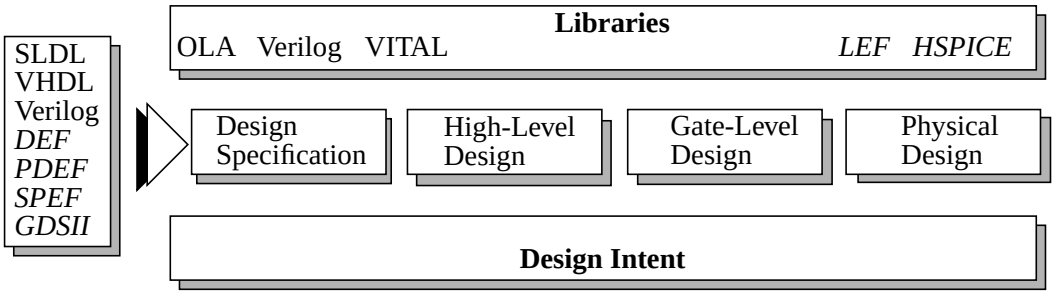



**Figure 1-1 Design Standards**

However, a standard method for specifying and reusing design intent is considered by many to be one of the last, major missing pieces required to support an interoperable design flow.

In March 1998, Open Verilog International (OVI) chartered the Design Constraints Working Group (DC-WG) to specify a standard means to express design intent and to work with other standards groups to promote its use within the design community. The key deliverable of the DC-WG is this specification.

## Scope

DCDL is intended for use in any EDA tool flow and it covers all major constraint domains, as Figure 1-2 shows.

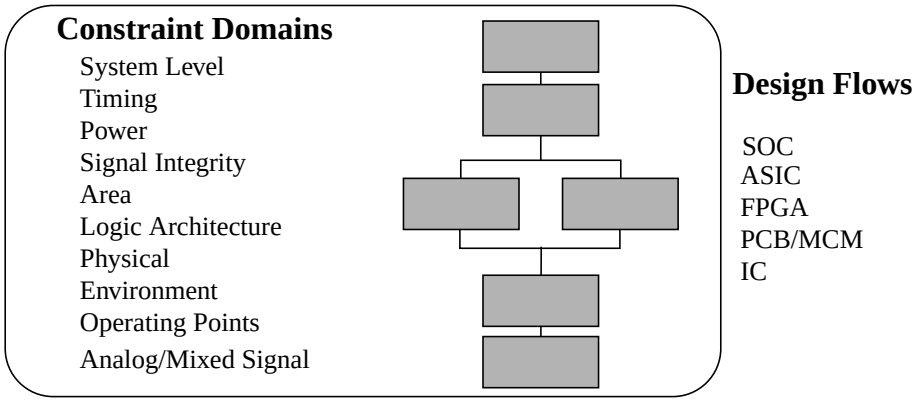



**Figure 1-2 DCDL Scope**

This document serves “users” of DCDL, such as designers and intellectual property (IP) providers and integrators; and “implementors” such as EDA vendors and silicon vendors.

## Usage Models

The usage models for DCDL include the following:

a) *Initial constraint entry*. Initial design intent is specified by using pure DCDL or by embedding DCDL in tool scripts.

b) *Constraint interchange*. A very important facet of interoperability is the ability to pass DCDL constraint files between tools after initial constraint entry. This usage model requires constraint-based tools that read and write DCDL.

c) *IP authoring*. Soft and firm IP products require a means to express intent and to guarantee that the block will operate under a range of constraints. This usage model can be similar to initial constraint entry or the author could export DCDL from a trusted design tool (constraint interchange).

## Standards Organization

The DC-WG is sponsored by Accellera (formerly OVI and VI) and endorsed by the Virtual Socket Interface Alliance (VSIA).

The DC-WG is responsible for specifying DCDL, working with related standards groups, moving the specification through the standards process, analyzing and resolving issues, and teaming with Accellera to promote the use of the standard (via vehicles such as conferences, press releases, meetings, etc.).

The DC-WG defines one constraint domain at a time, by collecting input from members, creating a taxonomy (generic functional specification) and then mapping the taxonomy into the syntax and semantics described in this specification. The specification is then voted on and the next domain is addressed. Figure 1-3 shows the DC-WG creation process.

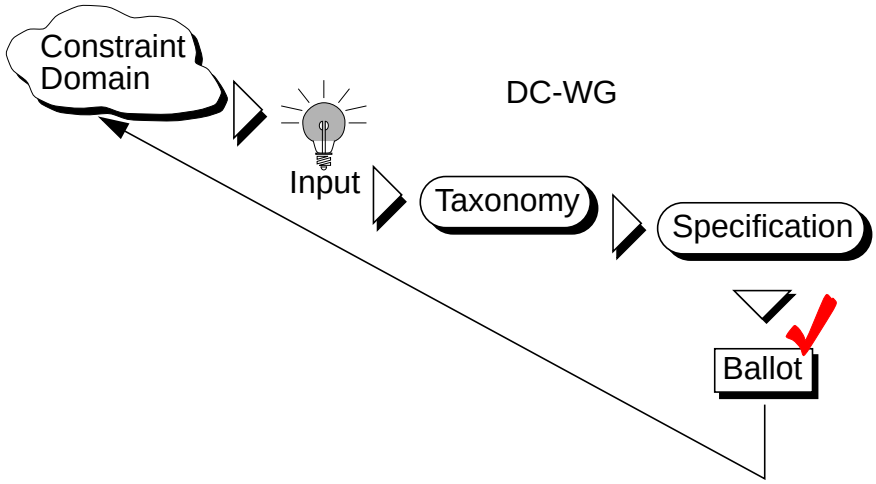



**Figure 1-3 DC-WG Creation Process**

The DC-WG consists of volunteers representing designers, silicon vendors, standard groups, industry consortiums, and EDA vendors. Details about DC-WG activities can be found at:

http://www.eda.org/dcwg

## Standards Interaction

DCDL makes a clear distinction between library-related standards and design-related standards. DCDL is a design standard. However, in order for DCDL to co-exist in any design flow and to ensure that duplicate work is not being performed, the DC-WG interacts with related standards groups. Based on where the DC-WG is in the creation process and which constraint domain is being addressed, this interaction is always changing. The following subsections provide an overview of such standards interactions at the time of the writing of version 0.3.7 of this specification.

### SLDL Interaction

A joint working group between DC-WG and the System Level Design Language (SLDL) group ensures that DCDL is syntactically compatible with SLDL. The SLDL effort is sponsored by Accellera.

*** Add other appropriate details here. ***

### OLA and the Delay and Power Calculation WG Interaction

When appropriate, DCDL commands strive for compatibility with the Open Library API (OLA) standard effort and the related Delay and Power Calculation standard IEEE 1481. Appropriate commands are those that allow the designer to choose from options defined within the library. For example, choosing from a set of defined operating conditions or modes.

An informative annex contains a table showing DCDL commands and the related OLA API functions (refer to page 177).

### Basic Terminology

Thoughout the industry, there are many terms that are associated with design intent. For the purposes of this specification, the following terms are defined:

> **Constraint**: a desired characteristic that a tool must enforce or satisfy.
> **Assertion**: a statement of truth about a design object that a tool accepts as fact during analysis.
> **Directive**: a statement that directs a tool to implement a specific characteristic.
> **Annotation**: characterized data attached to a design object that is based on the design implementation.

In terms of DCDL, this specification documents commands that can be used as constraints, assertions, and directives. Commands supporting annotations are not part of DCDL.

### Participants

The Design Constraints Description Language Working Group contributors include:

**Mark Hahn,** *Chair*

Tim Baldwin
David Barton
Paul Bonnel
Simon Butler
Joe Daniels
Dan Devries
Tom Dewey
Bob Dilly
Jim Engel
Ibna Faruque
Vassilios Gerousis
Steve Grout
Hemlata Gupta
Jeff Handong
Kareem Lafala
Enrico Malavasi
Ed Martinage
Dan Moritz
John Paul
Gregory Schulte
Vikas Sharma
Jin-sheng Shyr
Alex Suess
Jim Swift
Andres Teene

## Corporate Participation*r*

Ambit
Cadence
Exemplar
Fujitsu
IBM
Infineon
Intermetrics
LSI Logic
Lucent
Mentor Graphics
Motorola
Sematech
Symbios
Synopsys
Toshiba
Xilinx

# 2. Language Documentation Conventions

This section describes the language conventions used to document DCDL and provides an overview of the document contents.

## 2.1 Language Conventions

The formal syntax of DCDL is described using Backus-Naur Form (BNF). The textual cues (such as fonts and symbols) used to document are:

a) Lowercase words, some containing embedded underscores, are used to denote syntactic constructs (terminals and non-terminals). For example:

    port_list

b) Boldface words are used to denote reserved keywords, operators, and punctuation marks as a required part of the syntax. For example:

    **-ports {** in1 in2 in3 **}**

    indicates that *-ports* and the braces are required.

c) The **::=** operator separates the two parts of a BNF syntax definition. The syntax category appears to the left of this operator and the syntax description appears to the right of the operator. For example:

    text ::= character_set

d) A vertical bar separates alternative items (use one only). For example:

    integer_digit ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

e) Square brackets enclose optional elements, unless the brackets are part of the syntax (in which case they would appear in bold). For example:

    exponent ::= **E** | **e** [ sign ]

    indicates *sign* is an optional syntax item.

f) Braces enclose a repeated item, unless the braces are part of the syntax (in which case they would appear in bold). The item may appear zero or more times and the repetitions occur from left to right. For example:

    unsigned_number ::= integer_digit { integer_digit }

g) Parentheses enclose items within a group, unless the parentheses are part of the syntax (in which case they would appear in bold). For example:

    **-waveform** *waveform*_identifier | ( **-root** port_identifier | pin_identifier )

    the parentheses group the *-root* keyword and its two keyword value choices.

h) A hyphen (-) is used to denote a range. For example:

    identifier_first_letter ::= **a**-**z** | **A**-**Z**

    indicates the first letter of the identifier can be a lowercase letter (from a to z) or an uppercase letter (from A to Z).

i) If the name of any element starts with an italicized part, it is equivalent to the construct name without the italicized part. The italicized part is intended to convey some semantic information. For example:

-**keywords** *keyword*_identifier

indicates that the *keyword*_identifier is equivalent to the identifier construct.

The text of this document uses *italicized* font when a literal identifier is being used (typed exactly as it is presented) or to reference a construct name.

The `monospace` font is used for examples.

## 2.2 Status

[To be removed at ballot]

This specification is a “living” document that changes based on input through the DC-WG. The status of each command is indicated by a symbol following the command name on the reference pages:

● represents a reviewed and approved command. This command description is stable.

◗ represents a command that is currently being reviewed and further refined. This command description could change.

❍ represents the initial draft description of a command. This command description is very likely to change.

## 2.3 Specification Organization

Beyond the introduction section and this conventions section, this standard is organized into the following sections:

a) *Basic language features*: the building blocks and general rules of the language.
b) *Universal commands and features*: commands and features that can apply to all constraint domains.
c) *Scoping commands*: commands that indicate where in a design the constraint commands should be applied.
d) *Operating conditions*: commands that specify design operating conditions.
e) *Timing domain*: commands that specify timing constraints - clocking, timing boundaries, and timing exceptions.
f) *Parasitic domain*: commands that specify parasitic boundary conditions.
g) *Standard compliance*: what it means for a tool or DCDL file to be in compliance with this standard.
h) *Glossary*: definitions of some of the key terms used in this specification.
i) *Annex A - BNF*: the complete syntax of DCDL using Backus-Naur Form (BNF).
j) *Annex B - Bibliography*: a listing of documents referenced during the creation of DCDL.
k) *Annex C - Analyzed and rejected functionality*: information about features discussed but rejected for this version of the specification.
l) *Annex D - DCDL relationships to OLA*: a mapping of DCDL features to the appropriate Open Library API standard.
m) *Index*: page references for key topics.

# 3. Basic Language Features

This section provides a fundamental overview of the DCDL command structure, lexical elements, and general language features.

## 3.1 Command Structure

DCDL consists of a set of commands. Commands consist of one or more words. Words are formed from a character set. The first word is the command name and additional words are arguments to the command. Arguments consist of required and/or optional keywords (paired with keyword values if required) and zero or one positional parameter value. Keywords are indicated by a leading - (dash) and the dash is part of the keyword. A positional parameter is a value or identifier that is not associated with a particular keyword. Figure 3-1 shows the structure of a typical DCDL command:

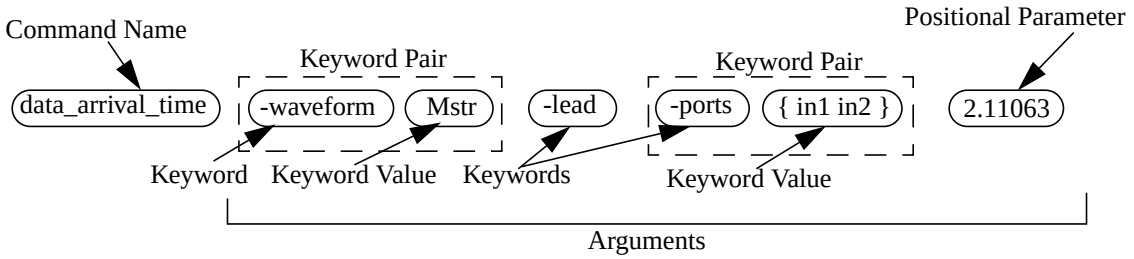



**Figure 3-1 Command Structure**

## 3.2 Lexical Elements and Rules

Lexical elements define the building blocks of DCDL. The language is built using the lexical elements and the rules that govern them.

### 3.2.1 Character Set

DCDL command words are formed using characters. Characters are comprised of three types: whitespace, reserved, and non-reserved characters. Whitespace characters include the space and tab characters. Reserved characters are those used as punctuation in the command syntax. Non-reserved characters represent all the other characters. For more information about the character types, refer to page 167.

### 3.2.2 Case Sensitivity

DCDL command names and keywords are case-sensitive. Case for keyword values and positional parameters are governed by the *design_name_space* command (refer to page 43).

### 3.2.3 Whitespace

Commands begin with the first non-whitespace character in a line. Command words are separated by one or more spaces or tabs. These spaces and tabs are not considered part of the command words.

Items in lists must be separated by one or more spaces or tabs (refer to page 29 for information about lists).

*** We need to say something about whitespace (or lack thereof) between punctuation: ; , { }, \, [], etc. For example, {a b} or { a b } or addr[1] or addr [ 1 ]. ***

### 3.2.4 Command Termination

DCDL commands terminate with a newline or optionally with a semicolon. Use of the semicolon allows placing multiple commands on a single line and it allows the use of comments to the right of a terminated command.

### 3.2.5 Line Continuation

For readability, commands can be split between lines in a file by using the continuation character \ (backslash). For example, the following command is split into 6 lines:

```
data_arrival_time \
   -waveform master_clk -lead -ports \
   { MemData[0] MemData[1] MemData[2] MemData[3] \
   MemData[4] MemData[5] MemData[6] MemData[7] \
   MemData[8] } \
   { 6.0 7.0 6.5 7.5}
```

### 3.2.6 Comments

Comments are indicated by the # (hash) character. All the characters between the # and the newline are considered comments, and shall be ignored. Comments can appear at the termination of a command if the optional semicolon is used. Comment examples:

```
#Comment on a line by itself
clock_mode -ideal; #Set the mode to ideal
clock_mode -ideal; tree_mode -ideal; #Set both modes
#clock_mode -actual
```

Continuation characters are honored in comments. For example:

```
clock_mode -ideal; #Set the clock mode for \
project pilot
```

In the preceding example, the comment extends to the next line to include the text *project pilot*.

### 3.2.7 Reserved Words and Characters

DCDL does not define any reserved words. Command names must appear as the first item in a line and keywords are denoted by words that begin with the - (dash) character, so reserved words are not necessary.

The following characters are reserved:

The # (hash) character is reserved to indicate a comment (refer to page 28).

The semicolon is reserved as an optional method to terminate commands (refer to page 28).

The { } (brace) characters are reserved to indicate lists (refer to page 29).

The “ (double quotes) are reserved to indicate strings or single characters (refer to page 29).

The ? (question mark) and * (asterisk) characters are reserved for wildcarding and for placeholding constraint value slots (refer to page 32).

The \ (backslash) is reserved to indicate line continuation - escaping the newline - (refer to page 28) and escaping characters in strings (refer to page 29).

The particular name space employed in the design flow using DCDL can impose reserved characters and words. Refer to the *design_name_space* command for details (page 43).

## 3.3 General Language Features

This section defines the language features that can apply to each DCDL command.

### 3.3.1 Command Shorthand

DCDL does not provide a convention for specifying command or command argument abbreviation. However, DCDL readers or interactive constraint entry tools can provide support for DCDL shorthand. DCDL writers shall write DCDL as specified in this standard (no shorthand).

### 3.3.2 Identifiers

DCDL commands often require identifying an item by name. Identifiers can specify design objects (refer to page 31) or arbitrary names provided by the user. Identifier rules are defined by the *design_name_space* command (refer to page 43). Refer to page 167 for the syntax for identifiers.

### 3.3.3 Lists as Arguments

Some DCDL keyword values and/or positional parameters allow lists as values. Lists are represented by enclosing the values in { } (left and right braces). The values within the list are separated by one or more spaces or tabs. For example:

```
data_arrival_time \
   -waveform master_clk -lead -ports \
   { MemData[0] MemData[1] MemData[2] MemData[3] \
   MemData[4] MemData[5] MemData[6] MemData[7] \
   MemData[8] } \
   { 6.0 7.0 6.5 7.5}
```

The preceding example shows a list of *-ports* keyword values and a list of data arrival times.

The basic rules for lists are that they are only one level (embedded lists do not exist in DCDL commands) and that any keyword value or positional parameter that expects a list, must use the {} characters. The {} characters must be used, even if a list contains only one element. For example:

```
-ports {pina}
```

### 3.3.4 Strings as Arguments

Several keyword values or positional parameters allow text strings or single characters as values. Text strings and single characters must be enclosed within double quotes. For example:

```
history “File created on 1/10/2000 by dcdl_writer”
version “1.0”
extend_dcdl my_command -arguments “-ports {in1 in2} 0.23”
```

DCDL supports an escaping mechanism for strings by using the \ character and defines a set of escapes for characters that are not typically visible in ASCII text.

In general, any character used in a string that can be interpreted as part of the syntax of the DCDL command, must be escaped. For example, to retain the double quotes within the following string, they must be preceded by the escape character:

```
history “File created by \”dcdl_writer\””
```

DCDL supports the following pre-defined escapes also:

\n to indicate a newline

\r to indicate a carriage return

\t to indicate a tab

\v to indicate a vertical tab

### 3.3.5 Design References

Many DCDL commands reference design objects through keywords, keyword values, and positional parameters. This section describes referencing design objects.

#### 3.3.5.1 Object Types

The DCDL commands that are associated with design objects, require that the type of these objects be identified. There exists logical, constraint, and physical object types.

##### 3.3.5.1.1 Logical Design Object Types

Logical design objects are identified by the use keywords that specify the following:

*pin*: representing terminal point(s) where an interconnect structure makes electrical contact with the fixed structures of a cell instance; or the conceptual point(s) where a net connects to a lower level in the design hierarchy.

*port*: representing conceptual point(s) at which a cell or a hierarchical design unit makes its interface available to higher levels in the design hierarchy.

*net*: representing an electrical connection between pins or ports in a design.

*cell*: representing functional design unit(s) (typically found in a library).

*instance*: or -*instances* representing reference(s) to a cell(s) within a design.

*library*: representing a collection of circuit functions (typically cells), implemented in a particular technology, that are used to create a design.

Some EDA tools do not differentiate pins from ports and may combine those two types into one.

*** We probably need to add direction options for ports and pins (in, out, inout). This will make wildcarding stronger and make embedded scripts useful. ***

##### 3.3.5.1.2 Constraint Object Types

Constraint objects are identified by the use of the following keyword within DCDL commands:

-*waveform*: representing an ideal waveform that is used as a reference point for other commands.

Constraint objects are elements that do not appear in the design.

##### 3.3.5.1.3 Physical Design Object Types

No physical object types are specified at this time. However, it is predicted that these object types will be necessary for future constraint domains. For example, providing the distinction between physical and logical ports might be necessary.

#### 3.3.5.2 Design Name Spaces

DCDL is designed to operate within any design flow. However, design object names usually must adhere to the name space rules for particular design flows. For example, a Verilog design flow requires that design objects follow the Verilog name space rules. Particular tools within a design flow might also restrict the design name space.

DCDL accommodates name spaces by providing the *design_name_space* command (refer to page 43). This command provides builtin support for VHDL and Verilog name spaces. In addition, the *design_name_space* command provides the means to specify custom name space rules.

#### 3.3.5.3 Design Object Identifiers

The design object identifiers used as keyword values can take several forms:

**Lists**: one or more design objects surrounded by braces (refer to page 29). For example:

```
-ports {reset set}
```

**Wildcards**: shorthand, character matching technique (refer to the following sections). For example:

```
-pins {out* bus1[*]}
```

**Pathnames**: fully qualified or relative pathnames. For example:

```
-pins {“/top/u1/*” “u12/in1”}
```

Fully qualified pathnames must use a leading hierarchy separator as specified by the *design_name_space* command (or the default if this command is not used). In the preceding example, the leading hierarchy character is /.

Relative pathnames are specified with respect to the current scope of the design as specified using the *current_scope* command (refer to page 63). In the preceding example, *u12* must be in the current scope. If referencing a design object in a higher-level scope, a fully qualified pathname must be specified.

All pathnames must be surrounded by double quotes.

#### 3.3.5.4 Bit Representation

The *design_name_space* command can be used to change bit and bit range characters. By default the [ ] (left and right square brackets) are used to indicate individual bits of bundled pin, ports, or nets. For example:

```
MemData[8]
```

By default, a colon is used to indicate a bit range. For example:

```
MemData[5:31]
```

If a bus name is provided without a bit or bit range indication, the command applies to the entire bus. For example, the following command specifies an external load value for all the bits of the bus *MemData*:

```
pin_capacitance -ports {MemData} 0.01
```

#### 3.3.5.5 Wildcards

In order to provide shorthand access to multiple logical design objects, DCDL provides limited wildcarding using the * (asterisk) and ? (question mark) characters. Wildcarding can only be used to reference design objects and only in the following manner:

**Character sequence matching**: wildcarding zero or more characters. For example:

```
pin_capacitance -ports {in*} 0.01
pin_capacitance -ports {*_out*} 0.02
pin_capacitance -ports {*} 0.002
```

In the preceding examples, the *port_capacitance* command is applied in three ways: all ports whose name starts with *in*, all ports whose name contains *_out*, and all ports (represented by the wildcard alone).

If the * wildcard characters match a base bus name, all the bits of the bus are matched. If the bus is named *bus_1* then *bus_1** or *bus_1[*]* match all bits of the bus - *bus_1[7:*]* is not allowed.

**Next character matching**. Wildcarding exactly one character. For example:

```
pin_capacitance in? 0.110
```

In the preceding example, *in?* would match *in* and one other character (for example *in1*, *in2*, *in3*). But, there would be no match for *in* itself or *in* and 2 or more characters (for example *in10, in11, in12*).

Many DCDL commands only operate on certain design objects or objects at certain design levels. For these commands, wildcarding only matches the appropriate design objects and either silently ignores any other matches or issues an appropriate message. For example, the *data_required_time* command only applies to output or bi-directional ports or pins. Therefore, any input ports or pins that match a wildcard would be ignored. The *driver_cell* command can only be applied to top-level ports, so any attempt to set this constraint at a lower level port causes an error.

Wildcarding does not match through hierarchy. The hierarchy delimiter stops the matching process.

When a wildcard is encountered it is expanded at that point and applied to the constraints at that time.

*** Might need to allow bus bit matching for IP parameterization. For example addr[5:*] ***

### 3.3.6 Command Ordering

Commands can appear in any order in an ASCII file, any number of times. However, DCDL uses a “describe before using” model - meaning that any command that depends on or is affected by another command implies an order of appearance. For example, in order to use a waveform description in a command, that waveform must be defined first.

The ordering of keywords, keyword values, and positional parameters is not pre-defined. While keywords, keyword values, and positional parameters are documented in a certain order in this specification, their positions are interchangeable. However, the command name must always appear first and keywords and associated keyword values must be paired. For example, it is not legal to place a positional parameter after a keyword that expects a keyword value.

Precedence rules can impose command ordering effects (refer to page 37).

#### 3.3.6.1 Command Ordering Examples

The following examples show correct and incorrect ordering of command words.

```
constant 0 -ports {grnd};#Correct. Positional argument first.
constant -ports {grnd} 0;#Correct. Positional argument last.
constant -ports 0;#Incorrect. No keyword value for -ports
clock -waveform -rise mstr -ports {in1};#Incorrect. -waveform mstr
   #is incorrectly placed after -ports
```

### 3.3.7 Constraint Values

DCDL commands specify constraint values using keyword values and/or positional parameters. Constraint values can be comprised of characters, strings, positive/negative real and integer numbers, and identifiers - or lists of those elements. These values are either explicitly expressed in a command, left undefined, assigned default values, or are implicitly assigned via precedence rules. The precedence rules govern how commands interact.

Constraint values are explicitly assigned by the use of commands. Some of these commands allow unsetting, or resetting existing constraint values, merging multiple constraint values, and the specification of one or multiple values for the constraint (value slots).

The following subsections discuss each of the preceding aspects of constraint values.

#### 3.3.7.1 Undefined Constraint Values

All constraints are undefined until a value is provided. After an application reads in all the specified DCDL commands, some constraints can still be left undefined. Particular DCDL commands assign a value to constraints that are left undefined. These assignments are documented within the appropriate command description.

### 3.3.7.2 Default Constraint Values

Default constraint values are assigned by the following methods:

**Keyword defaults**. Many DCDL commands apply keyword and/or keyword value defaults for optional arguments that are not specified. These assignments are documented within the appropriate command description. The defaults cannot be changed by the designer, except by explicitly specifying the desired keyword and keyword value pairs.

For example, in the *data_arrival_time* command there exists several optional keywords: *-early*, *-late*, *-rise*, *-fall*, *-lead*, and *-trail*. Without these options, the command could look like the following:

```
data_arrival_time -waveform sys_clk -ports {in1 in2} 5.0
```

Without the options, DCDL assumes both early and late analysis and both rise and fall transitions. The leading edge is also assumed (*-lead*). If this was not the desired effect, the designer must specify the keywords to change this behavior.

**Positional parameter defaults**. The designer can set default values for positional parameters in DCDL commands that contain the optional *-default* keyword. These default values apply to each matching command in the DCDL input until an explicit value is applied.

A default can only be assigned to a particular command - not a specific design object. Therefore, if *-default* is used, no design objects can be referenced in that command. For commands that allow specification of multiple design object types, the *-default* option allows the specification of one object type. This means that there must be separate commands for each object type.

For example, the following command allows default specification of a constraint value on both pins and ports. In order to specify default values for both pins and ports, two commands are required:

```
slew_limit -pins 1.34
slew_limit -ports 1.68
```

For a discussion about design object types, refer to page 31.

Explicit defaults are persistent - they are applied unless unset or explicitly changed. This allows defaults to be preserved throughout a design flow.

**Meta defaults**. DCDL does define a few commands that assume default values if none are specified - meta defaults. These types of commands are equivalent to setting the default for the command using *-default*, without explicitly specifying the command. These commands require default values to perform analysis correctly and are clearly noted in the appropriate command reference documentation.

*** Default causes some syntax havoc, because now the required *-ports* and/or *-pins* are no longer required. Also, the default value could be a list. Error? ***

### 3.3.7.3 Unsetting and Resetting Constraint Values

Unsetting and resetting actions are necessary to override any explicit defaults. Unsetting and resetting actions are not declarative (like the rest of DCDL) - but are included in this standard out of practicality. It is recognized that most tools supporting DCDL will need to unset and reset values and thus a standard mechanism for accomplishing that task is desirable.

A subset of DCDL commands allow their values to be unset or reset. This is accomplished by the use of optional *-unset* and *-reset* keywords. Unsetting explicitly returns a particular command value to undefined. Resetting explicitly returns a particular command value to its default values (if it had any). A command can either be unset or reset, but not both.

For example, this command unsets its value:

```
slew_limit -ports {inA inB} -unset
```

The preceding command specifies that the slew limit on ports *inA* and *inB* are undefined.

This command resets its value:

```
slew_limit -pins {int1 int2} -reset
```

The preceding command specifies that the slew limit on pins int1 and int2 are *** infinite? *** and that the optional keyword values (not specified in this example) should be applied.

*** These options cause syntax trouble, because the required positional parameter (always required) is no longer required and potentially required keywords like *-port* or *-pin* are not required either. So, you could have something like *slew_limit -reset* ? In this case, we need to object type argument, like *-default* ?***

*** The DC-WG is evaluating whether or not to include unset and reset. Therefore, the command reference pages do reflect this ability at this time. ***

### 3.3.7.4 Constraint Value Slots

Several DCDL commands contain value slots. Value slots allow the specification of multiple values for a constraint. For example, the *data_arrival_time* has four value slots:

the early rise time
the late rise time
the early fall time
the late fall time

Another example, the *multi_cycle_path* command has two value slots:

the early cycle offset value
the late cycle offset value

Commands that utilize value slots allow the specification of individual slot values or the specification of one value that applies to all the slots.

#### 3.3.7.4.1 Value Slot Placeholders

Commands that utilize value slots can hold the place of one or more of the existing values by using a placeholder represented by * (an asterisk). For example, commands that allow the specification of *-early*, *-late*, *-rise*, and *-fall* can specify placeholders for 1 to 4 of the values:

```
data_arrival_time \
   -waveform master_clk -lead -ports \
   restart { 6.0 7.0 * *}
```

In the preceding example, the early rise and late rise values are specified as 6.0 and 7.0. The early fall and late fall values are represented with placeholders, meaning that these values should use the previous values assigned (if any).

Placeholders are only allowed in value slots. For example, a placeholder cannot be used for a command that expects a single value:

```
borrow_limit *; #Incorrect
```

### 3.3.8 Precedence Rules

The precedence rules provide information about the relationship between explicitly defined constraints, defaults, resetting and unsetting constraints, and value slot placeholders.

The general DCDL precedence rules are the following:

1) **Matching**[1] **explicit commands**: the last command read overrides the preceding command.
2) **Matching default commands**: the last default command read overrides the preceding command.
3) **Reset matches explicit command**: will eliminate the effect of the explicit command but not the effect of any matching default command. This action occurs when the reset command is read, so any matching command after the reset is not affected.
4) **Unset matches explicit command**: will eliminate the effect of the explicit *and* default command. This action occurs when the unset command is read, so any matching command after the unset is not affected. However, any matching default command that is read after this unset will *not* take affect.
5) **Placeholders**: are not affected by the precedence rules, as they indicated that nothing is specified for a particular value slot.
6) **Value slots**: the preceding precedence rules apply independently to each value slot.
7) **Meta defaults**: for the few commands for which DCDL implies values, unset commands that match explicit commands return the commands to the DCDL default value instead of setting the value to undefined. Refer to page 35 for more information about meta defaults.

The preceding rules are considered universal to DCDL. In addition to these general types of rules, there are constraint-domain-specific precedence rules. The domain-specific precedence rules arbitrate between similar constraints that have the same purpose (such as driver cell type, driver strength, and input slew), or between constraints that overlap in their effect (such as a default slew for all internal pins and a default slew specific to the leaf pins in a particular clock tree). These concepts are discussed in the domain theory portions of this document.

[1] Matching means that the command name, keyword values, and design objects (if any) are exactly the same

#### 3.3.8.1 Precedence Rule Examples

*** An example of how each rule works will be included here. ***

### 3.3.9 Constraint Scoping

DCDL commands apply to default or explicit scopes within the design. Some commands assume a default scope, such as the top level of the design. These defaults are discussed in the command reference page. The scope can be explicitly set using the *current_scope* command (refer to page 63).

There exists a small set of commands whose values remain in affect (and could therefore affect other commands) for the entire DCDL file (and potentially any included file) until another matching command is found. These types of command are called file scope commands. If a command is a file scope command, it will be indicated as such in the command reference page.

For a general overview of scoping theory, refer to page 61.

### 3.3.10 Constraint Inheritance

DCDL does not provide *general* language rules for inheritance. Therefore, constraints set at one level of a hierarchy do not automatically apply to lower levels of the design. There are exceptions to this however:

**Specific commands**. Commands that apply to specific cells or instances within a design often can impose inherited constraint values on lower-levels of the cell or instance hierarchy (unless a lower level command applies a more specific value for the constraint). Several operating condition commands fall into this category. Also, some commands allow inheritance because of efficiency reasons. Commands that allow inheritance are discussed in their particular domain theory and specific command page sections.

The following commands allow inheritance:

clock_mode
operating_point
operating_process
operating_range
operating_temperature
operating_voltage
temperature_regime
voltage_regime

**Emulated inheritance**. DCDL does provide a set of commands that do allow emulation of inheritance if that behavior is desired: *include*, *current_scope* (refer to sections *Universal Commands and Features* and *Scoping Commands* for information about these commands) and wildcarding (refer to page 32).

In addition, the preceding DCDL commands can be coupled with tool-specific commands (such as those that return design objects) and extension languages to further automate constraint inheritance (refer to page 40).

### 3.3.11 Command Name Collisions

Because DCDL commands were created based on a common taxonomy of present and future constraint domains, the potential exists for DCDL command names colliding with proprietary command names in EDA tools.

It is the responsibility of the EDA tool vendor that supports DCDL to handle naming collisions between the language and the tool command language.

### 3.3.12 Message Handling

DCDL defines several syntactic and semantic conditions that are to be noted as an error or warning. It is the responsibility of the EDA tool vendor to define levels of messaging severity and the behavior of the tool when these messages assert.

*** The golden parser can be used to check syntax of a file. ***

## 3.4 DCDL and Extension Languages

[Informative]

*** A high-level discussion of pure, mixed, and interspersed (ie variables in DCDL) models and a brief overview of parsing & interpreter concepts will be added here. ***

### 3.4.1 Tcl Interoperability for Script Writers

If DCDL is to be embedded into a Tcl environment, there are several potential interoperability issues:

**Comments**. Tcl parses some comments and can return errors if comment characters are located in places that Tcl does not consider as the starting character of a command. For example, this code causes an error in Tcl:

```
if { 0 == 0 } {
# if { 0 == 1 } {
puts mistake
}
```

DCDL follows more straight-forward rules (refer to page 28) and the preceding example would not create an error.

**Wildcarding**. DCDL greatly limits the definition of wildcards in order to provide a useful balance between simplicity and efficiency (refer to page 32). Tcl has a much broader wildcarding ability. Therefore, scripts should limit wildcarding to the DCDL subset.

**Include versus source**. DCDL defines an *include* command that is similar to the Tcl *source* command with a few exceptions. If the *-inline* option is used, the behavior of *include* matches *source*. Refer to page 55 for details.

**Escapes**. For the *design_name_space* command, DCDL provides an escape character and the escaped tab and newline characters (*t* and *n*). Refer to page 43 for details.

**Spaces in pathnames**. DCDL double quotes specifications of pathnames in commands such as *include*. When such commands appear within Tcl scripts, the double quotes might not be present.

### 3.4.2 Tcl Interoperability for Application Developers

Application developers that work within a Tcl environment, need to understand interactions with DCDL. These interactions include accounting for:

**Command namespace**. Applications need to account for the DCDL command names in order to prevent naming collisions. For example, Tcl 8.0 (and later versions) provides a *namespace* construct to prevent collisions.

**Lists**. DCDL strictly defines lists as arguments (refer to page 29) by using braces {}, even when a list contains only one item. A Tcl list is a major element of that language. Therefore the concepts and usage of lists are much more complex in Tcl versus DCDL.

# 4. Universal Commands and Features

This section describes commands and features that can be applied within all the constraint domains.

It is a good practice to specify a set of universal commands for application within a DCDL file and for individual blocks of a design. These commands are typically placed at the top of a DCDL file, for convenient access.

The universal commands follow the general rule within DCDL that states that for any command that depends on or is affected by another command - that other command must be specified first (refer to page 33). However, the majority of the universal command are not dependent on nor do they affect other commands.

## 4.1 constant



The *constant* command specifies a continuous value for an input, output, or inout pin or port.

### 4.1.1 Usage

**constant**

( -**ports** port_list | -**pins** pin_list ) **0** | **1**

### 4.1.2 Required Keywords

-**ports** port_list

The -*ports* keyword specifies the port or ports to which the constant applies. Either –*ports* or –*pins* (or both) must be specified.

-**pins** pin_list

The -*pins* keyword specifies the pin or pins to which the constant applies. Either –*ports* or –*pins* (or both) must be specified.

### 4.1.3 Optional Keywords

None.

### 4.1.4 Positional Parameters

**0** | **1**

The constant value can be a one or a zero. Any other value will result in an error.

### 4.1.5 Examples

```
constant -ports {grnd} 0
constant -pins {vcc pwr1 pwr2} 1
```

### 4.1.6 Semantics

### 4.1.7 Related Commands

The following commands are related to the *constant* command:

## 4.2 design_name_space

The *design_name_space* command either specifies a predefined name space for design objects or a custom name space.

### 4.2.1 Usage

**design_name_space**

**-verilog** ( **“ 1995 ”** | **“ 2000 “** ) | **-vhdl** ( “ **1987 “** | **“ 1993 “** | **“ 2000** “ ) |
**-edif** ( **“2 0 0 “** | **“ 3 0 0 “** | **“ 4 0 0** “ ) |
**-custom** ( **-characters “ [** character_set **] ”** | **“ [** character_range **] ”** |
**“ [** character_set character_range **] ” |**
**“ [** character_range character_set **] ”** )
( **-case_sensitive | -case_insensitive** )
( **-character_escape “** *escape*_character **”** |
**-string_escape_start “** *escape*_character **“** [ **-string_escape_end “** *escape*_character **”** ] )
( **-escape_type include | exclude** )
( **-bus_range_separator_up “** *index*_character **”** | **“** *index*_identifier **”** )
(**-bus_range_separator_down “** *index*_character **”** | **“** *index*_identifier **”** )
( **-bus_bit_left “** *bit*_character **”** )
( **-bus_bit_right “** *bit*_character **”** )
( **-hierarchy_delimiter “** *delimiter*_character **”** )

### 4.2.2 Required Keywords

One of the following keywords shall be specified:

**-verilog** ( **1995** | **2000** )

The *-verilog* keyword indicates that identifiers in the DCDL file adhere to the name space rules for Verilog, as defined by IEEE 1364. The IEEE version (year of approval) of must be specified in case there are any naming differences in future standards. As of version 0.3.7 of the DCDL standard, there were no such naming differences between Verilog versions.

Table 4-1 describes the main elements of the Verilog name space.

| Rule | Definition |
|---|---|
| Legal characters | a-z, A-Z, 0-9, $, and _ |
| Case | Case-sensitive |
| Escape | Yes with \ (backslash) - whitespace terminates the escape |
| Bus index | : (colon) |
| Bus bit | [ ] (square brackets) |
| Hierarchy delimiter | . (period) |

**Table 4-1—Verilog Name Space Overview**

**-vhdl** ( **1987 | 1993** | **2000** )

The *-vhdl* keyword indicates that the identifiers in the DCDL file adhere to the name space rules for VHDL, as defined by IEEE 1076. The IEEE version (year of approval) of must be specified. There are naming differences between the 1987 and 1993 versions and there could be naming changes in future versions of VHDL.

Table 4-2 describes the main elements of the VHDL name space.

| Rule | Definition |
|---|---|
| Legal characters | a-z, A-Z, 0-9, and _ |
| Case | Case-insensitive (unless the name is escaped - *only in VHDL93*). |
| Escape | Yes with enclosing \ \ (backslash backslash) - *only in VHDL93* |
| Bus index | *to* or *downto* |
| Bus bit | ( ) (parentheses) |
| Hierarchy delimiter | : (colon) returned string from ‘PATH_NAME |

**Table 4-2—VHDL Name Space Overview**

**-edif** ( **“2 0 0 “** | **“ 3 0 0 “** | **“ 4 0 0** “ )

The *-edif* keyword indicates that the identifiers in the DCDL file adhere to the name space rules for EDIF, as defined by IEC and EN 61690. The EDIF version must be specified in case there are any naming differences in future standards.

Table 4-3 describes the main elements of the EDIF name space.

| Rule | Definition |
|---|---|
| Legal characters | a-z, A-Z, 0-9 (first character must be alpha - if not use &) |
| Case | Case-insensitive |
| Escape | None |
| Bus index | None |
| Bus bit | None |
| Hierarchy delimiter | None |

**Table 4-3—EDIF Name Space Overview**

**-custom** ( **-characters “ [** character_set **] ”** | **“ [** character_range **] ”** | **“ [** character_set character_range **] ” |**
**“ [** character_range character_set **] ”** )
( **-case_sensitive | -case_insensitive** )
( **-character_escape “** *escape*_character **”** |
**-string_escape_start “** *escape*_character **“** [ **-string_escape_end “** *escape*_character **”** ] )
( **-escape_type include | exclude** )
( **-bus_index “** *index*_character **”** | **“** *index*_identifier **”** )
( **-bus_bit_left “** *bit*_character **”** )

( -**bus_bit_right “** *bit*_character **”** )
( -**hierarchy_delimiter “** *delimiter*_character **”** )

The -*custom* keyword defines the custom name space rules. This keyword allows for a programatic solution to custom design flows with respect to name spaces.

Required keywords for -*custom*:

-**characters “ [** character_set **] ”** | **“ [** character_range **] ”** | **“ [** character_set character_range **] ”** |
**“ [** character_range character_set **] ”**

The -*characters* keyword specifies one or more characters that are allowed in identifiers. A range of characters can be specified as well as multiple character sets. Any combination of ranges and character sets (a character set can contain 1 character) can be specified. For example, to specify all characters of the alphabet, some special characters (_ and @), and numbers:

```
-characters “[a-zA-Z_@0-9]”
```

-**case_sensitive** | -**case_insensitive**

The -*case_sensitive* or -*case_insensitive* keywords specify whether the identifiers are differentiated by case (*sensitive*) or whether identifiers are case insensitive (*insensitive*).

-**character_escape “** *escape*_character **”**

The -*character_escape* keyword specifies the method of specifying an escape character to escape the character that follows. The escaping ends after the first character that follows the escape character.

-**string_escape_start “** *escape*_character **“** [ -**string_escape_end “** *escape*_character **”** ]

The -*string_escape_start* and -*string_escape_end* keywords specify a beginning and an optional end character to indicate that the identifier string between these characters should be escaped.

-**escape_type include** | **exclude**

The -*escape_type* keyword specifies whether the escape character is included as part of the identifier (*include*) or not (*exclude*).

-**bus_range_separator_up “** *index*_character **”** | **“** *index*_identifier **”**
-**bus_range_separator_down “** *index*_character **”** | **“** *index*_identifier **”**

The -*bus_range_separator_up* and -*bus_range_separator_down* keywords define the character or identifier that separates the bus MSB and LSB values. For example, a colon is the bus index for: `bus_a[0:31]`. The identifiers *to* and *downto* are the bus indices for: `bus_b(0 to 7)` `bus_c(7 downto 0)`. If the separator character or identifier is the same for up and down, that character or identifier is specified for both of the keywords. For example, if colon is used for both up and down:

```
-bus_range_separator_up “:”
-bus_range_separator_down “:”
```

-**bus_bit_left “** *bit*_character **”** )
-**bus_bit_right “** *bit*_character **”** )

The *-bus_bit_left* and *-bus_bit_right* keywords specify the pair of characters that delimit single or multiple bus bits. These characters separate the bus identifier and the bits. For example, left and right brackets surround the bus bit 1:

```
bus_b[1]
```

-**hierarchy_delimiter “** *delimiter*_character **”**

The *-hierarchy_delimiter* keyword specifies the character that separates levels of hierarchy within pathnames.

### 4.2.3 Optional Keywords

None.

### 4.2.4 Positional Parameters

None.

### 4.2.5 Examples

```
design_name_space -custom -characters “[a-zA-Z0-9]” -case_sensitive \
-character_escape “\\” -bus_range_separator_up “:” \
-bus_range_separator_down “:” -bus_bit_left “[“ -bus_bit_right “]” \
-hierarchy_delimiter “/”
```

The preceding example shows the design name space rules that match SDF. The escape character must be escaped (see page 29).

```
design_name_space -vhdl “1993”
design_name_space -verilog “2000”
```

The preceding examples show the specification of the VHDL 1993 and Verilog 2000 name spaces.

### 4.2.6 Semantics

The *design_name_space* command specifies what naming rules were followed for DCDL design object identifiers. Refer to page 31 for information about DCDL design objects. If a particular tool chooses to not support any of the predefined name spaces (Verilog, VHDL, and EDIF), the tool shall report an error if these keywords are encountered.

If no *design_name_space* command exists in the DCDL file, the Verilog name space, version 1995 is assumed.

The *design_name_space* definition applies to all design object identifiers in a DCDL file from the point that the command exists in the file until another *design_name_space* command appears (known as a file scope command - refer to page 38).

If new versions of the VHDL and Verilog standards exist before they are supported as keyword values in DCDL, the user must work with the tool vendor for support or use the *extend_dcdl* command. If the tool does not support the version indicated, it shall report an error.

For custom design name space definitions, if a particular rule does not apply, the keyword value is left blank (two double quotes without whitespace). For example, if the name space does not support an escaping mechanism:

```
-character_escape “”
```

The *-characters* keyword allows a small subset of regular expressions as defined by the POSIX 1003.2 standard. The *-characters* keyword can specify a range of characters that evaluate to single characters of a design object string. This keyword does not express a pattern of characters to match, rather the set of characters allowed in a design object identifier. The keyword value(s) that follow *-characters* must be double quoted and surrounded by square braces [] with no space separation between the ranges and/or characters. The DCDL escape characters can be used in the keyword values (refer to page 29). In addition, the regular expression notation of using the circumflex ^ is also allowed for *-characters*, to indicate a set of characters that are not allowed in a design object identifier. Here are some examples:

To specify a range of letters allowed:

```
-characters “[A-Za-z]”
```

To specify that all characters except the % and $ characters are allowed, the circumflex notation is used:

```
-characters “[^%$]”
```

To specify a set of characters that include those that must be escaped:

```
-characters “[A-Za-z\]]”
```

In the preceding example, the right brace character must be escaped because it is also used to denote the regular expression (part of the command syntax).

For each keyword value that is a string (double quoted), the rules of DCDL strings apply (refer to page 29). For example, if the *-escape_string_start* keyword value is \, that character must be escaped also because it is a reserved character. If tab is a keyword value in *-escape_string_end*, the *t* must be used:

```
design_name_space -custom -characters “[a-zA-Z0-9_] \
-case_sensitive -string_escape_start “\\” -string_escape_end “\t” \
-bus_range_separator_up “:” -bus_range_separator_down “:” \
-bus_bit_left “(“ -bus_bit_right “)” -hierarchy_delimiter “.”
```

### 4.2.7 Related Commands

The following commands are related to the *design_name_space* command:

*extend_dcdl*
*include*

## 4.3 extend_dcdl



The *extend_dcdl* command provides a method to call non-standard DCDL commands.

### 4.3.1 Usage

**extend_dcdl**

*command*_identifier [ -**arguments “** *argument*_text **“** ]

### 4.3.2 Required Keywords

None.

### 4.3.3 Optional Keywords

-**arguments “** *argument*_text **“**

The -**arguments** keyword specifies all the arguments to the extended command, surrounded by required double quotes.

### 4.3.4 Positional Parameters

*command*_identifier

The *command_identifier* parameter provides a name for the extended command. A DCDL command name can be used for this identifier.

The suffix *DCDL_* is reserved for use by the DC-WG within the *extend_dcdl* command. The designer cannot use this suffix in the *command_identifier* parameter.

### 4.3.5 Examples

```
extend_dcdl custom_cmd -arguments “-ports {p11 p14} 1.1063”
```

The preceding example shows a call to a non-standard command called *custom_cmd* that associates the value *1.1063* to the *p11* and *p14* ports.

```
extend_dcdl acme_include -arguments “/net/fish/includes/rf.txt”
```

The preceding example shows the use of a corporate suffix *acme_* to call a custom *include* command.

### 4.3.6 Semantics

The main use for the *extend_dcdl* command is to prototype new DCDL commands that might be included in the next version of the specification. Secondarily, DCDL can be extended in order to support company-specific design flows or specific tools. This allows the designer to pass tool-specific commands to the tools that understand them within the DCDL file.

The designer uses the *extend_dcdl* command at his or her own risk. There is the potential that downstream tools will not understand the command or that the command will be included (not necessarily using the same syntax or semantics) in future versions of the specification.

If a the tool that processes the DCDL file does not recognize the extended command, it shall be ignored.

The *extend_dcdl* command is persistent only to the scope in which it appears. This means that when the designer writes out a DCDL file from an application, the *extend_dcdl* command will appear within the file at the location of the original module or instance, if that scope still exists. If the scope no longer exists, the *extend_dcdl* command will not appear in the output file. The actual location in the output file is not defined by the standard, other than that defined by the preceding scope rules. For more information about scope, refer to page 61.

*** Did not emulate the *extension* command from GCF in terms of including data, as that can be done with the include command. This makes *extend_dcdl* cleaner. ***

### 4.3.7 Related Commands

The following commands are related to the *extend_dcdl* command:

*include*

## 4.4 functional_mode



The *functional_mode* command selects the state-dependent effects (or mode) for analysis of instances.

### 4.4.1 Usage

**functional_mode**

( [ **-group_name** *group*_identifier ] **-mode_name** *mode*_identifier ) | ( **-all** | **-default** ) instance_list

### 4.4.2 Required Keywords

Either:

**-mode_name** *mode*_identifier

The *-mode_name* keyword specifies the name of a mode as defined by the library. The optional group name can also be specified to indicate a mode within a group.

Or:

-**all**

The *-all* keyword specifies that all the modes in all the mode groups should be used for analysis.

-**default**

The *-default* keyword specifies that the default mode (as specified in the library) should be used for analysis.

### 4.4.3 Optional Keywords

**-group_name** *group*_identifier

The *-group_name* keyword specifies a name for the group of modes specified either within a design tool or a library. Grouping is a convenient method for activating and de-activating sets of modes.

### 4.4.4 Positional Parameters

instance_list

The *instance_list* specifies a list of one or more instance names to which the active mode or modes apply.

### 4.4.5 Examples

```
functional_mode -all {inst1 inst2}
```

The preceding example would make active all the modes in all mode groups defined for instances *inst1* and *inst2*.

```
functional_mode -group mode_gr1 {u1}
```

The preceding example would make active all the modes in the group *mode_gr1* for instance *u1*.

```
functional_mode -group mode_gr2 -mode_name write {ram1}
```

The preceding example would make active the *write* mode within the group *mode_gr2* for instance *ram1*. All the other modes in *mode_gr2* would become inactive unless another *functional_mode* command explicitly activated them.

### 4.4.6 Semantics

The behavior of complex blocks in a design will often be highly state-dependent. For example, the internal operations performed by microprocessors and DSP cores can be radically different depending on the instructions they execute. Even a simple RAM will have different behavior depending on whether data is being read or written.

Modes are a way to simplify the process of defining state-dependency by assigning a mnemonic label to each of the states of a block, and using that label to specify valid combinations of states involving several blocks. Each combination of states can have a label as well (a mode group), so that it is possible to define all of the combinations up front within a cell library, and then perform an analysis for a particular combination by simply setting the design mode using the *functional_mode* command.

While the library controls the legal values of the *functional_mode* command, DCDL does define several aspects of mode semantics:

Once a mode is set, it remains set from the point that the command exists in the file, until another value for that mode appears in the file (known as a file scope command - refer to page 38)

Individual mode names, mode group names, and the names within mode groups must be unique. However, two or more mode groups can contain alike mode names.

Modes can be unconditional or conditional. Conditional modes use conditional expressions. Conditional and unconditional modes can be combined within mode groups.

**Unconditional modes**. If the *functional_mode* command is not used, all unconditional modes are assumed to be active. If the unconditional mode is set in a mode group, all other unconditional modes in the group become inactive. Setting the mode in one mode group has no affect on any other mode groups.

**Conditional modes**. If the *functional_mode* command is not used, all conditional modes are assumed active. The conditional mode is set if explicitly specified by the *functional_mode* command. If not explicitly set, the mode is set if the conditional expression associated with the mode evaluates to true. Once one conditional mode in a mode group is set, all other modes in the group are disabled. Setting the mode in one mode group has no affect on any other mode groups.

If a timing arc is governed by both an unconditional mode and a conditional mode, the arc is enabled only if the mode is set and the conditional expression evaluates to true.

If a timing arc is controlled by multiple modes within a mode group, the arc is enabled if any one of the modes is active. Timing arcs controlled by multiple mode groups must be enabled by a mode in every one of those mode groups. For example, if the *RW* mode group contained modes *read* and *write* and the second mode group *CHIP_ENABLE* contained modes *on* and *off*, an arc that depended on *read* and *on* would not be enabled if the active mode for *CHIP_ENABLE* was *off*.

*** Need to discuss the effect of ordering on mode interpretation. Also, need to add a methodology for case analysis (design mode) - when in a mode - certain constraints apply. Can do this with another cmd, by scoping of *functional_mode*, or by adding options to the appropriate cmds. ***

### 4.4.7 Related Commands

The following commands are related to the *functional_mode* command:

*operating_process*
*operating_range*
*operating_temperature*
*operating_voltage*
*units*
*voltage_regime*

## 4.5 history



The *history* command provides a placeholder for comments about the lineage of the DCDL file.

### 4.5.1 Usage

**history**

**“** *history*_text **”**

### 4.5.2 Required Keywords

None.

### 4.5.3 Optional Keywords

None.

### 4.5.4 Positional Parameters

**“** *history*_text **”**

The *history_text* can contain any non-reserved characters to indicate history information. The double quotes are required.

### 4.5.5 Examples

```
history “Updated clock tree values on 12/25 by writer dcdl_write”
```

### 4.5.6 Semantics

The *history* command provides a means to document creation, changes, and any other relevant information about the DCDL file. DCDL readers can parse the file and interpret the text of the *history* command as needed.

The *history* command is persistent only to the scope in which it appears. This means that when the designer writes out a DCDL file from an application, the *history* command will appear within the file at the location of the original module or instance, if that scope still exists. If the scope no longer exists, the *history* command will not appear in the output file. The actual location in the output file is not defined by the standard, other than that defined by the preceding scope rules. Typically, the *history* command will appear at the top of the scope. For more information about scope, refer to page 61.

### 4.5.7 Related Commands

The following commands are related to the *history* command:

*include*
*version*

## 4.6 include

The *include* command inserts DCDL commands from another file.

### 4.6.1 Usage

**include**

[ **-inline** ] **“** *pathname*_identifier **”**

### 4.6.2 Required Keywords

None.

### 4.6.3 Optional Keywords

**-inline**

The **-inline** keyword indicates that any file scope command (refer to page 38) in the included file will be in affect for the parent file, until another file scope command is encountered.

For example, *units* is a file scope command that applies to an entire DCDL file until another *units* command is encountered. If the included file contains a *units* command and *-inline* is specified, the *units* command value within the included file remains in affect for the parent DCDL file until another *units* command is encountered.

If **-inline** is not specified, file scope commands are only in affect for the included file. At the end of the included file, the parent file scope commands resume control.

### 4.6.4 Positional Parameters

*pathname*_identifier

The *pathname_identifier* specifies the pathname to the file to be included. This identifier can use any non-reserved character. The *pathname_identifier* must specify a path relative to the parent (or calling) DCDL file for portability. Full pathnames should not be used.

### 4.6.5 Examples

```
include “design1/common_cnstr.dcdl”
include -inline “design2/com_cstr.dcdl”
```

### 4.6.6 Semantics

The include command is treated as including multiple physical files into a single file. At the location in the DCDL file that the *include* command exists, DCDL commands from the pathname specified are inserted. If the pathname does not exist or the file cannot be opened, an error shall be issued. The included DCDL must be syntactically and semantically legal for the location that it is being inserted.

Including files is a means to share DCDL commands and to assemble multiple block constraints from separate sources into one file. For example, file scope DCDL commands such as *units* and *design_name_space* can be specified once and included in many, individual DCDL files.

Recursive includes are supported by making sure that the *pathame_identifier* is always relative to the parent DCDL file.

The DCDL standard does not specify how a tool processes or writes out the include command. However, it is assumed that the include structure matches the current design hierarchy.

### 4.6.7 Related Commands

The following commands are related to the *include* command:

*extend_dcdl*

## 4.7 units



The *units* command specifies a quantity in terms of a multiplier for time, capacitance, resistance, voltage, and temperature values.

### 4.7.1 Usage

**units**

[ -**time** multiplier ] [ -**capacitance** multiplier ] [ -**resistance** multiplier ] [ -**voltage** multiplier ]
[ -**temperature** multiplier ] [ -**inductance** multiplier ]

### 4.7.2 Required Keywords

None.

### 4.7.3 Optional Keywords

-**time** multiplier

The -*time* keyword specifies a multiplier for all command values that express time.

The base unit is seconds.

-**capacitance** multiplier

The -*capacitance* keyword specifies a multiplier for all command values that express capacitance.

The base unit is farads.

-**resistance** multiplier

The -*resistance* keyword specifies a multiplier for all command values that express resistance.

The base unit is ohms.

-**voltage** multiplier

The -*voltage* keyword specifies a multiplier for all command values that express voltage.

The base unit is volts.

-**temperature** multiplier

The -*temperature* keyword specifies a multiplier for all command values that express temperature. Temperature refers to junction temperature.

The base unit is celsius.

-**inductance** multiplier

The -*inductance* keyword specifies a multiplier for all command values that express inductance.

The base unit is henries.

The *multiplier* specifies a number that shall be used to scale the base unit. If an optional keyword is specified, the multiplier for that keyword is required. Scientific notation is allowed.

Default: 1

### 4.7.4 Positional Parameters

None.

### 4.7.5 Examples

```
units -time 1.0E-9 -capacitance 1.0E-12
```

The preceding example sets the following units: time in nanoseconds and capacitance in pico farads. All other units are set to the default base units.

```
units -time 1.0E-12
units -voltage 1.0E-3
```

The preceding example shows the use of individual units commands to specify time in picoseconds and voltage in milli-volts. All other units are set to the default base units.

### 4.7.6 Semantics

The units defined by this command are either base or derived units from the International System of Units standard.

Certain keyword values of commands specify a number value and that number is a quantity with some unit type. The *units* command affects these types of values from the point that the command exists in the file, until another *units* command appears (known as a file scope command - refer to page 38). If the *units* command is not specified, all affected commands will use the default values.

The *units* command follows a similar concept as value slots (refer page 36) in that the command can set one value, or several values at the same time. For example, a *units* command can set only capacitance and later in a file, another *units* command can set time and leave capacitance alone.

### 4.7.7 Related Commands

The following commands are related to the *units* command:

*include*

## 4.8 version



The *version* command identifies the DCDL specification version to which the commands that follow reference.

### 4.8.1 Usage

**version**

version_identifier

### 4.8.2 Required Keywords

None.

### 4.8.3 Optional Keywords

None.

### 4.8.4 Positional Parameters

version_identifier

The *version_identifier* provides the DCDL specification version number. The *version_identifier* can only be *1.0* and must be surrounded by quotes.

Default: “1.0”

### 4.8.5 Examples

```
version “1.0”
```

This example shows that the DCDL specification version utilized is 1.0.

### 4.8.6 Semantics

DCDL readers can utilize the *version* command to adapt to any differences between DCDL specification versions. DCDL readers might allow specification of one or more versions.

The *version* command must appear in a DCDL description before any version-dependent command is specified. The *version* definition applies to all commands in a DCDL description from the point that the command exists in the file until another *version* command appears (known as a file scope command - refer to page 38).

### 4.8.7 Related Commands

The following commands are related to the *version* command:

*history*

*1*

*5*

*10*

*15*

*20*

*25*

*30*

*35*

*40*

*45*

*50*

# 5. Scoping Commands

This section describes the method used to indicate the location within a design to which the DCDL commands that follow apply.

## 5.1 Scoping Theory

In the context of DCDL scoping refers to the following aspects of the language:

**Design scope**: the region or level of the design to which the DCDL commands apply. DCDL assumes that there exists a top level of a design and that zero or more designs, blocks, or instances exist “below” that top level. This is set using the *current_scope* command (refer to page 63).

**File scope**: the region within the actual DCDL file. A file can be logical - a file with all includes expanded, or a physical file - individual file with includes unexpanded. DCDL contains several file scope commands whose definition applies to all design object identifiers in a DCDL file from the point that the command exists in the file until the command appears again.

Figure 3-2 shows the general DCDL scoping concept.

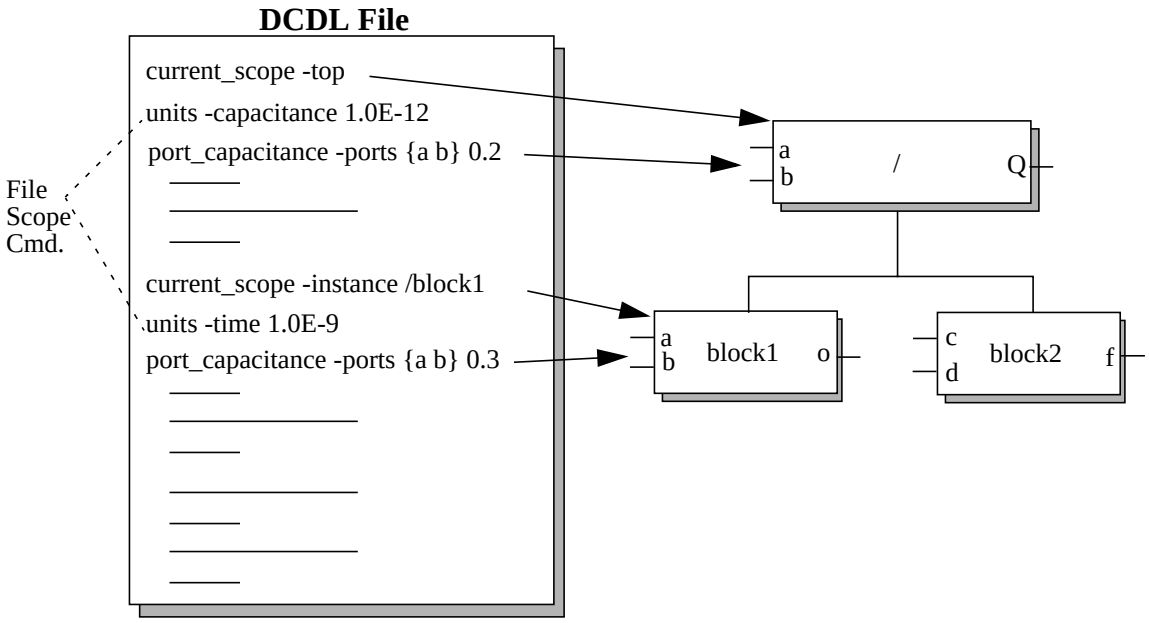



**Figure 3-2 General DCDL Scoping Concept**

Figure 3-2 presents the following concepts through example:

The first design scope is set to the top level, which is /. This is the default if the *current_scope* command is not used. All non-file scope DCDL commands that follow apply to this block until the scope is changed.

The file scope command *units* sets the capacitance units to picofarads. Any DCDL command expecting a capacitance unit will use picofarads until another *units* command is found in the file that sets capacitance to another unit. In this example, loads of .2 picofarads are specified for ports *a* and *b* on /. The scope is changed to the lower-level block called *block1* and the units are changed to nanofarads. In this example, an external load of .3 nanofarads is specified for ports *a* and *b* of *block1*.

The file scope concept takes on special meaning when using the *include* command. By default, the *include* command assumes that any file scope commands in the included file are in effect only within that included file - in essence a new scope is created. If the *-inline* keyword is specified, any file scope command in the parent file is in effect for the included file, until the included file specifies a file scope command - in effect "inlining" the include file as if it was physically part of the parent file. Figure 3-3 presents this concept.

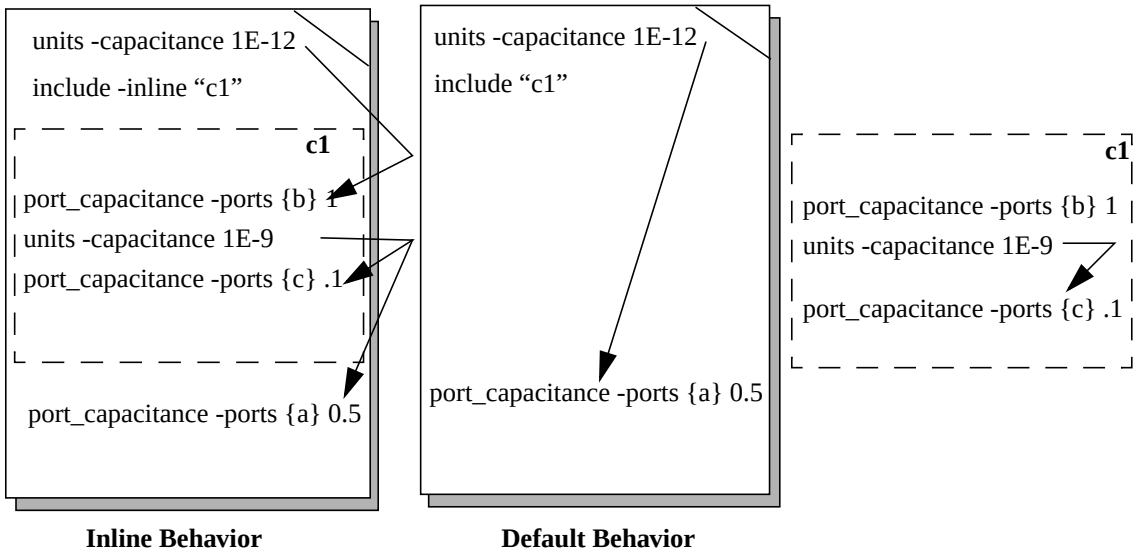



**Figure 3-3 File Scopes and Includes**

Figure 3-3 presents the following concepts through example:

**Inline behavior**. The *c1* file is included using the *-inline* option. The capacitance units for the parent file is picofarads. Thus, the external load applied to port *b* is in picofarads. However, the units for capacitance are set to nanofarads in the included file, so any external load within *c1* and the parent file uses nanofarads - until another *units -capacitance* command is specified.

**Default behavior**. The *c1* file is included without the *-inline* option. The external load units for port *b* is whatever it was set to previously to in *c1*. If no units were specified, the default capacitance unit is 1 farad. The *units* command in *c1* sets the capacitance units to nanofarads. Thus the external load for port *c* is set to .1 nanofarads. However, the external load on port *a* is set to 0.5 picofarads, the units set by the parent file.

Refer to page 55 for details about the *include* command.

A concept related to scope is the application of DCDL commands to a cell versus an instance. If a command is applied to a cell, all instances of that cell in the design receive the command value. If a command is applied to an instance, only that instance receives the command value.

## 5.2 current_scope

The *current_scope* command establishes the design level where all the referenced design objects can be found by subsequent commands.

### 5.2.1 Usage

**current_scope**

**-instance** *instance*_identifier | **-cell** *cell*_identifier | **-top** | **-up** *level*_unsigned_number

### 5.2.2 Required Keywords

One of the following:

**-instance** *instance*_identifier

The *-instance* keyword provides the instance name that is considered the root of the hierarchy for all design object searches. Only one instance name can be specified.

**-cell** *cell*_identifier

The *-cell* keyword provides the cell name that is considered the root of the hierarchy for all design object searches. Only one cell name can be specified.

For the *instance_identifier* and *cell_identifier* constructs, pathnames can be used (refer to page 32). If whitespace is used for these constructs, the identifier must be surrounded by double quotes.

**-top**

The *-top* keyword provides a means to specify that the top of the design is the root of the hierarchy for all design object searches.

**-up** *level*_unsigned_number

The *-up* keyword provides a method to move up a specified number of levels in the hierarchy relative to the current position. The level is indicated using a positive integer.

### 5.2.3 Optional Keywords

None.

### 5.2.4 Positional Parameters

None.

### 5.2.5 Examples

```
current_scope -instance U2
current_scope -instance “/top/u2/my design”
current_scope -cell mux32
current_scope -top
current_scope -up 2
```

### 5.2.6 Semantics

If the *current_scope* command is not used, the scope defaults to the top level of the design.

If the specified instance or cell does not exist, an error shall be issued. If *-up* specifies a number of levels that takes the level past the top of the design, an error shall be issued.

### 5.2.7 Related Commands

The following commands are related to the *current_scope* command:

# 6. Operating Conditions

This section presents the set of commands used to specify the conditions in which a design or portion of a design will operate.

## 6.1 Operating Conditions Theory

In general, the library defines the legal operating conditions, the effects on the cells, and the valid user-specified parameters. The role of operating condition commands in DCDL is to communicate the value of any parameter that is available in the library. These parameters (or variables) fall into two categories:

**Named**: operating points predefined by the library vendor identified with a label.

**Explicit**: operating point values (numbers) provided via the DCDL command and used in the library in an operating condition equation.

DCDL provides the flexibility to specify both named and explicit values in individual commands, or all at once in a single command. In general, the commands provide a means to select a set of characterization data and then specify how that data should be used. However, the library always arbitrates the legal keyword and keyword values.

### 6.1.1 Correlation and Operating Conditions

Correlation specifies whether early and late delays and slews should be computed assuming that variations in operating conditions are on the chip (correlated) or between chips (uncorrelated).

DCDL defines separate commands that specify operating process, voltage, and temperature. These commands contain process point options (*-best*, *-nominal*, *-worst*, *-min_best*, *-typ_best*, *-max_best*, *-min_worst*, *-typ_worst*, and *-max_worst*) that indicate which operating point is being defined, and how that operating point should be interpreted:

For an uncorrelated analysis using just the operating extremes, both *-best* and *-worst* operating points should be specified.

For an analysis that considers correlated variations on-chip under best case conditions, both *-min_best* and *-max_best* operating points should be specified.

For an analysis that considers correlated variations on-chip under worst case conditions, both *-min_worst* and *-max_worst* operating points should be specified.

For an analysis that simultaneously considers correlated variations on-chip under best case conditions, and correlated variations on-chip under worst case conditions, *-min_best*, *-max_best*, *-min_worst*, and *-max_worst* operating points should be specified.

For delay calculation to generate SDF min:typ:max triplets, a *-nominal*, *-typ_best*, or *-typ_worst* operating point can be specified.

### 6.1.2 Regimes

*** To be added. Discussion of voltage and temperature regimes. Regimes are also known as islands. Physical relationships, concept of min,typ,max and the relationship to base voltages to be added.***

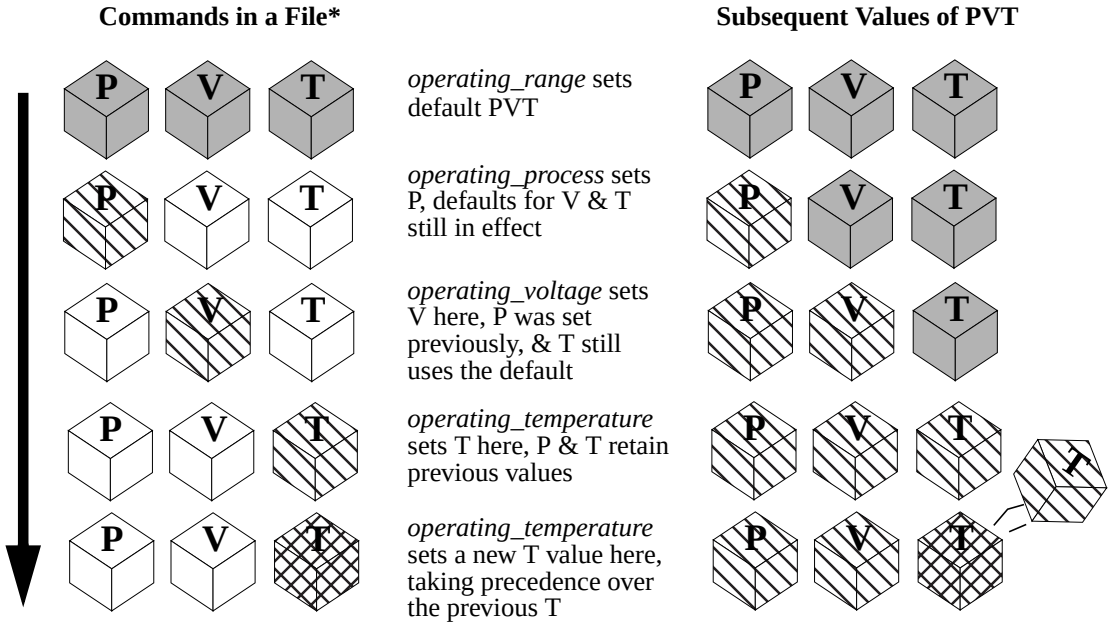
### 6.1.3 Operating Condition Command Precedence

In general, DCDL defines precedence such that a the more specific a value on a specific design object always has precedence over a general value or a higher-level design object. This is true of operating conditions as well - but with added scenarios:

The *operating_range* command specifies default values for process, voltage, and temperature. Specifically setting these points with the *operating_process*, *operating_temperature*, and/or *operating_voltage* commands takes precedence over these defaults.

Operating conditions support the concept of inheritance as discussed in the next section.

Like many DCDL commands, many operating condition commands can specify more than one value. This is known as the value slot concept (refer to page 36). For operating conditions, value slots refer to either process, temperature, or voltage values; or minimum and maximum values. If a certain value slot is filled by an earlier command, any following command that also fills that slot, has precedence - but any undefined slots remain at their last value. Figure 3-4 shows value slots and precedence concepts.




* Assumes the same scope & design object level

**Figure 3-4 Operating Condition Value Slots and Precedence**

The value slot precedence is affected by:

Voltage and temperature regimes, because these options create an extra value slot. In order for two operating conditions to match (and thus invoke a precedence rule), the regime slot must also match if present (refer to page 37 for more information about matching).

Library rules can define legal value slot values and if an error occurs, the value slot with retain its original value.

### 6.1.4 Operating Condition Command Inheritance

DCDL provides no general command inheritance specification. However, operating condition commands specified at certain scopes or applied to particular cells or instances are inherited by lower levels. This allows

a natural and efficient means to specify general operating conditions and differing particular conditions at lower levels. If an operating condition is specified at the top level of a design and no other operating conditon commands are specified, those conditions are inherited by all levels of the design. If operating conditions are specified at a particular scope, all the design objects from that scope down in the hierarchy use those operating conditions. This applies to hierarchical cells or instances. Thus, an operating condition applies to all the contents of the scope, cell, or instance to which it is assigned.

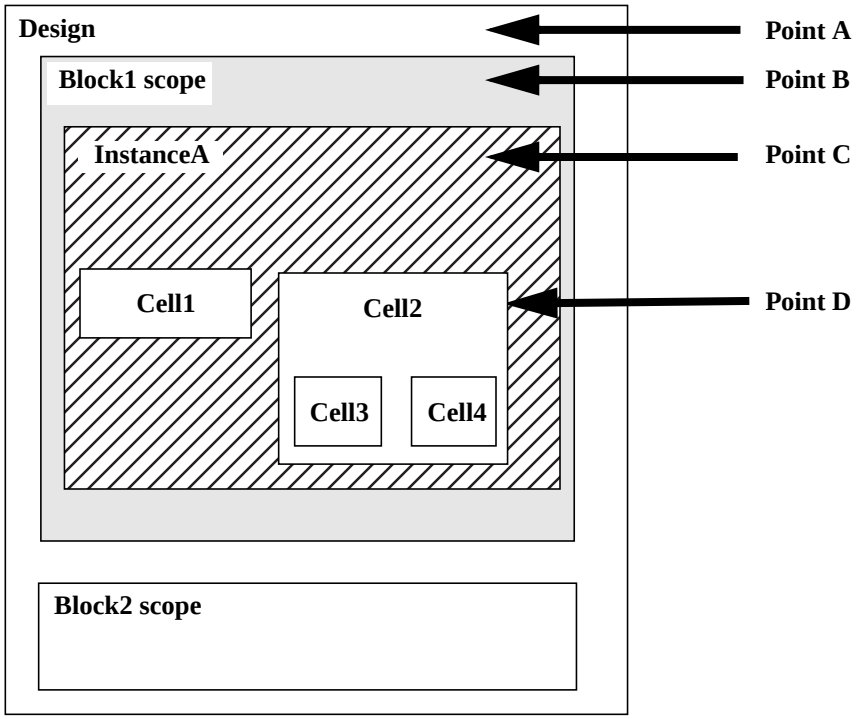



**Figure 3-5 Example of Operating Condition Inheritance**

The example in Figure 3-5 provides the following points about inheritance:

An operating condition set at *Point A* would be in affect for the whole design, until a more specific constraint is set.

An operating condition set at *Point B* would be in affect for all objects contained in *Block1*, but not *Block2*.

An operating condition set at *Point C* on *InstanceA* would be in affect for all objects contained in that instance (*Cell1-4*).

An operating condition set at *Point D* on *Cell2* would be in affect for *Cell3* and *Cell4.*

### 6.1.5 Operating Condition Precedence and Inheritance Interactions

Operating condition command precedence and inheritance interact with each other following these rules:

Specific cell or instance values take precedence over scope-level values.

The narrower (more specific) scope value takes precedence over a more general scope value.

If two commands set an operating condition on a cell and an instance that contains the cell, the instance value takes precedence (more specific).

## 6.2 operating_point



The *operating_point* command provides a means to set the process, temperature, and voltage operating point for a design all at once (as opposed to using the separate commands available).

### 6.2.1 Usage

**operating_point**

[ -**voltage_regime** *voltage_regime*_identifier ] [ -**temperature_regime** *temperature_regime*_identifier ]
[ -**library** *library*_identifier ] -**name** *operating_point*_identifier
( -**best** | -**nominal** | -**worst** | -**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst** )

### 6.2.2 Required Keywords

-**name** *operating_point*_identifier

The -*name* keyword specifies the operating point by name - what set of characterization data should be used. The operating points are named and defined in the current technology library for the design or in the library explicitly named using the optional -*library* keyword.

-**best** | -**nominal** | -**worst** | -**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst**

These keywords specify the operating points - how the named set of characterization data should be used. The operating points are defined in the current technology library for the design or in the library explicitly named using the optional -*library* keyword.

Refer to page 65 for general information about using these keywords.

### 6.2.3 Optional Keywords

-**voltage_regime** *voltage_regime*_identifier

The -*voltage_regime* keyword associates operating point information with a previously-defined voltage regime through the use of the *voltage_regime* command. This option only applies to the voltage value slot.

-**temperature_regime** *temperature_regime*_identifier

The -*temperature_regime* keyword associates operating temperature information with a previously-defined temperature regime through the use of the *temperature_regime* command. This option only applies to the temperature value slot.

-**library** *library*_identifier

The -*library* keyword associates an explicit library with the operating point.

Default: the operating point specified applies to all the libraries used in the design.

### 6.2.4 Positional Parameters

None.

### 6.2.5 Examples

```
operating_point -name MIL_worst -worst
operating_point -library BC_lib -name COM_best -best
```

### 6.2.6 Semantics

The *operating_point* command provides a convenient method of specifying a single operating point as defined in a library.

### 6.2.7 Related Commands

The following commands are related to the *operating_point* command:

*operating_process*
*operating_range*
*operating_temperature*
*operating_voltage*
*temperature_regime*
*voltage_regime*

## 6.3 operating_process

●

The *operating_process* command specifies the process condition that should be applied to the design. Several specification methods are available.

### 6.3.1 Usage

**operating_process**

[ -**library** *library*_identifier ]
[ -**value** *operating_point*_rvalue ] ( -**best** | -**nominal** | -**worst** | -**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst** )

### 6.3.2 Required Keywords

**-best** | **-nominal** | **-worst | -min_best** | **-typ_best** | **-max_best** | **-min_worst** | **-typ_worst** | **-max_worst**

These keywords specify the process points. The process points are defined in the current technology library for the design or in the library explicitly named using the optional -*library* keyword.

Refer to page 65 for general information about using these keywords.

### 6.3.3 Optional Keywords

-**library** *library*_identifier

The -*library* keyword associates an explicit library with the operating process.

Default: the operating process specified applies to all the libraries used in the design.

-**value** *operating_point*_rvalue

The -*value* keyword allows the process point to explicitly be defined using a real number. This value is typically a variable within a library equation.

### 6.3.4 Positional Parameters

None.

### 6.3.5 Examples

```
operating_process -library acme -nominal
operating_process -best -value 1.0
```

### 6.3.6 Semantics

If the library pre-defines a process value associated with an operating point, the -*value* option in not required. However, some libraries might require the -*value* option.

### 6.3.7 Related Commands

The following commands are related to the *operating_process* command:

*operating_point*
*operating_range*
*operating_temperature*
*operating_voltage*
*units*
*voltage_regime*

## 6.4 operating_range



The *operating_range* command provides a method to specify a *default* range of operating conditions for a design (or design portion) through the use of a operating name specified in a technology library.

### 6.4.1 Usage

**operating_range**

[ -**library** *library*_identifier ] *operating_range*_identifier

### 6.4.2 Required Keywords

None.

### 6.4.3 Optional Keywords

-**library** *library*_identifier

The -*library* keyword specifies the name of the technology library.

If no library is specified, the current technology library for the design (as specified using a mechanism within a tool or supporting tool library) is assumed.

### 6.4.4 Positional Parameters

*operating_range*_identifier

The *operating_range_identifier* specifies a name for the operating range as specified in the library.

### 6.4.5 Examples

```
operating_range -library com_lib COMMERCIAL
```

### 6.4.6 Semantics

The *operating_range* command is intended to specify default process, voltage, and temperature values by indicating a library “label” that represents a particular model characterization. This label must be defined within the library in order to be specified using the *operating_range* command. If not, it is an error.

The *operating_process*, *operating_temperature*, and *operating_voltage* commands override values specified by the *operating_range* command, because *operating_range* provides defaults and these commands provide more specific values (refer to 6.1.3 “Operating Condition Command Precedence” ).

Some libraries might require specific values for process, temperature, and voltage and thus not allow the use of *operating_range*.

### 6.4.7 Related Commands

The following commands are related to the *temperature_regime* command:

*operating_process*
*operating_temperature*
*operating_voltage*

## 6.5 operating_temperature



The *operating_temperature* command specifies the temperature value that should be applied to the design. Several specification methods are available.

### 6.5.1 Usage

**operating_temperature**

( [ -**temperature_regime** *temperature_regime*_identifier ] |
[ -**instances** instance_list [ -**pin** *pin*_identifier ] )
[ -**library** *library*_identifier ] [ -**value** *operating_point*_rvalue ] ( -**best** | -**nominal** | -**worst** |
-**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst** )

### 6.5.2 Required Keywords

-**best** | -**nominal** | -**worst** | -**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst**

These keywords specify the temperature points. The temperature points are defined in the current technology library for the design or in the library explicitly named using the optional -*library* keyword.

Refer to page 65 for general information about using these keywords.

### 6.5.3 Optional Keywords

-**temperature_regime** *temperature_regime*_identifier

The -*temperature_regime* keyword associates operating temperature information with a previously-defined temperature regime through the use of the *temperature_regime* command. This option only applies to the temperature value slot.

-**instances** instance_list [-**pin** *pin*_identifier]

The -*instances* keyword specifies one or more instance names that should be assigned the temperature value.

The optional -*pin* keyword specifies the pin name on the instance.

Either -*temperature_regime* or -*instances* can be specified (not both).

-**library** *library*_identifier

The -*library* keyword associates an explicit library with the operating temperature.

Default: the operating temperature specified applies to all the libraries used in the design.

-**value** *operating_point*_rvalue

The -*value* keyword allows the temperature point to explicitly be defined using a real number. This value is typically a variable within a library equation.

### 6.5.4 Positional Parameters

None.

### 6.5.5 Examples

```
operating_temperature -temperature_regime low_temp -best
operating_temperature -best -value 20.0
operating_temperature -instances alu -pin in1 -best -value 21.5
```

### 6.5.6 Semantics

The *operating_temperature* command specifies junction temperatures.

### 6.5.7 Related Commands

The following commands are related to the *operating_temperature* command:

*operating_point*
*operating_process*
*operating_range*
*operating_voltage*
*temperature_regime*
*units*
*voltage_regime*

## 6.6 operating_voltage



The *operating_voltage* command specifies the voltage value that should be applied to the design. Several specification methods are available.

### 6.6.1 Usage

**operating_voltage**

( [ -**voltage_regime** *voltage_regime*_identifier ] |
[ -**instances** instance_list [ -**pin** *pin*_identifier ] )
[ -**library** *library*_identifier ] [ -**value** *operating_point*_rvalue ] ( -**best** | -**nominal** | -**worst** |
-**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst** )

### 6.6.2 Required Keywords

-**best** | -**nominal** | -**worst** | -**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst**

These keywords specify the voltage points. The voltage points are defined in the current technology library for the design or in the library explicitly named using the optional -*library* keyword.

Refer to page 65 for general information about using these keywords.

### 6.6.3 Optional Keywords

-**voltage_regime** *voltage_regime*_identifier

The -*voltage_regime* keyword associates operating voltage information with a previously-defined power regime through the use of the *voltage_regime* command. This option only applies to the voltage value slot.

-**instances** instance_list [-**pin** *pin*_identifier]

The -*instances* keyword specifies one or more instance names that should be assigned the voltage value.

The optional -*pin* keyword specifies the pin name on the instance.

Either -*voltage_regime* or -*instances* can be specified (not both).

-**library** *library*_identifier

The -*library* keyword associates an explicit library with the operating voltage.

Default: the operating voltage specified applies to all the libraries used in the design.

-**value** *operating_point*_rvalue

The -*value* keyword allows the voltage point to explicitly be defined using a real number. This value is typically a variable within a library equation.

### 6.6.4 Positional Parameters

None.

### 6.6.5 Examples

```
operating_voltage -voltage_regime low_v -best
operating_voltage -worst -value 5.5
operating_voltage -instances {alu cpu} -best -value 3.45
```

### 6.6.6 Semantics

### 6.6.7 Related Commands

The following commands are related to the *operating_voltage* command:

*operating_point*
*operating_process*
*operating_range*
*operating_temperature*
*units*
*voltage_regime*

## 6.7 temperature_regime



The *temperature_regime* command provides a method to specify a portion of a design within which temperature variations are assumed to be correlated.

### 6.7.1 Usage

**temperature_regime**

[ -**cells** cell_list ] | [ -**instances** instance_list ]
*temperature_regime*_identifier

### 6.7.2 Required Keywords

None.

### 6.7.3 Optional Keywords

-**cells** cell_list

The -*cells* keyword specifies one or more cell names to which the temperature regime should apply.

-**instances** instance_list

The -*instances* keyword specifies one or more instance names to which the temperature regime should apply.

If neither -*cells* or -*instances* is specified, the regime applies to the instance or module specified using the *current_scope* command.

Either -*cells* or -*instances* can be specified, but not both.

### 6.7.4 Positional Parameters

*temperature_regime*_identifier

The *temperature_regime_identifier* specifies the name for the regime. This regime can then be referenced within other operating condition DCDL commands.

### 6.7.5 Examples

```
temperature_regime -cells {ram4x4 rom} mem_tregime
```

### 6.7.6 Semantics

*** Options to select a physical area will be appropriate when the physical constraint domain is added to this specification. ***

### 6.7.7 Related Commands

The following commands are related to the *temperature_regime* command:

*current_scope*
*operating_temperature*
*operating_range*

## 6.8 voltage_regime



The *voltage_regime* command provides a method to specify a portion of a design within which voltage variations are assumed to be correlated

### 6.8.1 Usage

**voltage_regime**

[ -**logical_rail** *logical_rail*_identifier ] | [ -**physical_rail** *physical_rail*_identifier ]
[ -**base_voltage** *voltage*_rvalue ]
[ -**min_voltage** *minimum*_rvalue ] [ -**max_voltage** *maximum*_rvalue ]
[ ( -**cells** cell_list [ -**port** *port*_identifier ] ) ] |
[ ( -**instances** instance_list [ -**pin** *pin*_identifier ] ) ]
*voltage_regime*_identifier

### 6.8.2 Required Keywords

None.

### 6.8.3 Optional Keywords

-**logical_rail** *logical_rail*_identifier

The -*logical_rail* keyword specifies the name of the power rail within the design.

-**physical_rail** *physical_rail*_identifier

The -*physical_rail* keyword specifies the physical net name that represents the power rail within the design.

The physical design system might map the physical rail name into several physical nets (particularly when the physical design is implemented hierarchically).

-**base_voltage** *voltage*_rvalue

The -*base_voltage* keyword specifies a voltage for the power rails of the design, if that value is not defined in the technology library.

-**min_voltage** *minimum*_rvalue

The -*min_voltage* keyword specifies the minimum voltage value of the design power rail for all calculation modes.

-**max_voltage** *maximum*_rvalue

The -*max_voltage* keyword specifies the maximum voltage value of the design power rail for all calculation modes.

-**cells** cell_list [-**port** *port*_identifier]

The -*cells* keyword specifies one or more cell types that should be analyzed using the logical rail name and the connected physical rail name. Usually, the cell names are inferred by an analysis tool, but for libraries that do not contain power pin information, this option might be required.

The optional *-port* keyword specifies the port name that represents the power port on the cell, in case there are more than one.

**-instances** instance_list [**-pin** *pin*_identifier]

The *-instances* keyword specifies one or more instance names that should be analyzed using the logical rail name and the connected physical rail name.

The optional *-pin* keyword specifies the pin name that represents the power port on the instance, in case there are more than one.

If neither *-cells* or *-instances* is specified, the regime applies to the instance or module specified using the *current_scope* command.

Either *-cells* or *-instances* can be specified, but not both.

### 6.8.4 Positional Parameters

*voltage_regime*_identifier

The *voltage_regime_identifier* specifies the name for the regime. This regime can then be referenced within other operating condition DCDL commands.

### 6.8.5 Examples

```
voltage_regime -logical_rail vcc -min_voltage 2.5 low_v
```

### 6.8.6 Semantics

If the instance specified by *-instances* belongs to a hierarchical cell, the constraint applies to the entire hierarchy under the named instance(s), unless a *voltage_regime* command has been applied to a lower-level instance.

Because the *voltage_regime* contains value slots for minimum and maximum voltage, separate commands can define the minimum and maximum values within a file.

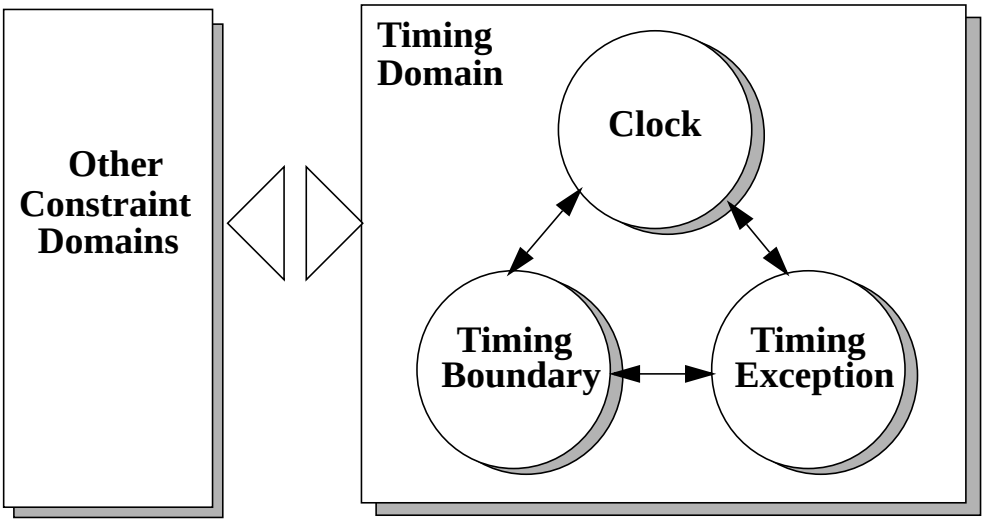
### 6.8.7 Related Commands

The following commands are related to the *voltage_regime* command:

*current_scope*
*operating_process*
*operating_range*
*operating_temperature*
*operating_voltage*

# 7. The Timing Domain

This section describes the theory and the commands representing the timing constraints domain. This domain is divided into 3 categories: clock, timing boundary, and timing exception commands. These categories interact with each other and with other constraint domains.




*** This section will provide the concepts required to understand the timing domain command semantics. Material will be provided by participants, including background materials and graphics. ***

## 7.1 Clock (Synchronous) Theory

### 7.1.1 Clock Domains

### 7.1.2 Clock Roots and Networks

#### 7.1.2.1 Clock and Data Conversion

*** Discussion about swapping clock and data as per IBM request on 1/5/00 ***

#### 7.1.2.2 Clock Gating

### 7.1.3 Ideal Versus Propagated Clocks

#### 7.1.3.1 Insertion Delay Model

*** Explain the different insertion delay types and "left" and "right" (external and internal) concepts. ***

### 7.1.4 Time Relative to Clock Edges

*** Include information about ideal vs effective edges here also. ***

### 7.1.5 Default Cycle Accounting

### 7.1.6 Clock Uncertainties

*** Include relationship to the concept of absolute. ***

#### 7.1.6.1 Jitter

#### 7.1.6.2 Inter-Clock Uncertainty

#### 7.1.6.3 Intra-Clock Tree Skew

#### 7.1.6.4 Target-Based Uncertainty

## 7.2 Timing Boundary Theory

*** required and arrival theory ***

## 7.3 Timing Exception Theory

### 7.3.1 False Paths and Disables

### 7.3.2 Latching

## 7.4 Timing Domain Interactions

*** How the 3 sub-domains within the timing domain work and affect each other and how other domains such as operating conditions and the parasitic domains affect the timing domain will appear here. ***

## 7.5 Common Timing Command Conventions

*** The -early, -late (and relationship to -typ), delay calculation vs analysis, -rise, -fall convention, min/max values, reference to placeholder and wildcarding discussion, defaults, ***

## 7.6 Clock Commands

This section documents all the commands associated with clocking.

### 7.6.1 clock

The *clock* command associates a waveform with actual design pins or ports.

#### 7.6.1.1 Usage

**clock**

-**waveform** *waveform*_identifier ( -**pins** pin_list | -**ports** port_list ) [ -**parent_pin** *pin*_identifier | -**parent_port** *port*_identifier ]

#### 7.6.1.2 Required Keywords

**-waveform** *waveform_identifier*

The -*waveform* keyword specifies an ideal waveform used as a reference point for the clock. This is usually the clock waveform for an external register that drives the pin(s). This waveform name can refer to an ideal waveform specified previously using the *waveform* command or a derived waveform specified previously using the *derived_waveform* command.

**-ports** port_list

The -*ports* keyword specifies the port or ports to which the clock applies. Either –*ports* or –*pins* (or both) must be specified.

**-pins** pin_list

The -*pins* keyword specifies the pin or pins to which the clock applies. Either –*ports* or –*pins* (or both) must be specified.

#### 7.6.1.3 Optional Keywords

-**parent_pin** *pin*_identifier
-**parent_port** *port*_identifier

The -*parent_pin* or -*parent_port* keyword is used to automatically calculate the actual phase shift between the parent clock root and the derived clock root.

#### 7.6.1.4 Positional Parameters

None.

#### 7.6.1.5 Examples

```
clock -waveform master_clk -pins {Clk}
clock -waveform master_clk -ports {clka clkb} -parent_port cina
```

#### 7.6.1.6 Semantics

The pins or ports specified are considered the clock root.

Derived clock functionality is supported through the use of the *-parent_pin* or *-parent_port* keywords. These keywords shall specify pin or port identifiers that define a parent clock root (that was specified using a separate *clock* command).

#### 7.6.1.7 Related Commands

The following commands are related to the *clock* command:

*derived_waveform*
*waveform*

### 7.6.2 clock_arrival_time

The *clock_arrival_time* command defines a window of time in which clock signals will arrive at pins and ports with respect to a specified reference point (waveform).

#### 7.6.2.1 Usage

**clock_arrival_time**

**-waveform** *waveform*_identifier [ **-lead** | **-trail** ] [ **-early** | **-late** ]
**-ports** port_list | **-pins** pin_list *clock_arrival*_time_value_list

#### 7.6.2.2 Required Keywords

**-waveform** *waveform*_identifier

The *-waveform* keyword specifies one ideal waveform used as a reference point for the *clock_arrival_time_value*. This name is usually the clock waveform for an external register that drives the port or pin.

**-ports** port_list

The *-ports* keyword specifies the port or ports to which the clock arrival time applies. Either *–ports* or *–pins* (or both) must be specified.

**-pins** *pin_list*

The *-pins* keyword specifies the pin or pins to which the clock arrival time applies. Either *–ports* or *–pins* (or both) must be specified.

#### 7.6.2.3 Optional Keywords

**-lead** | **-trail**

The *-lead* or *-trail* keywords specify the edge of the ideal waveform used as a reference point for the *clock_arrival_time_value*.

Default: if neither keyword is specified, both are implied.

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies the latest time that the clock signal can arrive at the port or pin (setup time); indicating that the clock signal will not change after the specified time. The *-early* keyword specifies the earliest time that the clock signal can arrive at the port or pin (hold time); indicating that the clock signal will remain stable at the beginning of a clock cycle at least as long as the time specified.

Default: if neither *-early* nor *-late* is specified, both early and late analysis is implied.

#### 7.6.2.4 Positional Parameters

*clock_arrival*_time_value_list

The *clock_arrival_time_value_list* specifies the effective offset (shift) in edge positions from the ideal clock waveform - thus accounting for insertion delay to the "left" of the pin or port. Either a single number or a list of four numbers is required, and the numbers can be negative.

#### 7.6.2.5 Examples

```
clock_arrival_time -waveform sys_clk -lead \
   -early -ports {clk1 clk2} 5.0
```

The preceding example shows the use of several optional keywords.

```
clock_arrival_time -waveform sys_clk \
   -ports {clk3} {1.0 1.3 1.2 1.4}
```

The preceding example shows the use of a list of four values that specify early and late lead and trail values.

#### 7.6.2.6 Semantics

The *clock_arrival_time* command specifies the insertion delay for the portion of a clock network that lies outside (to the "left" of the port or pin) from an implicit reference point (while *clock_delay* specifies the delay to the "right" of the port or pin). This insertion delay affects the relative arrival time of the clock edges at clock roots within the design.

The *clock_arrival_time* specifies a timing window *assertion* while *clock_required_time* specifies a timing window *constraint* (refer to page 23).

One or more *clock_arrival_time* commands may be specified to associate different clock ports or pins with the same or different ideal waveforms.

When any of the options *-early,-late, -lead,* or *-trail* are specified, the *clock_arrival_time_value_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

When *-early, -late*, *-lead*, or *-trail* are not specified, the *clock_arrival_time_value_list* can either be a single value that applies to all time value slots, or a list of four values (one for each time value slot).

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.6.2.7 Related Commands

The following commands are related to the *clock_arrival_time* command:

*clock*
*clock_required_time*
*common_insertion_delay*
*data_arrival_time*
*waveform*

### 7.6.3 clock_delay



The *clock_delay* command specifies the delay characteristics of a clock network or a portion of a hierarchical clock network.

#### 7.6.3.1 Usage

**clock_delay**

**-waveform** *waveform*_identifier | ( **-root_port** *port*_identifier | **-root_pin** *pin*_identifier ) |
( **-leaf** *pin*_identifier ) [ **-rise** | **-fall** ] [ **-early** | **-late** ] *delay*_unsigned_time_value_list

#### 7.6.3.2 Required Keywords

Only one of the following:

**-waveform** *waveform*_identifier

The *-waveform* keyword indicates the ideal waveform to specify default values that affect all clock networks driven by that waveform. This includes virtual clock networks referenced in arrival and required time commands.

**-root_port** *port*_identifier
**-root_pin** *pin*_identifier

The *-root_port* or *-root_pin* keywords identify the name of one hierarchical input or bi-directional port or primitive output pin that drives the clock network.

**-leaf** *pin*_identifier

The *-leaf* keyword identifies an input clock leaf pin (as opposed to a clock root) on a primitive that contains an internal clock network with significant insertion delay.

#### 7.6.3.3 Optional Keywords

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether the clock delay refers to the rising or falling edge of the port, pin, or waveform specified.

Default: if neither keyword is specified, both are implied.

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies that the clock delay is applied to the late arrival time (setup). The *-early* keyword specifies that the clock delay is applied to the early arrival time (hold time).

Default: if neither keyword is specified, both are implied.

#### 7.6.3.4 Positional Parameters

*delay*_unsigned_time_value_list

The *delay_unsigned_time_value_list* specifies the time(s) at which the delay transition(s) occur. Either a single number or a list of four numbers is required, and the numbers must be positive.

#### 7.6.3.5 Examples

```
clock_delay -waveform m_clk {1.2}
```

The preceding example specifies the clock delay on the waveform *m_clk* as being 1.2 for the early rise, early fall, late rise, and late fall time value slots.

```
clock_delay -root_port {clk} -early -rise {2.1}
```

The preceding example specifies the clock delay on the clock root *clk* as being 2.1 for all four time value slots.

```
clock_delay -leaf {int_clk} {1.3 1.4 1.3 1.5}
```

The preceding example specifies the clock delay on the clock leaf *int_clk* for each of the four time value slots.

#### 7.6.3.6 Semantics

The *clock_arrival_time* specifies a delay value with respect to the "left" of the external port or pin, while the *clock_delay* command specifies the delay to the "right" of the port or pin, in terms of the *-root* or *-leaf* options. This is referred to as the elements that can be "seen" in a design. Whereas, the *-waveform* option specifies an ideal waveform, which cannot be "seen" in a design.

When any of the options *-early*, *-late*, *-rise*, or *-fall* are specified, the *delay_unsigned_time_value_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

When *-early*, *-late*, *-rise*, and *-fall* are not specified, the *delay_unsigned_time_value_list* can either be a single value that applies to all four time value slots, or a list of four values (one for each time value slot).

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

Values specified using the *-root* option override those specified using the *-waveform* option.

To describe clock delay external to a block for a leaf port (instead of a pin), *clock_required_time* should be used.

#### 7.6.3.7 Related Commands

The following commands are related to the *clock_delay* command:

*clock_arrival_time*
*clock_mode*
*clock_required_time*
*tree_delay*
*tree_mode*

### 7.6.4 clock_mode

The *clock_mode* command specifies the default analysis for clock network delays.

#### 7.6.4.1 Usage

**clock_mode**

[ **-root_port** port_list | **-root_pin** pin_list ] **-ideal** | **-actual**

#### 7.6.4.2 Required Keywords

Either:

**-ideal**

The *-ideal* keyword specifies whether the ideal insertion delay, skew, and transition times within all clock networks should be used in analysis.

**-actual**

The *-actual* keyword specifies that the actual values should be computed for insertion delay, skew, and transition times.

#### 7.6.4.3 Optional Keywords

**-root_port** port_list
**-root_pin** pin_list

The *-root_port* or *-root_pin* keywords specify a particular port or pin that represents a clock root for which the mode applies.

Default: if *-root_port* or *-root_pin* is not specified, the clock mode applies to all clock roots in the design.

#### 7.6.4.4 Positional Parameters

None.

#### 7.6.4.5 Examples

```
clock_mode -ideal
clock_mode -actual
clock_mode -root_port {clk1} -actual
```

#### 7.6.4.6 Semantics

In general, the analysis mode is explicitly set to *-ideal* prior to inserting the clock networks and *-actual* after clock network insertion. It is expected that the command be explicitly used within the appropriate DCDL file such that tools can choose the correct data.

If *clock_mode* is not specified in a DCDL file, a clock mode of ideal is assumed by the tool.

If both pins and ports need to be specified, separate *clock_mode* commands must be used.

The *clock_mode* command allows inheritance; the specification of *clock_mode* applies to each lower design block level, until another *clock_mode* command is assigned, as per the general precedence rules (refer to page 37).

#### 7.6.4.7 Related Commands

The following commands are related to the *clock_mode* command:

*clock_delay*
*tree_delay*
*tree_mode*

### 7.6.5 clock_required_time

The *clock_required_time* command defines a window of time in which clock signals are insured to arrive at pins and ports with respect to a specified reference point (waveform).

#### 7.6.5.1 Usage

**clock_required_time**

**-waveform** *waveform*_identifier [ **-lead** | **-trail** ] [ **-early** | **-late** ]
**-ports** port_list | **-pins** pin_list *clock_required*_time_value_list

#### 7.6.5.2 Required Keywords

**-waveform** *waveform*_identifier

The *-waveform* keyword specifies one ideal waveform used as a reference point for the *clock_required_time_value*. This name is usually the clock waveform for an external register that drives the port or pin.

**-ports** port_list

The *-ports* keyword specifies the port or ports to which the clock required time applies. Either *–ports* or *–pins* (or both) must be specified.

**-pins** *pin_list*

The *-pins* keyword specifies the pin or pins to which the clock required time applies. Either *–ports* or *–pins* (or both) must be specified.

#### 7.6.5.3 Optional Keywords

**-lead** | **-trail**

The *-lead* or *-trail* keywords specify the edge of the ideal waveform used as a reference point for the *clock_required_time_value*.

Default: if neither keyword is specified, both are implied.

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies the latest time that the clock signal will arrive at the port or pin (setup time); indicating that the clock signal will not change after the specified time. The *-early* keyword specifies the earliest time that the clock signal will arrive at the port or pin (hold time); indicating that the clock signal will remain stable at the beginning of a clock cycle at least as long as the time specified.

Default: if neither *-early* nor *-late* is specified, both early and late analysis is implied.

#### 7.6.5.4 Positional Parameters

*clock_required*_time_value_list

The *clock_required_time_value_list* specifies the time(s) at which the transition(s) occur. Either a single number or a list of four numbers is required, and the numbers can be negative.

#### 7.6.5.5 Examples

```
clock_required_time -waveform sclk -lead \
   -early -ports {clk1 clk2}
```

The preceding example shows the use of several optional keywords.

```
clock_required_time -waveform sclk \
   -ports {clk3} {1.0 1.3 1.2 1.4}
```

The preceding example shows the use of a list of four values that specify early and late lead and trail values.

#### 7.6.5.6 Semantics

The *clock_arrival_time* specifies a timing window *assertion* while *clock_required_time* specifies a timing window *constraint* (refer to page 23).

One or more *clock_required_time* commands may be specified to associate different clock ports or pins with the same or different ideal waveforms.

When any of the options -*early,*-*late,* -*lead,* or -*trail* are specified, the *clock_required_time_value_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

When -*early*, -*late*, -*lead,* or -*trail* are not specified, the *clock_required_time_value_list* can either be a single value that applies to all time value slots, or a list of four values (one for each time value slot).

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.6.5.7 Related Commands

The following commands are related to the *clock_required_time* command:

*clock*
*clock_arrival_time*
*common_insertion_delay*
*data_arrival_time*
*waveform*

### 7.6.6 clock_skew

The *clock_skew* command specifies skew characteristics of a clock network (or network portion).

#### 7.6.6.1 Usage

**clock_skew**

( -**root_port** *port*_identifier | -**root_pin** *pin*_identifier ) [ -**rise** | -**fall** ] [ -**early** | -**late** ]
*skew*_unsigned_time_value_list

#### 7.6.6.2 Required Keywords

-**root_port** *port*_identifier or
-**root_pin** *pin*_identifier

The -*root_port* or -*root_pin* keywords identify the name of one hierarchical input or bi-directional port or primitive output pin that drives the clock network.

#### 7.6.6.3 Optional Keywords

**-rise** | **-fall**

The -*rise* or -*fall* keywords indicate whether the clock skew refers to the rising or falling edge of the port or pin specified.

Default: if neither keyword is specified, both are implied.

**-early** | **-late**

The -*early* or -*late* keywords specify the type of analysis. The -*late* keyword specifies that the clock skew is applied to the late arrival time (setup). The -*early* keyword specifies that the clock skew is applied to the early arrival time (hold time).

Default: if neither keyword is specified, both are implied.

#### 7.6.6.4 Positional Parameters

*skew*_unsigned_time_value_list

The *skew_unsigned_time_value_list* specifies the maximum difference in clock insertion delays to any leaf pins implied by the specified root port or pin. Either a single number or a list of four numbers is required, and the numbers must be positive.

#### 7.6.6.5 Examples

```
clock_skew -root_port {clk} -early -rise {3.4}
```

The preceding example specifies the clock skew on the clock root *clk* as being 3.4 for all four time value slots.

#### 7.6.6.6 Semantics

When any of the options *-early*, *-late*, *-rise*, or *-fall* are specified, the *skew_unsigned_time_value_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

When *-early*, *-late*, *-rise*, and *-fall* are not specified, the *skew_unsigned_time_value_list* can either be a single value that applies to all four time value slots, or a list of four values (one for each time value slot).

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.6.6.7 Related Commands

The following commands are related to the *clock_skew* command:

*clock*
*clock_arrival_time*
*clock_mode*
*clock_required_time*
*tree_delay*
*tree_mode*

### 7.6.7 clock_uncertainty



The *clock_uncertainty* command specifies the worst-case uncertainty between two clock distribution networks.

#### 7.6.7.1 Usage

**clock_uncertainty**

[ **-from** *root_waveform*_identifier ] [ **-to** *target_waveform*_identifier ] [ **-from_edge rise** | **fall** ] [ **-to_edge rise** | **fall** ] [ **-early** | **-late** ] [ **-absolute** | **-increment** ] [ **-ideal** | **-actual** ] **{** *uncertainty*_rsvalue **}**

#### 7.6.7.2 Required Keywords

Either of the following keywords (or both):

**-from** *root_waveform*_identifier

The *-from* keyword specifies default values that affect all source clock networks driven by the specified waveform. This includes virtual clock networks referenced in arrival constraints.

**-to** *target_waveform*_identifier

The *-to* keyword specifies default values that affect all target clock networks driven by the specified waveform. This includes virtual clock networks reference in required time constraints.

#### 7.6.7.3 Optional Keywords

**-from_edge rise** | **fall**

The *-from_edge* keyword indicates the edge (rising or falling) of the clock that launches data from the source register of a path between the source and target clock networks.

**-to_edge rise** | **fall**

The *-to_edge* keyword indicates the edge (rising or falling) of the clock that captures data at the target register of a path between the source and target clock networks.

**-early** | **-late**

The *-early* or *-late* keywords specify whether the uncertainty value is with respect to the hold or setup checks.

Default: if no option is selected, both early and late analysis is applied.

**-absolute** | **-increment**

The *-absolute* and *-increment* keywords indicate the calculation mode for the uncertainty value. The *-absolute* option indicates that the uncertainty value represents the entire skew between the clocks. The *-increment* option indicates that the uncertainty value is added to any skew calculated between the clocks.

Default: *-increment*

**-ideal** | **-actual**

The *-ideal* and *-actual* keywords indicate the propagation mode for the uncertainty value. The *-ideal* option indicates that the uncertainty value applies when either the clock network for the source register or clock network for the target register is analyzed in ideal mode. The *-actual* option indicates that the uncertainty value applies when the clock networks to both the source and target registers are analyzed in actual mode.

Default: *-ideal*

#### 7.6.7.4 Positional Parameters

**{** *uncertainty*_rsvalue **}**

The *uncertainty_rsvalue* is the positive or negative real number list that indicates the uncertainty for early and late analysis (1 or 2 values must be present in the list). The value(s) can override or add to any skew inherent in the harmonic relationship between the edges in the source and target clock waveforms; and any skew introduced by the insertion delays in the clock networks to the source and target registers.

If one value is present, this value is used for both early and late analysis. If two values are present, the first value is used for early analysis and the second is used for late analysis.

#### 7.6.7.5 Examples

```
clock_uncertainty -from main_clk -absolute -ideal {.02 .05}
```

#### 7.6.7.6 Semantics

The *clock_uncertainty* command can be used to increase optimism by using negative values for the *uncertainty_rsvalue*. This can be useful when the design has paths starting in one clock domain and ending in another and when there are no synchronous relationships between two clocks.

#### 7.6.7.7 Related Commands

The following commands are related to the *clock_uncertainty* command:

*clock*
*clock_delay*
*clock_mode*
*target_uncertainty*

### 7.6.8 common_insertion_delay

The *common_insertion_delay* command specifies the portion of the external insertion delay that is common to two clock roots.

#### 7.6.8.1 Usage

**common_insertion_delay**

( **-from_port** *clock_port*_identifier | **-from_pin** *clock_pin*_identifier )
( **-to_port** *clock_port*_identifier | **-to_pin** *clock_pin*_identifier ) [ **-rise** | **-fall** ] [ **-early** | **-late** ]
*insertion*_rvalue_list

#### 7.6.8.2 Required Keywords

**-from_port** *clock_port*_identifier
**-from_pin** *clock_pin*_identifier

The *-from_port* or *-from_pin* keywords specify the source root clock port or pin.

**-to_port** *clock_port*_identifier
**-to_pin** *clock_pin*_identifier

The *-to_port* or *-to_pin* keywords specify the target root clock port or pin.

#### 7.6.8.3 Optional Keywords

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether the insertion delay refers to the rising or falling edge of the port or pin specified.

Default: if neither keyword is specified, both are implied.

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies that the delay is applied to the late arrival time (setup). The *-early* keyword specifies that the delay is applied to the early arrival time (hold time).

Default: if neither keyword is specified, both are implied.

#### 7.6.8.4 Positional Parameters

*insertion*_rvalue_list

The *insertion_rvalue_list* specifies a delay value that represents a portion of the clock network as specified by the "from" and "to" port or pins. Either a single number or a list of four numbers is required and positive numbers shall be used.

#### 7.6.8.5 Examples

```
common_insertion_delay -from_port clk_main -to_pin ckl_div {1 2 3 4}
common_insertion_delay -from_port clk1 -to_port clk2 -early -rise \
   {1.2}
```

#### 7.6.8.6 Semantics

The *common_insertion_delay* command is commonly used for path pessimism removal techniques.
*** Mark to supply better description ***
The command allows the specification of how much of the clock path represents insertion delay such that a tool does not have to analyze an entire clock tree network. This is useful because In many cases, the tool might not have visibility of the entire network. This is particularly for board-level analysis.

When any of the options *-early*, *-late*, *-rise*, or *-fall* are specified, the *insertion_rvalue_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

When *-early*, *-late*, *-rise*, and *-fall* are not specified, the *insertion_rvalue_list* can either be a single value that applies to all four time value slots, or a list of four values (one for each time value slot).

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.6.8.7 Related Commands

The following commands are related to the *common_insertion_delay* command:

*clock*
*clock_arrival*
*clock_delay*

### 7.6.9 derived_waveform

The *derived_waveform* command specifies a new waveform, derived from an existing waveform.

#### 7.6.9.1 Usage

**derived_waveform**

**-waveform** *parent_waveform*_identifier **-name** *derived_waveform*_identifier [ **-inverted** ]
[ **-phase {** *offset_shift*_rsvalue_list **}** ]
( [ **-multiplier** *mult*_unsigned_number ] [ **-divisor** *divisor*_unsigned_number ] ) |
[ **-derived_edges {** l*ead_edge*_unsigned_number *trail_edge*_unsigned_number **}** ]
[ **-lead_jitter** jitter_value | **-trail_jitter** jitter_value ]

jitter_value ::= **{** *left*_unsigned_time_value *right*_unsigned_time_value **}** |
**{** *offset*_unsigned_time_value **}** [ **-increment** | **-absolute** ]

#### 7.6.9.2 Required Keywords

**-waveform** *parent_waveform*_identifier

The *-waveform* keyword indicates the name of the waveform from which the new waveform is derived. This waveform is considered the parent.

**-name** *derived_waveform*_identifier

The *-name* keyword specifies a name for the new, derived waveform.

#### 7.6.9.3 Optional Keywords

**-inverted**

The *-inverted* option changes the lead and trail values of the existing waveform edges for the new waveform. The lead edge is falling, and the trail edge is rising.

Default: lead edge is rising and the trail edge is falling.

**-phase {** *offset_shift*_rsvalue_list **}**

The *-phase* keyword specifies an offset (phase shift) from each edge of the parent waveform. The value can be negative. There must be 1 or 2 values only - corresponding to the leading and/or trailing edges from the parent waveform edge list.

Default: 0

**-multiplier** *mult*_unsigned_number

The *-multiplier* keyword specifies a positive, integer multiplier value relative to the parent waveform frequency.

Default: 1

-**divisor** *divisor*_unsigned_number

The -*divisor* keyword specifies a positive, integer divisor value relative to the parent waveform frequency.

Default: 1

-**derived_edges** derived_edge_list

The -*derived_edges* keyword specifies that the edge positions in the derived waveform are obtained by selecting particular rising and falling edges of the parent waveform. Exactly 2 values are required.

Default: if -*multiplier* and/or -*divisor* are used, -*derived_edges* cannot be used. If none of these keywords are used, the derived waveform is a copy of the parent that can be inverted using -*inverted* or shifted using -*phase*.

-**lead_jitter {** *left*_unsigned_time_value *right*_unsigned_time_value **}** | **{** *offset*_unsigned_time_value **}**
-**trail_jitter {** *left*_unsigned_time_value *right*_unsigned_time_value **}** | **{** *offset*_unsigned_time_value **}**

The -*lead_jitter* and -*trail_jitter* keywords describe the maximum deviation (across all possible clock cycles) from the lead and trail edge positions. This deviation is expressed as a time offset value. A left and a right offset can be specified. If only one value is specified, that value applies to both the left and right offset. The time values must be positive and a placeholder can be specified.

Default: 0.

-**increment** | -**absolute**

The -*increment* keyword specifies that the -*lead_jitter* and/or -*trail_jitter* values are to be added to the jitter values from the parent waveform. The -*absolute* keyword specifies that the -*lead_jitter* and/or -*trail_jitter* values represent the actual jitter value for the derived waveform, replacing any jitter values specified in the parent waveform.

The -*increment* and -*absolute* keywords are ignored if -*lead_jitter* and/or -*trail_jitter* are not specified.

Default: -*absolute* (if -*lead_jitter* and/or -*trail_jitter* are specified).

#### 7.6.9.4 Positional Parameters

None.

#### 7.6.9.5 Examples

```
derived_waveform -waveform mstr -name half_wave -divisor 2
```

The preceding example shows the specification of a waveform that has been divided in half.

#### 7.6.9.6 Semantics

The *derived_waveform* command is a method to create derived waveforms based on a waveform specifed by the waveform command. The -*multiplier* and -*divisor* keywords provide an easy method to describe clock dividers or multipliers, or more complex waveforms using the -*derived_edges* keyword:

Both the *-multiplier* and *-divisor* options can be specified to indicate a rational ratio between the parent and derived frequencies.

The *-multiplier* or *-divisor* options cannot be specified when *-derived_edges* is used.

The *-phase* option is applied to the waveform after *-multiplier* and/or *-divisor* are applied.

The *-derived_edges* keyword is used when a simple divider or multiplier waveform is not sufficient. Using this keyword allows specifying one rising and one falling edge of the parent waveform to be used in deriving a new waveform. The lead and trail edge order is affected by the *-inverted* keyword.

Figure 3-6 shows the derived edge concept.

```
-derived_edges { -posedge 1 -negedge 6 }
```

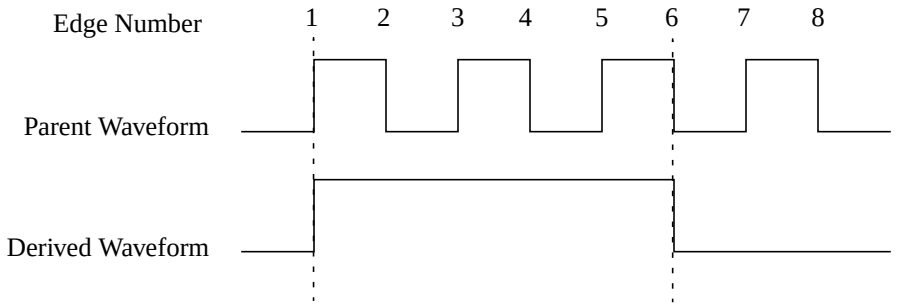



**Figure 3-6 Derived Edges Concept**

*** Does the phase shift affect the ideal waveform or not? ***

The derived waveform belongs to the same domain as the parent waveform (either an implied domain or an explicit domain as specified using the *-domain* keyword in *waveform* command). While the parent waveform can also be a derived waveform, the domain is estabilished by the top parent waveform as established by the *waveform* command.

### 7.6.9.7 Related Commands

The following commands are related to the *derived_waveform* command:

*clock*
*waveform*

### 7.6.10 target_uncertainty

The *target_uncertainty* command specifies the worst-case uncertainty between the clock edge at a target register and the clock edges at any source register.

#### 7.6.10.1 Usage

**target_uncertainty**

-**port** *clock_root*_identifier | ( -**pin** *clock_leaf*_identifier | *clock_root*_identifier ) | -**instance** *instance*_identifier [ -**early** | -**late** ] [ -**absolute** | -**increment** ] [ -**ideal** | -**actual** ] *uncertainty*_rsvalue

#### 7.6.10.2 Required Keywords

-**port** *clock_root*_identifier | ( -**pin** *clock_leaf*_identifier | *clock_root*_identifier ) |
-**instance** *instance*_identifier

An explicit target is specified using -*pin clock_leaf_identifier*. Implied targets are specified using -*port*, -*pin clock_root_identifier*, or -*instance*. The clock root is specified by providing the name of the port or pin that is the root of a clock distribution network. The clock leaf is specified by providing the name of the clock input port or pin on a target register or an instance. If -*instance* is specified, all the clock pins on that instance are implied targets.

#### 7.6.10.3 Optional Keywords

-**early** | -**late**

The -*early* or -*late* keywords specify whether the uncertainty value is with respect to the hold or setup checks.

Default: if no option is selected, both early and late analysis is applied.

-**absolute** | -**increment**

The -*absolute* and -*increment* keywords indicate the calculation mode for the uncertainty value. The -*absolute* option indicates that the uncertainty value represents the entire skew between the target clock and any source clock. The -*increment* option indicates that the uncertainty value is added to any skew calculated between the target clock and any source clock.

Default: -*increment*

-**ideal** | -**actual**

The -*ideal* and -*actual* keywords indicate the propagation mode for the uncertainty value. The -*ideal* option indicates that the uncertainty value applies when the source or target register is analyzed in ideal mode. The -*actual* option indicates that the uncertainty value applies when the clock network to both the source and target registers are analyzed in actual mode.

Default: -*ideal*

#### 7.6.10.4 Positional Parameters

*uncertainty*_rsvalue

The *uncertainty_rsvalue* is the positive or negative real number that indicates the uncertainty. This value can override or add to any skew inherent in the harmonic relationship between the edges in the source and target clock waveforms; and any skew introduced by the insertion delays in the clock networks to the source and target registers.

#### 7.6.10.5 Examples

```
target_uncertainty -port {clk1} 0.124
target_uncertainty -instance {U1} -absolute -actual 0.214
```

#### 7.6.10.6 Semantics

All paths to any explict or implied targets are affected by the *target_uncertainty* command. By default, the *uncertainty_rsvalue* is added to any explicitly-computed uncertainty skews. This is overridden by the use of *-absolute* keyword.

The *-ideal* and *-actual* keywords control when the *uncertainty_rsvalue* is applied to the specified paths based on when the paths are driven in ideal or actual mode as specified by the *clock_mode* command.

#### 7.6.10.7 Related Commands

The following commands are related to the *target_uncertainty* command:

*clock*
*clock_delay*
*clock_mode*
*clock_uncertainty*

### 7.6.11 waveform

The *waveform* command specifies an abstract, ideal waveform that can be used in other DCDL commands as a reference.

#### 7.6.11.1 Usage

**waveform**

**-name** *waveform*_identifier [ **-period** *period*_rvalue ] [ **-edges { ***lead*_rsvalue *trail*_rsvalue **}** ]
[ **-lead_jitter { ***left*_unsigned_time_value *right*_unsigned_time_value **}** |
**{** *offset*_unsigned_time_value **}** ]
[ **-trail_jitter { ***left*_unsigned_time_value *right*_unsigned_time_value **}** |
**{** *offset*_unsigned_time_value **}** ]
[ **-inverted** ] [ **-domain** *domain*_identifier ]

#### 7.6.11.2 Required Keywords

**-name** *waveform*_identifier

The *-name* keyword specifies the name for the abstract waveform.

#### 7.6.11.3 Optional Keywords

Either *-period* and/or *-edges* keywords must be specified:

**-period**

The *-period* keyword specifies the waveform period using a positive, real number.

Default: 50% duty-cycle clock derived from the *-edges* specification (if present). For example, if the edges were 0 and 9, the period would be 18. If *-edges* is not specified, the waveform starts at time 0 and it is rising.

**-edges { ***lead*_rsvalue *trail*_rsvalue **}**

The *-edges* keyword defines the waveform's leading and trailing edges. To create an asymmetrical waveform, a list of edge pairs can be specified. The *lead_rsvalue* is the time at which the first transition occurs (rising). The *trail_rsvalue* is the time at which the second transition occurs (falling).

Exactly 2 edge values are required and negative numbers can be specified. The separation between between the first and second edge value must be not be greater than the period value (if specified).

Default: none.

**-lead_jitter { ***left*_unsigned_time_value *right*_unsigned_time_value **}** | **{** *offset*_unsigned_time_value **}**
**-trail_jitter { ***left*_unsigned_time_value *right*_unsigned_time_value **}** | **{** *offset*_unsigned_time_value **}**

The *-lead_jitter* and *-trail_jitter* keywords describe the maximum deviation (across all possible clock cycles) from the lead and trail edge positions. This deviation is expressed as a time offset value. A left and a right offset can be specified. If only one value is specified, that value applies to both the left and right offset. The time values must be positive and a placeholder can be specified.

Default: 0.

**-inverted**

The *-inverted* option changes the lead and trail values of the waveform edges. The lead edge is falling, and the trail edge is rising.

Default: lead edge is rising and the trail edge is falling.

**-domain** *domain*_identifier

The *-domain* option identifies the name of a clock domain. All clocks in the same domain are synchronous with respect to each other.

Default: implicitly created clock domain created by the tool, to which all waveforms belowng that do not specify the *-domain* keyword. All the clocks in the default domain are synchronous with respect to each other.

### 7.6.11.4 Positional Parameters

None.

### 7.6.11.5 Examples

```
waveform -name master_clk -period 18.0 -edges {0 9.0}
```

The preceding example defines a waveform called *master_clk* with a period of 18 ns (units set to ns in a preceding command) with edges at 0 and 9 ns.The *-edges* flag is not strictly required in this example because the default of a 50% duty-cycle and the leading edge at time 0.

```
waveform -name inverted_clk -period 18.0 -edges {1.0 10.0} -inverted
```

The preceding example defines a clock that is inverted. The same effect can be specified without the *-inverted* keyword:

```
waveform -name inverted_clk -period 18.0 -edges {11.0 19.0}
```

The *-inverted* keyword is replaced by adding half the period to each edge on the original example.

```
waveform -name jclk -edges {0 10.0} -lead_jitter {1 -2} \
 -trail_jitter {1.1}
```

The preceding example shows the specification of jitter with respect to the lead and trail edges. The lead jitter percent deviation is 1% positive and -2% negative (the negative sign is optional). The jitter with respect to the trailing edge is 1.1% positive deviation and -1.1% for the negative.

### 7.6.11.6 Semantics

If a 50% duty cycle is not desired, both the *-period* and *-edges* keywords must be specified.

*** The effects of multiple clocks (cycle shift) and the effects on setup/hold, and other analyses will be explored. A general waveform picture and a jitter waveform picture to be added. ***

### 7.6.11.7 Related Commands

The following commands are related to the *waveform* command:

*clock*
*derived_waveform*

## 7.7 Timing Boundary Commands

*** We need a clear definition of these conditions versus parasitic boundary conditions ***

This section provides information about timing boundary commands - commands that describe conditions external to the design that affect timing behavior.

### 7.7.1 data_arrival_time

The *data_arrival_time* command specifies when transitions on data signals arrive at input or bi-directional ports or pins with respect to a specified reference point.

#### 7.7.1.1 Usage

**data_arrival_time**

**-waveform** *waveform*_identifier [ **-target** | **-source** ] [ **-lead** | **-trail** ] [ **-early** | **-late** ] [ **-rise** | **-fall** ]
**-ports** port_list | **-pins** pin_list *arrival*_time_value_list

#### 7.7.1.2 Required Keywords

**-waveform** *waveform*_identifier

The *-waveform* keyword specifies one ideal waveform used as a reference point for the *arrival_time_value*.

**-ports** port_list

The *-ports* keyword specifies the port(s) to which the arrival time applies. Either *–ports* or *–pins* (or both) must be specified.

**-pins** pin_list

The *-pins* keyword specifies the pin(s) to which the arrival time applies. Either *–ports* or *–pins* (or both) must be specified.

#### 7.7.1.3 Optional Keywords

**-target** | **-source**

The *-target* or *-source* keywords specify whether the constraint affects the cycle in which the launching clock edge is assumed to occur (*-source*) or the cycle in which the capturing clock edge is assumed to occur (*-target*). *** check this ***

Default: if neither keyword is specified, *-source* is implied.

**-lead** | **-trail**

The *-lead* or *-trail* keywords specify the edge of the ideal waveform used as a reference point for the *arrival_time_value*.

Default: if neither keyword is specified, *-lead* is implied.

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies the latest time that the signal can arrive at the port or pin (setup time); indicating that the signal will not change after the specified time. The *-early* keyword specifies the earliest time that the signal can arrive at the port or pin (hold time); indicating that the signal will remain stable at the beginning of a clock cycle at least as long as the time specified.

Default: if neither *-early* nor *-late* is specified, both early and late analysis is implied.

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether the data transition offset from the *-waveform* edge refers to the rising or falling edge of the port or pin.

Default: if neither *-rise* nor *-fall* are specified, both transitions are implied.

#### 7.7.1.4 Positional Parameters

*arrival*_time_value_list

The *arrival_time_value_list* specifies the time(s) at which the transition(s) occur. Either a single number or a list of four numbers is required and the numbers can be negative.

#### 7.7.1.5 Examples

```
data_arrival_time -waveform sys_clk -lead \
    -early -rise -ports {in1 in2} 5.0
```

The preceding example shows the use of several optional keywords.

```
data_arrival_time -waveform sys_clk -trail \
   -ports {in3} {1.0 1.1 1.2 1.3}
```

The preceding example shows the use of a list of four values that specify early rise, late rise, early fall, and late fall.

#### 7.7.1.6 Semantics

The *data_arrival_time* command specifies a partial path delay time range at an input or bi-directional port or an internal pin. This value does not include interconnect and loading delays due to the net external to a port. This provides the constraint for the timing allowed for the remaining path internal to the block. If this command is applied to an internal pin, the value shall override any propagated value.

When any of the options *-early*, *-late*, *-rise*, or *-fall* are specified, the *arrival_time_value_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

*** clock defaults and combinational delay, -reset ; Need to address pure combo. circuits. ***
*** Explain the concept of inside and outside the module ***

When *-early*, *-late*, *-rise*, and *-fall* are not specified, the *arrival_time_value_list* can either be a single value that applies to all four time value slots, or a list of four values (one for each time value slot).

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.7.1.7 Related Commands

The following commands are related to the *data_arrival_time* command:

*clock*
*clock_arrival_time*
*waveform*

### 7.7.2 data_required_time

The *data_required_time* command specifies the time required for output or bi-directional ports or pins to be stable with respect to a specified reference point.

#### 7.7.2.1 Usage

**data_required_time**

-**waveform** *waveform*_identifier [ -**target** | -**source** ] [ -**lead** | -**trail** ] [ -**early** | -**late** ] [ -**rise** | -**fall** ]
-**ports** port_list | -**pins** pin_list *required*_time_value_list

#### 7.7.2.2 Required Keywords

-**waveform** *waveform*_identifier

The -*waveform* keyword specifies one ideal waveform used as a reference point for the *required_time_value*.

-**ports** port_list

The -*ports* keyword specifies the port(s) to which the required time applies. Either –*ports* or –*pins* (or both) must be specified.

-**pins** pin_list

The -*pins* keyword specifies the pin(s) to which the required time applies. Either –*ports* or –*pins* (or both) must be specified.

#### 7.7.2.3 Optional Keywords

-**target** | -**source**

The -*target* or -*source* keywords specify whether the constraint affects the cycle in which the launching clock edge is assumed to occur (-*source*) or the cycle in which the capturing clock edge is assumed to occur (-*target*).

Default: if neither keyword is specified, -*target* is implied.

-**lead** | -**trail**

The -*lead* or -*trail* keywords specify the edge of the ideal waveform used as a reference point for the *required_time_value*.

Default: if neither keyword is specified -*lead* is implied.

-**early** | -**late**

The -*early* or -*late* keywords specify the type of analysis. The -*late* keyword indicates that the required time is the latest time that the output port is allowed to change. The -*early* keyword indicates that the required time is the earliest time that the output port is allowed to change.

Default: if neither keyword is specified, -*late* is implied.

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether the data transition offset from the *-waveform* edge refers to the rising or falling edge of the port or pin.

Default: if neither *-rise* nor *-fall* are specified, both transitions are implied.

#### 7.7.2.4 Positional Parameters

*required*_time_value_list

The *required_time_value_list* specifies the time(s) at which the transition(s) occur. Either a single number or a list of four numbers is required, and the numbers can be negative.

#### 7.7.2.5 Examples

```
data_required_time -waveform sys_clk -target -lead \
   -early -rise -ports {out1 out2} 5.0
```

The preceding example shows a list of ports and several of the optional keywords.

```
data_required_time -waveform sys_clk -target -trail \
   -ports {out3} {1.0 1.1 1.2 1.3}
```

The preceding example shows the use of a list of four values that specify early rise, late rise, early fall, and late fall.

#### 7.7.2.6 Semantics

The timing values for the *data_required_time* command are specified in terms of when the valid data signal is required to be stable at output ports and pins. The change at the specified output ports or pins is associated with either the leading or trailing edge of the specified ideal clock.

When any of the options *-early*, *-late*, *-rise*, or *-fall* are specified, the *required_time_value_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

When *-early*, *-late*, *-rise*, and *-fall* are not specified, the *required_time_value_list* can either be a single value that applies to all four time value slots, or a list of four values (one for each time value slot).

In the case where the clock frequency of the source and the target clock differ, DCDL specifies which clock that the cycle adjustments are relative to. Early offsets are positive additions to the source clock edges and late offsets are positive additions to the target clock edges. *** does this need to go into the *data_arrival_time* semantics in some form? ***

There exists an interaction between the use of *-source* | *-target* and *-early* | *-late*: in general, the larger the cycle offset numbers, the looser the constraint. For example, *-source* combined with *-early* results in the early launching clock edge occurring later than the default (making the hold check at the target easier to satisfy). Combining the *-source* with *-late* option results in the late launching clock edge occurring earlier than the default (hold check 0 cycle, setup check 1 cycle). This makes the setup check at the target easier to satisfy.

*** clock defaults and combinational delay, -reset, ***
*** Explain the concept of inside and outside the module ***

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.7.2.7 Related Commands

The following commands are related to the *data_required_time* command:

*clock*
*waveform*

### 7.7.3 departure_time

❍

The *departure_time* command specifies a partial path delay time range beyond a pin or port (not including interconnect and loading due to the external net) for the timing reserved for the remaining path external to the block.

#### 7.7.3.1 Usage

**departure_time**

-**waveform** *waveform*_identifier [ -**early** | -**late** ] [ -**rise** | -**fall** ]
-**ports** port_list | -**pins** pin_list *departure*_time_value_list

#### 7.7.3.2 Required Keywords

**-waveform** *waveform*_identifier

The -*waveform* keyword specifies an existing waveform used as a reference point for the *departure_time_value*. *** the taxonomy states this is optional ***

**-ports** port_list

The -*ports* keyword specifies one or more ports to which the departure time applies. Either –*ports* or –*pins* (or both) must be specified.

**-pins** pin_list

The -*pins* keyword specifies one or more pins to which the departure time applies. Either –*ports* or –*pins* (or both) must be specified.

#### 7.7.3.3 Optional Keywords

**-early** | **-late**

The -*early* or -*late* keywords specify the type of analysis. The -*late* keyword indicates that the departure time is the latest time that the port or pin is allowed to change. The -*early* keyword indicates that the departure time is the earliest time that the port or pin is allowed to change.

Default: if neither keyword is specified, -*late* is implied.

**-rise** | **-fall**

The -*rise* or -*fall* keywords indicate whether the data transition offset from the -*waveform* edge refers to the rising or falling edge of the port or pin.

Default: if neither -*rise* nor -*fall* are specified, both transitions are implied.

#### 7.7.3.4 Positional Parameters

*departure*_time_value_list

The *departure_time_value_list* specifies the time(s) at which the transition(s) occur. Either a single number or a list of four numbers is required, and the numbers can be negative.

#### 7.7.3.5 Examples

```
departure_time -waveform sys_clk \
   -early -rise -ports {out1 out2} 5.0
```

The preceding example shows a list of ports and several of the optional keywords.

```
departure_time -waveform sys_clk \
   -ports {out3} {1.0 1.1 1.2 1.3}
```

The preceding example shows the use of a list of four values that specify early rise, late rise, early fall, and late fall.

#### 7.7.3.6 Semantics

*** Need a clearer definition of this command ***

When any of the options *-early*, *-late*, *-rise*, or *-fall* are specified, the *departure_time_value_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

When *-early*, *-late*, *-rise*, and *-fall* are not specified, the *departure_time_value_list* can either be a single value that applies to all four time value slots, or a list of four values (one for each time value slot).

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.7.3.7 Related Commands

The following commands are related to the *departure_time* command:

*clock*
*external_delay*
*waveform*

### 7.7.4 external_delay

❍

The *external_delay* command specifies purely combinational delays that are external to the design.

#### 7.7.4.1 Usage

**external_delay**

**-waveform** *waveform*_identifier [ **-early** | **-late** ]
**-ports** port_list | **-pins** pin_list
**-rise_range** *rise*_time_value_list **-fall_range** *fall*_time_value_list

#### 7.7.4.2 Required Keywords

**-waveform** *waveform*_identifier

The *-waveform* keyword specifies an existing waveform used as a reference point.

**-ports** port_list

The *-ports* keyword specifies one or more ports to which the external delay applies. Either *–ports* or *–pins* (or both) must be specified.

**-pins** pin_list

The *-pins* keyword specifies one or more pins to which the external delay applies. Either *–ports* or *–pins* (or both) must be specified.

**-rise_range** *rise*_time_value_list

The *-rise_range* keyword specifies the earliest and latest delay of a rising transition. Negative values are allowed. *** 2 values here, correct? ***

**-fall_range** *fall*_time_value_list

The *-fall_range* keyword specifies the earliest and latest delay of a falling transition. Negative values are allowed. *** 2 values here, correct? ***

#### 7.7.4.3 Optional Keywords

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword indicates that the external delay refers to setup analysis. The *-early* keyword indicates that external delay refers to hold analysis.

Default: if neither keyword is specified, *-late* is implied.

*** Need lead, trail ? ***

#### 7.7.4.4 Positional Parameters

None.

#### 7.7.4.5 Examples

#### 7.7.4.6 Semantics

If there exist multiple specifications for the same port or pin, relative to the same waveform edge, only the latest external delay command is in effect. *** Is this not a general topic that should be covered in "Precedence Rules" ? ***

*** Taxonomy seems to differ from the Ambit *set_external_delay.* Also, GCF differentiates external delay, from *path_delay* - do we need *path_delay*?, interaction with required times? ***

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.7.4.7 Related Commands

The following commands are related to the *external_delay* command:

*clock*
*required_time*
*waveform*

### 7.7.5 slew_limit

❍

The *slew_limit* command specifies the maximum slew time allowed for input and output pins or ports.

#### 7.7.5.1 Usage

**slew_limit**

[ **-early** | **-late** ] [ **-rise** | **-fall** ] **-ports** port_list | **-pins** pin_list *slew_limit*_time_value

#### 7.7.5.2 Required Keywords

**-ports** port_list

The *-ports* keyword specifies one or more ports to which the slew limit applies. Either *–ports* or *–pins* (or both) must be specified.

**-pins** pin_list

The *-pins* keyword specifies one or more pins to which the slew limit applies. Either *–ports* or *–pins* (or both) must be specified.

#### 7.7.5.3 Optional Keywords

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies the latest slew time for the port or pin (setup time). The *-early* keyword specifies the earliest slew time for the port or pin (hold time).

Default: if neither *-early* nor *-late* is specified, both early and late analysis is implied.

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether the maximum slew time refers to the rising or falling edge of the port or pin.

Default: if neither *-rise* nor *-fall* are specified, both transitions are implied.

#### 7.7.5.4 Positional Parameters

*slew_limit*_time_value

The *slew_limit_time_value* specifies the maximum slew allowed.

#### 7.7.5.5 Examples

```
slew_limit -ports {addr[*] select} 1.34
```

#### 7.7.5.6 Semantics

*** Ambit and GCF - not in taxonomy, relationship with *slew_time*, ***

### 7.7.5.7 Related Commands

The following commands are related to the *slew_limit* command:

*slew_time*
*units*

### 7.7.6 slew_time

The *slew_time* command specifies the ramp time required for a signal to cross two threshold points for a pin or port. Clock slew can be specified with this command.

#### 7.7.6.1 Usage

**slew_time**

[ -**early** | -**late** ] [ -**rise** | -**fall** ]
-**ports** port_list | -**pins** pin_list
*slew*_time_value_list

#### 7.7.6.2 Required Keywords

**-ports** port_list

The *-ports* keyword specifies the port(s) to which the slew time applies. Either *–ports* or *–pins* (or both) must be specified.

**-pins** pin_list

The *-pins* keyword specifies the pin(s) to which the slew time applies. Either *–ports* or *–pins* (or both) must be specified.

The pin or port identifier can be a clock root - thus describing clock slew.

#### 7.7.6.3 Optional Keywords

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies the latest slew time for the port or pin (setup time). The *-early* keyword specifies the earliest slew time for the port or pin (hold time).

Default: if neither *-early* nor *-late* is specified, both early and late analysis is implied.

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether the slew time refers to the rising or falling edge of the port or pin.

Default: if neither *-rise* nor *-fall* are specified, both transitions are implied.

#### 7.7.6.4 Positional Parameters

*slew*_time_value_list

The *slew_time_value_list* specifies the slew time for the port or pin. Either a single number or a list of four numbers is required, and the numbers can be negative.

#### 7.7.6.5 Examples

```
slew_time -early -rise -ports {in1 in2} 5.0
```

The preceding example shows the use of several optional keywords.

```
slew_time -ports {in3} {1.0 1.1 1.2 1.3}
```

The preceding example shows the use of a list of four values that specify early rise, late rise, early fall, and late fall.

#### 7.7.6.6 Semantics

If the specified pin or port is clock root, in ideal mode - the value specified will be propagated directly through the clock network to all leaf pins. Thus, slew at leaf pins is same as that specified on clock root. In actual mode - the slew value behaves the same as data network.

When any of the options *-early*, *-late*, *-rise*, or *-fall* are specified, the *slew_time_value_list* must be a single value that applies to all of the time value slots implied by the combination of these options.

*** Based on Ambit Strawman using DAC subset syntax - does not appear in taxonomy, -reset ***

When *-early*, *-late*, *-rise*, and *-fall* are not specified, the *slew_time_value_list* can either be a single value that applies to all four time value slots, or a list of four values (one for each time value slot).

Wildcarding is allowed for pin and port references (refer to page 32). Placeholders are allowed for time values (refer to page 37).

#### 7.7.6.7 Related Commands

The following commands are related to the *slew_time* command:

*clock_mode*
*slew_limit*

## 7.8 Timing Exception Commands

*** Need a clear definition of timing exceptions here ***

### 7.8.1 borrow_limit

The *borrow_limit* command specifies the maximum amount of time that can be borrowed from a cycle for level-sensitive latches.

#### 7.8.1.1 Usage

**borrow_limit**

[ -**ports** port_list | -**pins** pin_list | -**waveform** *waveform*_identifier |
-**instances** instance_list ] *borrow_limit*_rvalue

#### 7.8.1.2 Required Keywords

None.

#### 7.8.1.3 Optional Keywords

**-ports** port_list

The *-ports* keyword specifies one or more data input or clock ports of level-sensitive latches to which the borrow limit applies. *** A particular latch or all of them? ***

**-pins** pin_list

The *-pins* keyword specifies one or more data input or clock pins of level-sensitive latches to which the borrow limit applies.

**-waveform** *waveform*_identifier

The *-waveform* keyword specifies a waveform name to which the borrow limit applies. Borrowing is restricted for all data inputs of the level-sensitive latches with respect to the clocks associated with this waveform.

**-instances** instance_list

The *-instances* keyword specifies a level-sensitive latch instance to which borrowing is restricted from all data inputs. Pathnames are allowed.

If no keywords are specified, the borrow limit is applied to all level-sensitive latches within the scope the command (current instance, module, or design).

#### 7.8.1.4 Positional Parameters

*borrow_limit*_rvalue

The *borrow_limit_rvalue* specifies a real number representing the maximum time available from the pulse width.

#### 7.8.1.5 Examples

```
borrow_limit -instances lu_latch 2.5
```

#### 7.8.1.6 Semantics

*** From GCF: The default limit on time borrowing for a given latch is the active pulse width of the clock minus the setup time of the latch. The *borrow_limit* command can only be used to specify a smaller limit; larger limits are ignored. The *borrow_limit_rvalue* applies for all operating points. ***

*** This command could really use some timing diagrams! ***

#### 7.8.1.7 Related Commands

The following commands are related to the *borrow_limit* command:

*current_scope*

### 7.8.2 disable

The *disable* command disables timing arcs in a library cell and all instances of that cell or a particular instance.

#### 7.8.2.1 Usage

**disable**

[ -**library** *library*_identifier ] -**cell** *cell*_identifier | -**instance** *instance*_identifier
[ -**from_port** port_list | -**to_pin** pin_list ] [ -**to_port** port_list | -**to_pin** pin_list ] | [ -**output_arcs** ] | [ -**input_arcs** ] | [ -**internal_arcs** ]

#### 7.8.2.2 Required Keywords

One of the following:

-**cell** *cell*_identifier

The -*cell* keyword specifies the name of the cell to be disabled.

-**instance** *instance*_identifier

The -*instance* keyword specifies the name of the instance to be disabled.

#### 7.8.2.3 Optional Keywords

-**library** *library*_identifier

The -*library* keyword specifies the name of the library that contains the cell to be disabled.

If the library is not specified, the current technology library for the design (as specified using a mechanism within a tool or supporting tool library) is assumed.

Either:

-**from_port** port_list
-**from_pin** pin_list

The -*from_port* or -*from_pin* keywords specify port(s) or pin(s) that start the timing arc. If "from" port(s) or pin(s) are specified without "to" port(s) or pin(s), all paths originating from the "from" port(s) or pin(s) shall be disabled. Specifying both "from" and "to" port(s) or pin(s) identifies a path with a specific start and end point.

-**to_port** port_list
-**to_pin** pin_list

The -*to_port* or -*to_pin* keywords specify port(s) or pins(s) that end the timing arc. If "to" port(s) or pin(s) are specified without "from" port(s) or pin(s), all paths ending at the "to" port(s) or pin(s) shall be disabled. Again, specifying both "from" and "to" port(s) or pin(s) identifies a path with a specific start and end point.

Or:

**-output_arcs**

The *-output_arcs* keyword specifies that only the timing arcs that lead through the outputs of the specified cell are disabled.

**-input_arcs**

The *-input_arcs* keyword specifies that only the timing arcs the originate from the inputs of the specificied cell are disabled.

**-internal_arcs**

The *-internal_arcs* keyword specifies that only the internal timing arcs of the specified cell are disabled.

Any combination of *-output_arcs*, *-input_arcs*, and *-internal_arcs* can be specified. Specifying all 3 options is the same as just specifying *-cell*.

#### 7.8.2.4 -Positional Parameters

None.

#### 7.8.2.5 Examples

```
disable -library acme1.0 -cell and2 -from_port {a} -to_port {q}
disable -library acme1.0 -cell and2
disable -cell and2 -from_port {a}
disable -cell and2 -to_port {q}
disable -library acme1.0 -cell and2 -input_arcs -output_arcs
disable -instance u1
disable -instance u2 -from_pin {r}
```

#### 7.8.2.6 Semantics

If only the *-cell* keyword is specified, all the timing arcs in the specified cell are disabled. Likewise, if only the *-instance* keyword is specified, all the timing arcs in the specified instance are disabled.

If *-cell* is specified, the *-from_port* and/or *-to_port* keywords must be used. If *-instance* is specified, the *-from_pin* and/or *-to_pin* keywords must be used..

*** Did not find in Ambit strawman. GCF (disable) and GCL (*set_broken_arc*) have similar concepts. Allow ports also? Pathnames and pathname rules. ***

*** We could add more to this command: through, nets, instances, rise, fall, early, late, etc. , ***

*** Can there be multiple from and to in a single *disable* command? How about pin and port combinations on from and to? ***

#### 7.8.2.7 Related Commands

The following commands are related to the *disable* command:

*false_path*

### 7.8.3 false_path

The *false_path* command identifies timing paths that should not be analyzed.

#### 7.8.3.1 Usage

**false_path**

[ -**early** | -**late** ] [ -**rise** | -**fall** ] path_options

path_options ::= ( -**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}** ) |
( -**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}** ) |
( { -**through_port** | -**through_pin** | -**through_instance** | -**through_net {** object_identifier **}** } )

#### 7.8.3.2 Required Keywords

Any single or combination of the following keywords:

-**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}**

The "from" keywords specify a list of design objects that start the false path. If a "from" keyword is used without a "to" keyword, all paths originating from the "from" object shall be false paths. Specifying both "from" and "to" keywords identifies a path with a specific start and end point. Multiple "from" design *object_identifier* constructs can be specified.

-**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}**

The "to" keywords specify a list of design objects that end the false path. If a "to" keyword is used without a "from" keyword, all paths ending at the "to" object shall be false paths. Again, specifying both "from" and "to" keywords identifies a path with a specific start and end point. Multiple "to" design *object_identifier* constructs can be specified.

{ -**through_port** | -**through_pin** | -**through_instance** | -**through_net {** object_identifier **}** }

The "through" keywords specify a list of design objects through which a path must flow in order to be considered a false path. Multiple "through" design *object_identifier* constructs can be specified.

Multiple through options can be used in a single *false_path* command.

For the "to" and "from" keyword values, a pin, port, instance, or waveform object identifier can be specified. For "through" keyword values, a pin, port, instance, or net object identifier can be specified. The *object_identifier* can use wildcarding and pathnames (refer to page 32).

*** Are all of these keywords overridden by the *waveform* command? ***

#### 7.8.3.3 Optional Keywords

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis (hold or setup). These options only apply to the "to" objects when specified. Otherwise, the option applies to the "from" objects. *** What about *through*? ***

Default: if neither keyword is specified, both analyses are implied.

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether *** what? ** with respect to the rising or falling edge of the data signal.

These options only apply to the "to" objects when specified. Otherwise, the option applies to the "from" objects. *** What about *through*? ***

Default: if neither *-rise* nor *-fall* are specified, both transitions are implied.

#### 7.8.3.4 Positional Parameters

None.

#### 7.8.3.5 Examples

```
false_path -from_port {test_in} -to_port {"/fpu/tout*" /alu/test_o"}
false_path -from_port {test} -to_pin {test_con}
false_path -from_instance {u1}
false_path -early -rise -from_pin {in[*] in2[*] -through_pin {a}
false_path -to {MemWrite}
false_path -from_port {a} -through_instance{u2} -to_port {in1}
```

#### 7.8.3.6 Semantics

If a waveform object is specified, all registers triggered by the specified waveform are considered the from/to points of the path.

*** Need to add something about the -*through* model - ANDed or ORed, waveform to waveform specifications ****

#### 7.8.3.7 Related Commands

The following commands are related to the *false_path* command:

*multi_cycle_path*
*waveform*

### 7.8.4 multi_cycle_path

The *multi_cycle_path* command identifies timing paths that span over multiple clock cycles.

#### 7.8.4.1 Usage

**multi_cycle_path**

[ -**target** | -**source** | -**waveform** *waveform*_identifier ] [ -**early** | -**late** ] [ -**rise** | -**fall** ]
path_options **{** cycle_number **}**

path_options ::= ( -**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}** ) |
( -**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}** ) |
( { -**through_port** | -**through_pin** | -**through_instance** | -**through_net {** object_identifier **}** } )

cycle_number ::= [ sign ] real | [ sign ] unsigned_number |
**{** [ sign ] real [ sign] real **} |**
**{** [ sign ] unsigned_number [ sign ] unsigned_number **}**
**{** [ sign ] real [ sign ] unsigned_number **} | {** [ sign ] unsigned_number [ sign ] real **} |**
placeholder

#### 7.8.4.2 Required Keywords

Any single or combination of the following keywords:

-**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}**

The "from" keywords specify a list of design objects that start the multi-cycle path. If a "from" keyword is used without a "to" keyword, all paths originating from the "from" object shall be multi-cycle paths. Specifying both "from" and "to" keywords identifies a path with a specific start and end point. Multiple "from" design *object_identifier* constructs can be specified.

-**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}**

The "to" keywords specify a list of design objects that end the multi-cycle path. If a "to" keyword is used without a "from" keyword, all paths ending at the "to" object shall be multi-cycle paths. Again, specifying both "from" and "to" keywords identifies a path with a specific start and end point. Multiple "to" design *object_identifier* constructs can be specified.

{ -**through_port** | -**through_pin** | -**through_instance** | -**through_net {** object_identifier **}** }

The "through" keywords specify a list of design objects through which a path must flow in order to be considered a multi-cycle path. Multiple "through" design *object_identifier* constructs can be specified.

Multiple through options can be used in a single *multi_cycle_path* command.

For the "to" and "from" keyword values, a pin, port, instance, or waveform object identifier can be specified. For "through" keyword values, a pin, port, instance, or net object identifier can be specified. The *object_identifier* can use wildcarding and pathnames (refer to page 32).

*** Are all of these keywords overridden by the *waveform* command? ***

#### 7.8.4.3 Optional Keywords

**-target** | **-source**

The *-target* or *-source* keywords specify whether the constraint affects the cycle in which the launching clock edge is assumed to occur (*-source*) or the cycle in which the capturing clock edge is assumed to occur (*-target*).

Default: if neither keyword is specified, *-source* is implied.

**-waveform** *waveform_*identifier

If *-waveform* is specified, all registers triggered by the specified waveform are considered the from/to of the path.

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword indicates that cycles are with respect to the late time of the data signal (setup). The *-early* keyword indicates that cycles are with respect to the early time of the data signal (hold).

These options only apply to the "to" objects when specified. Otherwise, the option applies to the "from" objects. *** what about through? ***

Default: if neither keyword is specified, both analyses are implied.

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether the cycles apply with respect to the rising or falling edge of the data signal.

These options only apply to the "to" objects when specified. Otherwise, the option applies to the "from" objects. *** What about *through*? ***

Default: if neither *-rise* nor *-fall* are specified, both transitions are implied.

#### 7.8.4.4 Positional Parameters

**{** cycle_number **}**

The *cycle_number* indicates the number of cycles for the path specified by "to", "from", and "through" keywords. This value is a real number and it can be negative.

When either *-early* or *-late* are specified, the *cycle_number* must be a single value and it applies to the value slot implied by that option. When neither *-early* nor *-late* is specified, the *cycle_number* can either be a single value that applies to both value slots, or a list of two values, one for each slot (early and late).

The *cycle_number* can use placeholders (refer to page 37)

The default hold check uses 0 cycles and the default setup check uses 1 cycle.

#### 7.8.4.5 Examples

```
multi_cycle_path -source -to_port {MemWrite} {1 2}
```

The preceding example specifies that the hold check should use 1 cycle and the setup check should use 2 cycles.

### 7.8.4.6 Semantics

The *cycle_number* represents the total number of cycles used for the setup or hold check - not the number of additional cycles.

In the case where the clock frequency of the source and the target clock differ, DCDL specifies which clock that the cycle adjustments are relative to. Early offsets are positive additions to the source clock edges and late offsets are positive additions to the target clock edges.

There exists an interaction between the use of -*source* | -*target* and -*early* | -*late*: in general, the larger the cycle numbers, the looser the constraint. For example, -*source* combined with -*early* results in the early launching clock edge occurring later than the default (making the hold check at the target easier to satisfy). Combining the -*source* with -*late* option results in the late launching clock edge occurring earlier than the default (making the setup check at the target easier to satisfy. *** what about -*waveform?* ***

*** Need to add something about the *through* model - ANDed or ORed ****

### 7.8.4.7 Related Commands

The following commands are related to the *multi_cycle_path* command:

### 7.8.5 tree_delay

The *tree_delay* command constrains the timing characteristics of a general buffer tree.

#### 7.8.5.1 Usage

**tree_delay**

-**root_port** *port*_identifier | -**root_pin** *pin*_identifier
[ -**ideal** | -**actual** ] [ -**explict_leaf** pin_list ]
[ -**default_insertion** *delay*_rvalue ] [ -**explicit_insertion** *delay*_rvalue ]
[ -**internal_insertion** *delay*_rvalue ]
[ -**default_skew** *skew*_rvalue ] [ -**default_transition** *time*_rvalue ]
[ -**explicit_transition** *time*_rvalue ]

#### 7.8.5.2 Required Keywords

-**root_port** *port*_identifier
-**root_pin** *pin*_identifier

The -*root_port* or -*root_pin* keywords identify the name of one hierarchical port or primitive output pin that drives the buffer tree.

#### 7.8.5.3 Optional Keywords

-**ideal** | -**actual**

The -*ideal* keyword specifies whether the ideal insertion delay, skew, and transition times within the buffer tree should be used in analysis. The -*actual* keyword specifies that the actual values should be computed for insertion delay, skew, and transition times.

Default: *** ? ***

-**explicit_leaf** pin_list

The -*explicit_leaf* keyword overrides the default rules to specify that pins which would otherwise lie within the default buffer tree should be treated as leaf pins. This has the effect of grouping a set of default leaf pins in order to override the default insertion delay or slew constraints.

-**default_insertion** *delay*_rvalue

The -*default_insertion* keyword specifies the nominal cumulative delay from the root to the default leaf pins and the default value for explicit leaf pins.

-**explicit_insertion** *delay*_rvalue

The -*explicit_insertion* keyword overrides the default insertion delay by specifying a different insertion delay for a group of explicit leaf pins.

-**internal_insertion** *delay*_rvalue

The -*internal_insertion* keyword specifies additional insertion delay that lies beyond an explicit leaf pin, but that should be included in the calculated insertion delay to the leaf.

-**default_skew** *skew*_rvalue

The -*default_skew* keyword indicates the range of differences in insertion delay allowed between any pair of leaf pins, except for the leaf pins excluded by the -*data_leaf* option.

-**default_transition** *time*_rvalue

The -*default_transition* keyword specifies the range of transition (ramp) times allowed at each default leaf pin and the default value for explicit leaf pins.

-**explicit_transition** *time*_rvalue

The -*explicit_transition* keyword specifies the range of transition times allowed at a group of explicit leaf pins.

#### 7.8.5.4 Positional Parameters

None.

#### 7.8.5.5 Examples

#### 7.8.5.6 Semantics

*** This command will be brought in line with the clock_delay (and required times) methodology. The options that mention ranges - do you need 2 values?***

In general, the analysis mode is set to -*ideal* prior to inserting the clock networks and -*actual* after clock network insertion.

The -*internal_insertion* keyword should be used when the explicit leaf pin is an input on a model of a hierarchical module in order to represent the internal insertion delay from the input port to the real leaf pins within the module.

#### 7.8.5.7 Related Commands

The following commands are related to the *tree_delay* command:

*clock_mode*
*clock_delay*
*tree_mode*

### 7.8.6 tree_mode

❍

The *tree_mode* command specifies the default analysis for buffer tree delays.

#### 7.8.6.1 Usage

**tree_mode**

**-ideal** | **-actual**

#### 7.8.6.2 Required Keywords

**-ideal** | **-actual**

The -*ideal* keyword specifies whether the ideal insertion delay, skew, and transition times within all buffer trees described by the *tree_delay* command should be used in analysis. The -*actual* keyword specifies that the actual values should be computed for insertion delay, skew, and transition times.

Default: *** ? ***

*** Do we need the same options as added to clock_mode? (root pin or port) ***

#### 7.8.6.3 Optional Keywords

None.

#### 7.8.6.4 Positional Parameters

None.

#### 7.8.6.5 Examples

```
tree_mode -ideal
tree_mode -actual
```

#### 7.8.6.6 Semantics

In general, the analysis mode is set to -*ideal* prior to inserting the buffer trees and -*actual* after buffer tree insertion. Refer to page 51 for information about modes.

The tree mode is overridden by the *tree_delay* command.

#### 7.8.6.7 Related Commands

The following commands are related to the *tree_mode* command:

*clock_mode*
*tree_delay*

## 7.9 SDF Mapping

*** GCF (*sdf_delay_conditions*): early/late value to min/typ/max SDF mapping technique - not sure if this goes here ***

# 8. The Parasitic Boundary Domain

*** Need a clear definition of this domain versus the timing boundary domain. Also, it looks like you could use these commands along with OLA to fully define electrical rule checks. Should this be an exploration area? ***

*** Included “raw” is some of the text from the taxonomy about parasitics for review purposes: ***

---

The scope of this section includes the following coverage and points of view:

> This section defines the boundary conditions that surround any particular net or fragment of a set within a block within a design hierarchy. The boundary conditions therefore define the specific environment for defining (prescribing) and/or constraining the timing of signals being carried by a net.
>
> Electrical, magnetic, optical, thermal, (and possibly other forms of energy), material, and mechanical properties that drive (source, transmit), load (sink, receive, terminate), surround, or otherwise affect the timing of signals carried by a net.
>
> The timing of signals on the net may be affected directly or indirectly by the above forms of energy, mechanics, and material properties. These parasitics may therefore include effects of coupling and ‘over the cell’ routing.

Presumed, typical, or actual characteristics of the signals being carried (propagated) to the net through some port or via. Both static, steady state and transient signal characteristics may be prescribed.

> Presumed, typical, or actual characteristics of the environment conditions, including, but not limited to, power, temperature, particle density, noise. Both static, steady state and transient characteristics may be prescribed.
>
> Hierarchical structure of that portion of the design surrounding the net:
>
> > Target net is assumed to be contained within one block of an overall design hierarchy. The target net may also possibly be further contained within one layer or strata of the physically implemented design. Where a net consists of more than one net fragment (portion, net segment, subnet), the fragments may be connected by a via or other intra-layer conducting mechanism.
> >
> > Hierarchical driving boundary conditions: The signals carried by a net may pass from an upper level portion of the design hierarchy to the target net within a block by flowing from a receiving instance port (occurrence port in the fully elaborated hierarchical design) ‘down’ to the containing block’s interface definition port. Signals may also be carried from a lower level portion (sub-block or block instance) of the design hierarchy to the net by flowing ‘up’ from a sub-block’s driving instance port.
> >
> > Hierarchical receiving boundary conditions: The signals are carried away from a net to some upper level portion of the design hierarchy from the target net within a block by flowing to the containing block’s receiving interface definition port ‘up’ to the above portion of the design hierarchy. Signals may also be carried away from a net to some lower level portion of the design hierarchy by flowing ‘down’ to a sub-block’s receiving instance port.

---

## 8.1 Parasitic Boundary Theory

*** To be added. ***

## 8.2 driver_cell



The *driver_cell* command provides a method to describe the characteristics of the driver cell that is driving the external net that connects to an input or bi-directional port of the design.

### 8.2.1 Usage

**driver_cell**

[ -**library** *library*_identifier ] -**cell** *cell*_identifier [ -**instance** *instance*_identifier ] [ -**to** *port*_identifier ]
( [ -**from** *port*_identifier ] [ -**rise_slew** *slew*_rvalue ] [ -**fall_slew** *slew*_rvalue ] )
( [ -**multiplier** multiplier ] | [ -**parallel** *driver*_unsigned_number ] ) [ -**rise** | -**fall** ] [ -**early** | -**late** ]
-**ports** port_list

### 8.2.2 Required Keywords

**-cell** *cell*_identifier

The -*cell* keyword specifies the cell name from a library that should be used to calculate the loading delay for the external net.

-**ports** port_list

The -*port* keyword specifies the port or ports on the design, driven by the driving cell or strength value.

### 8.2.3 Optional Keywords

-**library** *library*_identifier

The -*library* keyword specifies the name of the library that contains the driver cell.

Default: the current technology library for the design (as specified using a mechanism within a tool or supporting tool library) is assumed.

-**instance** *instance*_identifier

The -*instance* keyword specifies the name of a particular instance of the driver cell.

Default: the tool will implicitly name the instance.

**-to** *port*_identifier

If the cell has more than one output port, the -*to* keyword shall be used to specify the one desired output port of the driver cell. If there is only one output port for the driver cell, -*to* is optional.

Default: the output port of the driver cell (if there is only one).

-**from** *port*_identifier

The -*from* keyword specifies the name of the input port on the specified cell driving this port. This pin will be used for both rise and fall transitions.

Default: all inputs of the cell or instance of the cell (if -*instance* used).

Options available when specifying the *-from* keyword are:

-**rise_slew** *slew*_rvalue
-**fall_slew** *slew*_rvalue

The *-rise_slew* and *-fall_slew* keywords assign a rising and/or falling slew value for the input port on the driving cell for the purpose of calculating the loading delay of the external net. The value is a positive, real number.

Default: 0

-**multiplier** multiplier

The *-multiplier* keyword multiplies the delay characteristics of the specified cell by a specified factor.

Default: 1.0

-**parallel** *driver*_unsigned_number

The *-parallel* keyword virtually connects a specific number of the driver cell(s) to the port(s) specified using the *-ports* keyword. The number of cells must be expressed using a positive integer.

Default: 1

Either *-multiplier* or *-parallel* can be used, but not both.

-**rise** | -**fall**

The *-rise* and fall keywords specify the transition on the design port (specified with *-ports*) to which the driver value applies.

Default: if neither keywords are specified, both rise and fall transitions are assumed.

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies that the driver is applied using the late arrival time (setup). The *-early* keyword specifies that the driver is applied using the early arrival time (hold time).

Default: if neither *-early* nor *-late* is specified, both early and late analyses are assumed.

### 8.2.4 Positional Parameters

None.

### 8.2.5 Examples

```
driver_cell -cell d_cell -to Z -ports {in1 in2}
```

The preceding example assigns a driving cell named *d_cell*, output port *Z* to be used for analysis at the design ports *in1*and *in2*.

```
driver_cell -cell buf -to Z -from A -parallel 4 -rise -early \
   -ports {in3}
```

The preceding example assigns a driving cell named *buf* to the design port *in3*. The arc from input *A* to output *Z* on *buf* is specified. This driver applies to the rising transition of *in3* and early analysis should be used.

### 8.2.6 Semantics

The *driver_cell* command models the drive capability of an external driver connected to an input or bi-directional port. The specified cell is not considered part of the circuit, so the output capacitance of the *-to* is not included into the capacitance of the port specified by *-ports*.

It is an error if the cell name specified does not exist in the current library.
It is an error if the cell specified contains more than one output and the *-to* keyword and keyword value is not specified.

The *-instance* keyword allows for the specification of multiple instances of the same driver cell. This eliminates value overwriting problems due to matching commands in a DCDL file.

There are two general methods of assigning multiple drivers to ports in the design (if *-instance* is not specified):

**Simple**: using the *-multiplier* or *-parallel* keywords within a single driver cell command for a given port. This method allows the specification of one driver, but adjusts the influence of the driver. The *-multiplier* keyword has the affect of scaling the driver characteristics while *-parallel* indicates that 1 or more parallel drivers are to be used.

**Cumulative**: using multiple *driver_cell* commands for a given port. The general precedence rules of DCDL apply to multiple *driver_cell* commands (refer to page 37). In addition, these rules apply:

If the *-cell* name matches and the *-to* port and *-ports* names match but the *-from* name does not match, this shall be treated as specifying an additional delay arc to be used in the delay calculation on the design port.

If the *-cell* name and *-ports* names match and the *-to* port name does *not* match, this shall be treated as adding another driver to the design port.

If the *-ports* name matches and the *-cell* name does not match, this shall also be treated as adding another driver to the design port.

*** Might need to add options to support PVT and temperature/voltage regimes.***

### 8.2.7 Related Commands

The following commands are related to the *driver_cell* command:

*driver_resistance*
*port_capacitance*
*slew_time*

## 8.3 driver_resistance

❍

The *driver_resistance* command specifies a resistance value for the cell connected to an external net that is connected to an input or bi-directional port on the design.

### 8.3.1 Usage

**driver_resistance**

[ **-early** | **-late** ] [ **-rise** | **-fall** ]
**-ports** port_list *resistance*_rvalue

### 8.3.2 Required Keywords

**-ports** port_list

The *-ports* keyword specifies the port or ports to which the driver resistance applies.

### 8.3.3 Optional Keywords

**-early** | **-late**

The *-early* or *-late* keywords specify the type of analysis. The *-late* keyword specifies that the drive resistance is applied to the late arrival time (setup). The *-early* keyword specifies that the drive resistance is applied to the early arrival time (hold time).

Default: if neither *-early* nor *-late* is specified, both early and late analyses are assumed.

**-rise** | **-fall**

The *-rise* or *-fall* keywords indicate whether the drive resistance refers to the rising or falling edge of the port.

Default: if neither *-rise* nor *-fall* are specified, both transitions are implied.

### 8.3.4 Positional Parameters

*resistance*_rvalue

The *resistance_rvalue* specifies the resistance for the driver using a real number.

### 8.3.5 Examples

```
driver_resistance -ports {in2 in3} 3.5
```

### 8.3.6 Semantics

*** Interaction with arrival times, *driver_cell*, and *slew_time*, ***

### 8.3.7 Related Commands

The following commands are related to the *driver_resistance* command:

*driver_cell*
*units*

## 8.4 external_sinks

❍

The *external_sinks* command specifies the number of external sinks connected to ports.

### 8.4.1 Usage

**external_sinks**

-**ports** port_list *sinks*_unsigned_number

### 8.4.2 Required Keywords

-**ports** port_list

The *-ports* keyword specifies the port or ports to which the sinks should be connected.

### 8.4.3 Optional Keywords

None.

### 8.4.4 Positional Parameters

*sinks*_unsigned_number

The *sinks_unsigned_number* indicates an integer value representing the number of sinks to connect to the ports.

*** Allow real instead of integer? ***

### 8.4.5 Examples

```
external_sinks -ports {in1 out5} 3
```

### 8.4.6 Semantics

Specifying the number of sinks on the port does not contribute to the fanout count for analyses such as design rule checks.

*** Allow pins? How this relates to *wire_load_model* ***

### 8.4.7 Related Commands

The following commands are related to the *external_sinks* command:

*external_sources*
*wire_load_model*

## 8.5 external_sources

❍

The *external_sources* command specifies the number of external sources connected to ports.

### 8.5.1 Usage

**external_sources**

-**ports** port_list *sources*_unsigned_number

### 8.5.2 Required Keywords

-**ports** port_list

The *-ports* keyword specifies the port or ports to which the sources should be connected.

### 8.5.3 Optional Keywords

None.

### 8.5.4 Positional Parameters

*sources*_unsigned_number

The *sources_unsigned_number* indicates an integer value representing the number of sources to connect to the ports.

*** Allow real instead of integer? ***

### 8.5.5 Examples

```
external_sources -ports {in2 out6} 2
```

### 8.5.6 Semantics

*** Allow pins? How this relates to wire_load_model ***

### 8.5.7 Related Commands

The following commands are related to the *external_sources* command:

*external_sinks*
*wire_load_model*

## 8.6 fanout_load

❍

The *fanout_load* command specifies the number of loads on design ports.

### 8.6.1 Usage

**fanout_load**

-**ports** port_list *load*_unsigned_number

### 8.6.2 Required Keywords

-**ports** port_list

The *-ports* keyword specifies the port or ports to which the load should be connected.

### 8.6.3 Optional Keywords

None.

### 8.6.4 Positional Parameters

*load*_unsigned_number

The *load_unsigned_number* indicates an integer value representing the number of loads to connect to the ports.

*** Allow real instead of integer? ***

### 8.6.5 Examples

```
fanout_load -ports {in3 out7} 4
```

### 8.6.6 Semantics

*** allow pins? ***

### 8.6.7 Related Commands

The following commands are related to the *fanout_load* command:

*port_capacitance*

## 8.7 fanout_load_limit

❍

The *fanout_load_limit* command specifies the maximum fanout load allowed on design ports.

### 8.7.1 Usage

**fanout_limit**

-**ports** port_list *load_limit*_unsigned_number

### 8.7.2 Required Keywords

-**ports** port_list

The *-ports* keyword specifies the port or ports to which the load limit should be applied.

### 8.7.3 Optional Keywords

None.

### 8.7.4 Positional Parameters

*load_limit*_unsigned_number

The *load_limit_unsigned_number* indicates an integer value representing the load limits for the specified ports.

*** Allow real instead of integer? ***

### 8.7.5 Examples

```
fanout_load_limit -ports {in3 out7} 4
```

### 8.7.6 Semantics

*** allow pins? ***

### 8.7.7 Related Commands

The following commands are related to the *fanout_load_limit* command:

*fanout_load*

## 8.8 port_capacitance



The *port_capacitance* command specifies the capacitance external to a port in a design, based on input and output loading from other pins and nets connected to the port.

### 8.8.1 Usage

**port_capacitance**

[ -**early** | -**late** | -**typ** ] [ -**pin_load** | -**wire_load** | -**lumped_load** ] -**ports** port_list *capacitance*_rvalue_list

### 8.8.2 Required Keywords

-**ports** port_list

The -*ports* keyword specifies the port or list of ports to which the external capacitance shall be assigned.

### 8.8.3 Optional Keywords

**-early** | **-late** | **-typ**

The -*early* or -*late* keywords specify the type of analysis. The -*late* keyword specifies setup time analysis and the -*early* keyword specifies hold time analysis.

The -*typ* keyword indicates that the nominal operating condition as used in computing the *typ* slot in an SDF triplet should be used for delay calculation purposes.

Default: if neither -*early*, -*late*, nor -*typ* is specified, the port capacitance should be used in all analysis types.

-**pin_load** | -**wire_load** | -**lumped_load**

The -*pin_load* keyword indicates that the capacitance value represents the total pin capacitance for the external net (not the wire capacitance) on the port(s) specified by -*ports*.

The -*wire_load* keyword indicates that the capacitance value represents only the external wire capacitance (not the pin capacitance) on the port(s) specified by -*ports*.

The -*lumped_load* keyword indicates that the capacitance value represents both the pin and the wire load on the port(s) specified by -*ports*.

Default: if neither of these three options are specified, -*pin_load* is assumed.

### 8.8.4 Positional Parameters

*capacitance*_rvalue_list

The *capacitance_rvalue_list* specifies a positive, real number indicating the capacitance value. This value can be a single value that applies in early and late analysis, or 2 values that indicate early and late values.

### 8.8.5 Examples

```
port_capacitance -ports {WriteData[*] MemWrite} {0.2 0.4}
```

The preceding example shows specifying both the early and late capacitance values that should be assigned to all bus ports of *WriteData* and the *MemWrite* port. The capacitance model is pin load (default if not specified).

```
port_capacitance -lumped_load -ports {MemRead} 0.4
```

The preceding example shows the assignment of capacitance to the *MemRead* port using the lumped load model. The 0.4 value applies to both the early and late analysis.

### 8.8.6 Semantics

The capacitance value elements are determined by the -*pin_load*, -*wire_load*, or -*lumped_load* keywords.

### 8.8.7 Related Commands

The following commands are related to the *port_capacitance* command:

*port_capacitance_limit*
*port_wire_load*
*units*

## 8.9 port_capacitance_limit

The *port_capacitance_limit* command specifies the maximum capacitance value from a source external to the design port.

### 8.9.1 Usage

**port_capacitance_limit**

-**ports** port_list *load_limit*_rvalue

### 8.9.2 Required Keywords

-**ports** port_list

The *-ports* keyword specifies the port or ports to which the limit should be applied.

### 8.9.3 Optional Keywords

None. *** early and late needed? ***

### 8.9.4 Positional Parameters

*load_limit*_rvalue

The *load_limit_rvalue* indicates an real value representing the maximum capacitance for the specified port or ports.

### 8.9.5 Examples

```
port_capacitance_limit {in1 out5} 6.5
```

### 8.9.6 Semantics

The *port_capacitance_limit* command allows limiting external port capacitance contributed by loading from connected ports and nets.

*** Allow pins? How this relates to *wire_load_model* ***

### 8.9.7 Related Commands

The following commands are related to the *port_capacitance_limit* command:

*port_capacitance*
*units*
*wire_load_model*

## 8.10 port_wire_load

❍

The *port_wire_load* command specifies a wire load model for a specified port.

### 8.10.1 Usage

**port_wire_load**

[ **-library** *library*_identifier ] **-ports** port_list *wire_load_model*_identifier

### 8.10.2 Required Keywords

-**ports** port_list

The *-ports* keyword specifies the port or ports to which the wire load should be applied.

### 8.10.3 Optional Keywords

**-library** *library*_identifier

The *-library* keyword specifies the name of the library that contains the wire load model.

Default: if *-library* is not specified, the current technology library for the design is used.

### 8.10.4 Positional Parameters

*wire_load_model*_identifier

The *wire_load_model_indentifier* specifies the name of the wire load model from as specified in the technology library.

### 8.10.5 Examples

```
port_wire_load -library acme2.1 -ports {in1 out1} 10KGATES_5
```

### 8.10.6 Semantics

*** Not sure if this command is to assign a particular wire load model to a port, or some kind of load value - assumed model. This comes from Ambit. Do we allow pins also and add a separate command? ***

### 8.10.7 Related Commands

The following commands are related to the *port_wire_load* command:

*wire_load_model*

## 8.11 wire_load_model

The *wire_load_model* command specifies which wire load model should be applied from a library.

### 8.11.1 Usage

**wire_load_model**

[ **-library** *library*_identifier ] [ **-instances** instance_list ] *wire_load_model*_identifier

### 8.11.2 Required Keywords

None.

### 8.11.3 Optional Keywords

**-library** *library*_identifier

The -*library* keyword specifies the name of the library that contains the wire load model.

Default: if a library is not specified, the current technology library for the design is used.

**-instances** instance_list

The -*instances* keyword specifies a particular instance or list of instances to which to apply the wire load model. If the *wire_load_model* command applies to the top level instance, this keyword is not necessary.

If the *current_scope* command has been specified before the *wire_load_model* command and -*instances* is not specified, the wire load model is applied to the appropriate scope.

### 8.11.4 Positional Parameters

*wire_load_model*_identifier

The *wire_load_model_indentifier* specifies the name of the wire load model as specified in the technology library.

### 8.11.5 Examples

```
wire_load_model -library acme2.1 10KGATES_5
wire_load_model -library acme2.1 -instances {U11063 U1} 5KGATES_5
```

### 8.11.6 Semantics

### 8.11.7 Related Commands

The following commands are related to the *wire_load_model* command:

*current_scope*
*port_wire_load*

*** **NOTES**: Ambit provides a method to annotate physical data with *set_wire_capacitance*, *set_wire_resistance* as well as handles wireload estimates using *set_wire_load_mode*. These should be considered in the context of the 9/28/99 discussion on physically-aware synthesis during the phone conference.

The information about RCL and M found at the end of the parasitic boundary section in the taxonomy was not included. It looks like we need this information, but it is not ready for placement in the specification without some preliminary discussion.

***

# 9. Standard Compliance

*** Will we have levels of compliance like GCF or VITAL?***

EDA tools shall be considered compliant with this standard if ***

*** Do we need a statement about the use of *extend_dcdl*? ***

# 10. Glossary

For the purposes of this specification, the following terms and definitions apply. IEEE Std 100-1992, *The New IEEE Standard Dictionary of Electrical and Electronics Terms*, should be referenced for terms not defined in this specification.

10.1 **actual mode**: ***

10.2 **annotation**: characterized data attached to a design object that is based on the design implementation.

10.3 **application**, **EDA application**: any software program that interacts with DCDL.

10.4 **arc**: *See:* **timing arc**.

10.5 **arrival time**: ***

10.6 **assertion**: a statement of truth about a design object that a tool accepts as fact during analysis.

10.7 **back annotation**: the addition of information from further downstream steps (towards fabrication) in the design process. *See also*: **annotation**.

10.8 **back annotation file**: a file containing information to be read by a tool for the purpose of back annotation. *See also:* **back annotation**.

10.9 **bi-directional**: a pin or port which can place logic signals onto an interconnect and receive logic signals from it (i.e., act both as a driver and a receiver).

10.10 **borrow**: ***

10.11 **boundary**: the outer part of an integrated circuit where instances of cell types designed specifically to interface the internal circuitry to the “outside world” are placed. This part includes “pad” cells (which are input and output buffers) and power and ground pads; it may also include test circuitry, such as boundary scan cells.

10.12 **buffer tree**: ***

10.13 **bus**: a collection of nets, pins, or ports.

10.14 **cell**: a functional design unit usually contained in a library.

10.15 **chip**: ***

10.16 **clock**: ***

10.17 **clock root**: the physical pins or ports to which a virtual or ideal waveform is associated.

10.18 **clock tree**: ***

10.19 **constraint**: a desired characteristic that an EDA tool must enforce or satisfy.

10.20 **correlation**: specifies whether early and late delays and slews should be computed assuming that variations in operating conditions are on the chip (correlated) or between chips (uncorrelated).

10.21 **delay**: the time taken for a digital signal to propagate between two points.

10.22 **delay arc**: *See:* **timing arc**.

10.23 **delay equation**: any mathematical expression describing cell delay or interconnect delay.

10.24 **departure time**: ***

10.25 **design**: ***

10.26 **directive**: a statement that directs a tool to implement a specific characteristic.

10.27 **driver**: a pin of a cell instance that, in the current context, is placing or can place a signal onto an interconnect structure.

10.28 **early mode**: ***

10.29 **false path**: ***

10.30 **fanin**: ***

10.31 **fanout**: the pin count of a net (the number of pins connected to the net), minus one. This definition includes all input, output, and bidirectional pins on the net with the sole exception of one pin (assumed to be related to the particular timing arc currently of interest). Although less fundamental than pin count, fanout is frequently used in the definition of wireload models.

10.32 **forward annotation**: the annotation of information from further upstream (earlier in the design flow) in the design process. *See also:* **forward annotation file**.

10.33 **forward annotation file**: a file containing information to be read by a tool for the purpose of forward annotation, for example an SDF file. *See also:* **forward annotation**.

10.34 **gate**: a logical abstraction of an library primitive.

10.35 **hierarchical instance**: the concrete appearance of a design unit at some hierarchical level. Because higher-level design units may be instantiated multiple times, a single such appearance may give rise to multiple instances of the lower-level design units within it. Where instances are referred to as “occurrences”, hierarchical instances are referred to simply as instances.

10.36 **ideal mode**: ***

10.37 **implementation** (hardware): ***

10.38 **implementation** (software): ***

10.39 **implementation-defined behavior**: behavior, for a correct program construct and correct data, that depends on the implementation and that each implementation shall document. The range of possible behaviors is delineated by this specification.

10.40 **implementation limits**: restrictions imposed by an implementation.

10.41 **input**: a pin or port that shall only receive logic signals from a connected net or interconnect structure.

10.42 **insertion delay**: ***

10.43 **instance**, **cell instance**: a particular, concrete appearance of a cell in the fully-expanded (flattened, unfolded, elaborated) design description of an integrated circuit, also referred to elsewhere as an "occurrence." An instance is a "leaf" of the unfolded design hierarchy. *See also:* **cell**, **hierarchical instance**.

10.44 **intent** (or design intent): ***

10.45 **interconnect**: a collective term for structures (in an integrated circuit) that propagate a signal between the pins of cell instances with as little change as possible.

10.46 **jitter**: ***

10.47 **latch**: ***

10.48 **late mode**: ***

10.49 **leading edge**: ***

10.50 **level-sensitive**: ***

10.51 **library**: a collection of circuit functions, implemented in a particular integrated circuit technology, that an integrated circuit designer or EDA application can select in order to implement a design. *See also:* **cell**.

10.52 **load**: ***

10.53 **logical**: *** as opposed to physical ***

10.54 **logical file**: a DCDL file with all included files expanded.

10.55 **module**: ***

10.56 **multi-cycle path**: ***

10.57 **net**, **net instance**: an abstraction expressing the idea of an electrical connection between various points in a design. In a hierarchical representation of the design, nets can occur at all levels and may connect to pins of lower hierarchical levels (including cell instances), ports of the current hierarchical level and each other. In a flattened (unfolded and elaborated) design, electrically connected nets are collapsed and each net instance corresponds to an unique interconnect structure in the implementation.

10.58 **node**: a conceptual point (through which logic signals pass) that has been identified as an aid to modeling the timing properties of a cell but may not correspond to any physical structure.

10.59 **operating condition**: ***

10.60 **output**: a pin or port that shall only place logic signals onto a connected net or interconnect structure.

10.61 **overshoot**: ***

10.62 **parameter**: a data item required for the calculation of some result.

10.63 **parasitics**: electrical properties of a design (resistance, capacitance, and impedance) that arise due to the nature of the materials used to implement the design.

10.64 **pathname**: a set of characters separated by hierarchy characters that represent the relative position/location of a design object.

10.65 **periphery**: *See* **boundary**.

10.66 **physical**: *** as opposed to logical ***

10.67 **physical file**: an individual DCDL file containing unexpanded includes.

10.68 **pin**: a terminal point where an interconnect structure makes electrical contact with the fixed structures of a cell instance; or the conceptual point where a net connects to a lower level in the design hierarchy.

10.69 **pin count**: the number of cell instance pins that an interconnect structure visits, including all input, output, and bidirectional pins. Pin count is the number of “places” the interconnect goes to on the chip. A pin count of less than two is not possible.

10.70 **port**: a conceptual point at which a cell or a hierarchical design unit makes its interface available to higher levels in the design hierarchy.

10.71 **positional parameter**: a value or identifier not associated with a keyword in a DCDL command.

10.72 **primary input**: the point where a logic signal arrives at the boundary of the design as currently known to an EDA application. For a complete integrated circuit design, for example, this point is the metal pad of an input or bidirectional pad cell.

10.73 **primary output**: the point where a logic signal leaves the design as currently known to an EDA application. For a complete integrated circuit design, for example, this point is the metal pad of an output or bidirectional pad cell.

10.74 **process point**: ***

10.75 **rail**: ***

10.76 **RC, RC time constant**: the product of some resistance and some capacitance (having the dimensions of time) or a time constant computed in some other way. *Syn:* **Elmore delay**.

10.77 **receiver**: a pin of a cell instance that is receiving or can receive a signal from an interconnect structure.

10.78 **regime**: ***

10.79 **required time**: ***

10.80 **shared port**: an output or bidirectional port where some other output port of the cell derives its logic function. The output load at a shared port affects not only the delay to that port itself, but also the delay to any ports sharing it.

10.81 **sink**, **sink pin**: the pin is the end of a delay arc, i.e. the destination of the logic signal. For arcs across cell instances, the sink is the driver pin. For arcs across interconnect structures, the sink is the receiver pin.

10.82 **skew**: ***

10.83 **slew**: a measure of the shape of the waveform constituting a logic state transition. A slew value can have the dimensions of time, in which case it is a slew time, or the dimensions of voltage-per-time, in which case it is a slew rate.

10.84 **slew-dependent delay**: that part of an input-to-output delay that can be attributed to the signal at the input of the arc taking longer to make a transition than is considered ideal.

10.85 **slew rate**: a measure of how quickly a signal takes to make a transition, i.e., a voltage-per-unit time. Slew rate is inversely related to *slew time* and is sometimes used incorrectly where *slew time* is intended.

10.86 **slew time**: a measure of how long a signal takes to make a transition, i.e., the rise time or fall time. Slew time is inversely related to slew rate. The way a slew time value is abstracted from the continuous waveform at a cell pin varies with different cell characterization methods.

10.87 **source**, **source pin**: the source pin is the start of a delay arc, i.e., the origin of the logic signal. For arcs across cell instances, the source is the receiver pin. For arcs across interconnect structures, the source is the driver pin.

10.88 **target**, **target pin**: ***

10.89 **time-of-flight**: the time delay between a signal leaving a driving pin or primary input port and reaching a receiving pin or primary output port (measured at the logic threshold). Although light-speed delay may be significant in some cases, time-of-flight is generally dominated by the time taken to charge the distributed capacitance of the interconnect and the capacitance of the driven pins through the distributed impedance of the interconnect. The internal impedance of the driving port affects the load-dependent delay but not (directly) the time-of-flight.

10.90 **timing annotation (file)**: the annotation of a design in one tool with timing data computed by another tool. If timing calculation is performed as an off-line process (separately from the application using the timing data), the process of reading the timing data into the tool is known as timing annotation. A timing annotation file stores the data written by the timing calculator and is later read by an application. *Syn*: **back annotation**.

10.91 **timing arc**: a pair of ports, pins, or nodes that possess some timing relationship such as the propagation delay of a signal from one to the other or a timing check between them. Delay arcs may be from cell inputs to outputs or over the interconnect from driver pins to receiver pins.

10.92 **timing check**: a timing property of a circuit (frequently a cell) that describes a relationship in time between two input signal events. This relationship needs to be satisfied for the circuit to function correctly.

10.93 **timing model**: a timing model represents the timing behavior of a cell for applications such as simulation and timing analysis. For black-box timing behavior, it represents the definition of pin-to-pin delays between any pair of pins as well as internal nodes. In addition, for sequential cells it provides the definition of timing checks and constraints on any pair of pins and/or internal nodes.

10.94 **trailing edge**: ***

10.95 **transition**: the change of a logic signal from one state to another (as in "… a transition at the input shall cause …") or the pair of logic states between which a transition may occur (as in "… the delay for a low-to-high transition …").

10.96 **uncertainty**: ***

10.97 **unloaded delay**: the conceptual delay value for a delay arc of a cell when the output pin is unloaded (unconnected) and the signal at the input pin conforms to some ideal waveform. *Syn:* **intrinsic delay**.

10.98 **undefined behavior**: behavior for which this specification imposes no requirements (e.g., use of an erroneous program construct). Permissible undefined behavior ranges from

- ignoring a situation completely with unpredictable results;
- behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message);
- terminating a translation or execution (with the issuance of a diagnostic message).

10.99 **unspecified behavior**: behavior (for a correct program construct and correct data) that depends on the implementation. The implementation is not required to document which behavior occurs. Usually the range of possible behaviors is delineated by this specification.

10.100 **waveform**: ***

10.101 **wireload model**: a statistical model that estimates interconnect properties as a function of the geometric measures available before the completion of layout and routing. Typical model properties include: fanout, capacitance, length, and resistance.

# Annex A

# BNF

*** Min, typ, and max values are not represented yet. And/or constructs not represented well, like *-ports* and/or -*pins*.***

### A.1.1 General Structure BNF. . . . . .

dcdl_line_entry :: command | comment

comment ::= command **;** #text | ( # command | [ text ] )

command ::=
    universal_command |
    scoping_command |
    operating_conditions_command |
    timing_command |
    parasitics_command

universal_command ::=
    constant |
    design_name_space |
    extend_dcdl |
    functional_mode
    history |
    include |
    tool_domain |
    units |
    version

scoping_command ::=
    current_scope

operating_conditions_command ::=
    operating_point |
    operating_process |
    operating_range |
    operating_temperature |
    operating_voltage |
    temperature_regime |
    voltage_regime

timing_command ::=
    clock_command |
    timing_boundary_command |
    timing_exception_command

```
clock_command ::=
        clock |
        clock_arrival_time |
        clock_delay |
        clock_mode |
        clock_required_time |
        clock_skew |
        clock_uncertainty |
        common_insertion_delay |
        derived_waveform |
        target_uncertainty |
        waveform

timing_boundary_command ::=
        data_arrival_time |
        data_required_time |
        departure_time |
        external_delay
        slew_limit |
        slew_time

timing_exception_command ::=
        borrow_limit|
        disable |
        false_path |
        multi_cycle_path |
        tree_delay |
        tree_mode

parasitics_command ::=
        driver_resistance |
        driver_cell |
        external_sinks |
        external_sources |
        fanout_load |
        fanout_load_limit |
        port_capacitance |
        port_capacitance_limit |
        port_wire_load |
        wire_load_model
```

### A.1.2 Shared BNF Constructs

character ::=
nonreserved_character |
reserved_character |
whitespace

nonreserved_character ::= a - z | A-Z | integer_digit
| **_** | **$** | **.** | **/** | **&** | | | ^ | ~ | + | - | **%** | **!** | = | < | > | **:** | **(** | **)** | **[** | **]** | **@** | **,** | **‘**

reserved_character ::= ***** | **?** | **{** | **}** | **;** | **“** | **#** | \

whitespace ::= *space* | *horizontal_tab*

text ::= character_set

character_set ::= whitespace { whitespace } | nonreserved_character { nonreserved_character }

character_range ::= nonreserved_character **-** nonreserved_character
{ nonreserved_character **-** nonreserved_character }

identifier ::= nonreserved_character { nonreserved_character }[1]

identifier_list ::= **{** identifier **}** |
**{** identifier { identifier } **}**

real ::= unsigned_number **.** unsigned_number |
usigned_number [ **.** unsigned_number ] exponent unsigned_number

integer ::= [ sign ] unsigned_number

unsigned_number ::= integer_digit { integer_digit }

integer_digit ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

exponent ::= **E** | **e** [ sign ]

rvalue ::= real

rvalue_list ::= **{** real **}** |
**{** real { real } **}**

rsvalue ::= [ sign ] real

sign ::= **+** | **-**

time_value ::= [ sign ] real | placeholder

unsigned_time_value ::= real | placeholder

[1]Actual characters governed by design_name_space command.

time_value_list ::= **{** time_value **}** |
*** Might need to be more specific here, as it is usually 2 or 4 values only ***
**{** time_value { time_value } **}**

placeholder ::= ***

port_list ::= **{** object_identifier **}** |
**{** object_identifier { object_identifier } **} |**
wildcard

pin_list ::= **{** object_identifier **}** |
**{** object_identifier { object_identifier } **} |**
wildcard

instance_list ::= **{** object_identifier **}** |
**{** object_identifier { object_identifier } **} |**
wildcard

cell_list ::= **{** identifier **}** |
**{** object_identifier { object_identifier } **} |**
wildcard

wildcard ::= identifier***** | ***** | *****identifier***** | identifier**?**

object_identifier ::= identifier |
pathname

pathname ::= **“** [*hierarchy_delimiter*_character] {identifier [*hierarchy_delimiter*_character]} **“**

multiplier ::= real

### A.1.3 Universal Commands BNF . . . . . .

constant ::= **constant** ( -**ports** port_list | -**pins** pin_list ) **0** | **1**

design_name_space ::= **design_name_space -verilog** ( **“ 1995 ”** | **“ 2000 “**) |
**-vhdl** ( **“ 1987 “ | “ 1993 “** | **“ 2000 “** ) | **-edif** ( **“2 0 0 “** | **“ 3 0 0 “** | **“ 4 0 0** “ ) |
**-custom** ( **-characters “ [** character_set **] ”** | **“ [** character_range **] ”** |
**“ [** character_set character_range **] ” |**
**“ [** character_range character_set **] ”** )
( **-case_sensitive** | **-case_insensitive** )
( **-character_escape “** *escape*_character **”** |
**-string_escape_start “** *escape*_character **“** [ **-string_escape_end “** *escape*_character **”** ] )
( **-escape_type include** | **exclude** )
( **-bus_range_separator_up “** *index*_character **”** | **“** *index*_identifier **”** )
(**-bus_range_separator_down “** *index*_character **”** | **“** *index*_identifier **”** )
( **-bus_bit_left “** *bit*_character **”** )
( **-bus_bit_right “** *bit*_character **”** )
( **-hierarchy_delimiter “** *delimiter*_character **”** )

extend_dcdl ::= **extend_dcdl** *command*_identifier [**-arguments “** *argument*_text **“** ]

functional_mode::= **functional_mode** ( [ -**group_name** *group*_identifier ]
-**mode_name** *mode*_identifier ) | ( -**all** | -**default** ) instance_list

history ::= **history “** *history*_text **”**

include ::= **include** [ -**inline** ] **“** *pathname*_identifier **”**

units :: = **units** [-**time** multiplier] [-**capacitance** multiplier] [-**resistance** multiplier]
[ -**voltage** multiplier ] [ -**temperature** multiplier ] [-**inductance** multiplier]

version ::= **version** version_identifier

version_identifier ::= **“** V1.0 **“**

### A.1.4 Scoping Commands

current_scope ::= **current_scope -instance** *instance*_identifier | -**cell** *cell*_identifier | -**top** |
-**up** *level*_unsigned_number

### A.1.5 Operating Conditions BNF

operating_point ::= **operating_point** [ -**voltage_regime** *voltage_regime*_identifier ]
[ -**temperature_regime** *temperature_regime*_identifier ] [ -**library** *library*_identifier ]
-**name** *operating_point*_identifier ( -**best** | -**nominal** | -**worst | -min_best** | -**typ_best** |
-**max_best** | -**min_worst** | -**typ_worst** | -**max_worst** )

operating_process ::= **operating_process** [ -**library** *library*_identifier ]
[ -**value** *operating_point*_rvalue ] ( -**best** | -**nominal** | -**worst | -min_best** | -**typ_best** |
-**max_best** | -**min_worst** | -**typ_worst** | -**max_worst** )

operating_range ::= **operating_range** [ -**library** *library*_identifier ] *operating_range*_identifier

operating_temperature ::= **operating_temperature**
( [ -**temperature_regime** *temperature_regime*_identifier ] |
[ -**instances** instance_list [ -**pin** *pin*_identifier ] )
[ -**library** *library*_identifier ] [ -**value** *operating_point*_rvalue ] ( -**best** | -**nominal** | -**worst** |
-**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst** )

operating_voltage ::= **operating_voltage** ( [ -**voltage_regime** *voltage_regime*_identifier ] |
[ -**instances** instance_list [ -**pin** *pin*_identifier ] )
[ -**library** *library*_identifier ] [ -**value** *operating_point*_rvalue ] ( -**best** | -**nominal** | -**worst** |
-**min_best** | -**typ_best** | -**max_best** | -**min_worst** | -**typ_worst** | -**max_worst** )

temperature_regime ::= **temperature_regime** [ -**cells** cell_list ] |
[ -**instances** instance_list ]
*temperature_regime*_identifier

voltage_regime ::= **voltage_regime** [ -**logical_rail** *logical_rail*_identifier ] |
[ -**physical_rail** *physical_rail*_identifier ]
[ -**base_voltage** *voltage*_rvalue ]
[ -**min_voltage** *minimum*_rvalue ] [ -**max_voltage** *maximum*_rvalue ]
[ ( -**cells** cell_list [ -**port** *port*_identifier ] ) ] |
[ ( -**instances** instance_list [ -**pin** *pin*_identifier ] )]
*voltage_regime*_identifier

### A.1.6 Clock Commands BNF

clock ::= **clock** -**waveform** *waveform*_identifier ( -**pins** pin_list | -**ports** port_list )
[ -**parent_pin** *pin*_identifier | -**parent_port** *port*_identifier ]

clock_arrival_time ::= **clock_arrival_time** -**waveform** *waveform*_identifier
[ -**lead** | -**trail** ] [ -**early** | -**late** ]
-**ports** port_list | -**pins** pin_list *clock_arrival*_time_value_list

clock_delay ::= **clock_delay** -**waveform** *waveform*_identifier |
( -**root_port** *port*_identifier | -**root_pin** *pin*_identifier ) | ( -**leaf** *pin*_identifier )
[ -**rise** | -**fall** ] [ -**early** | -**late** ] *delay*_unsigned_time_value_list

clock_mode ::= **clock_mode** [ -**root_port** port_list | -**root_pin** pin_list ] -**ideal** | -**actual**

clock_skew ::= **clock_skew** ( -**root_port** *port*_identifier | -**root_pin** *pin*_identifier ) [ -**rise** | -**fall** ]
[ -**early** | -**late** ] *skew*_unsigned_time_value_list

clock_required_time ::= **clock_required_time** -**waveform** *waveform*_identifier [ -**lead** | -**trail** ]
[ -**early** | -**late** ] -**ports** port_list | -**pins** pin_list *clock_required*_time_value_list

clock_uncertainty ::= **clock_uncertainty** [ -**from** *root_waveform*_identifier ]
[ -**to** *target_waveform*_identifier ] [ -**from_edge rise** | **fall** ]
[ -**to_edge rise** | **fall** ] [ -**early** | -**late** ] [ -**absolute** | -**increment** ] [ -**ideal** | -**actual** ]
**{** *uncertainty*_rsvalue **}**

common_insertion_delay ::= **common_insertion_delay** ( -**from_port** *clock_port*_identifier |
-**from_pin** *clock_pin*_identifier ) ( -**to_port** *clock_port*_identifier |
-**to_pin** *clock_pin*_identifier ) [ -**rise** | -**fall** ] [ -**early** | -**late** ] *insertion*_rvalue_list

derived_waveform ::= **derived_waveform** -**waveform** *parent_waveform*_identifier
-**name** *derived_waveform*_identifier [ -**inverted** ] [ -**phase {** *offset_shift*_rsvalue_list **}** ]
( [ -**multiplier** *mult*_unsigned_number ] [ -**divisor** *divisor*_unsigned_number ] ) |
[ -**derived_edges {** l*ead_edge*_unsigned_number *trail_edge*_unsigned_number **}** ]
[ -**lead_jitter** jitter_value | -**trail_jitter** jitter_value ]

jitter_value ::= **{** *left*_unsigned_time_value *right*_unsigned_time_value **}** |
**{** *offset*_unsigned_time_value **}** [ -**increment** | -**absolute** ]

target_uncertainty ::= **target_uncertainty** -**port** *clock_root*_identifier |
( -**pin** *clock_leaf*_identifier | *clock_root*_identifier ) | -**instance** *instance*_identifier
[ -**early** | -**late** ] [ -**absolute** | -**increment** ] [ -**ideal** | -**actual** ] *uncertainty*_rsvalue

tree_delay ::= **tree_delay** -**root_port** *port*_identifier | -**root_pin** *pin*_identifier
[ -**ideal** | -**actual** ] [ -**explict_leaf** pin_list ]
[ -**default_insertion** *delay*_rvalue ] [ -**explicit_insertion** *delay*_rvalue ]

[ **-internal_insertion** *delay*_rvalue ]
[ **-default_skew** *skew*_rvalue ] [ **-default_transition** *time*_rvalue ]
[ **-explicit_transition** *time*_rvalue ]

tree_mode ::= **tree_mode -ideal** | **-actual**

waveform ::= **waveform -name** *waveform*_identifier [ **-period** *period*_rvalue ]
[ **-edges {** *lead*_rsvalue *trail*_rsvalue **}**]
[ **-lead_jitter {** *left*_unsigned_time_value *right*_unsigned_time_value **}** |
**{** *offset*_unsigned_time_value **}** ]
[ **-trail_jitter {** *left*_unsigned_time_value *right*_unsigned_time_value **}** |
**{** *offset*_unsigned_time_value **}** ]
[ **-inverted** ] [ **-domain** *domain*_identifier ]

### A.1.7 Timing Boundary Commands BNF

data_arrival_time ::= **data_arrival_time -waveform** *waveform*_identifier
[ **-target** | **-source** ] [ **-lead** | **-trail** ] [ **-early** | **-late** ] [ **-rise** | **-fall** ]
**-ports** port_list | **-pins** pin_list
*arrival*_time_value_list

data_required_time ::= **data_required_time -waveform** *waveform*_identifier
[ **-target** | **-source** ] [ **-lead** | **-trail** ]
[ **-early** | **-late** ] [ **-rise** | **-fall** ] **-ports** port_list | **-pins** pin_list *required*_time_value_list

departure_time ::= **departure_time -waveform** *waveform*_identifier
[ **-early** | **-late** ] [ **-rise** | **-fall** ]
**-ports** port_list | **-pins** pin_list *departure*_time_value_list

external_delay ::= **external_delay -waveform** *waveform*_identifier [ **-early** | **-late** ]
**-ports** port_list | **-pins** pin_list
**-rise_range** *rise*_time_value_list **-fall_range** *fall*_time_value_list

slew_limit ::= **slew_limit** [ **-early** | **-late** ] [ **-rise** | **-fall** ]
**-ports** port_list | **-pins** pin_list *slew_limit*_time_value

slew_time ::= **slew_time** [ **-early** | **-late** ] [ **-rise** | **-fall** ] **-ports** port_list | **-pins** pin_list
*slew*_time_value_list

### A.1.8 Timing Exception Commands BNF

*** Not sure how to specify one or more keywords in BNF. For example, in the following commands, *-to*, *-from*, and *-through* - one or more must be specified. ***

borrow_limit ::= **borrow_limit** [ **-ports** port_list | **-pins** pin_list |
**-waveform** *waveform*_identifier |
**-instances** instance_list ] *borrow_limit*_rvalue

disable ::= **disable** [ -**library** *library*_identifier ] -**cell** *cell*_identifier | -**instance** i*nstance*_identifier
[ -**from_port** port_list | -**from_pin** pin_list ] [ -**to_port** port_list | -**to_pin** pin_list ] |
[ -**input_arcs** ] | [ -**internal_arcs** ]

false_path ::= **false_path** [ -**early** | -**late** ] [ -**rise** | -**fall** ] path_options

multi_cycle_path ::= **multi_cycle_path** [ -**target** | -**source | -waveform** *waveform*_identifier ]
[ -**early** | -**late** ] [ -**rise** | -**fall** ] path_options **{** cycle_number **}**

cycle_number ::= [ sign ] real | [ sign ] unsigned_number |
**{** [ sign ] real [ sign] real **} |**
**{** [ sign ] unsigned_number [ sign ] unsigned_number **}**
**{** [ sign ] real [ sign ] unsigned_number **} | {** [ sign ] unsigned_number [ sign ] real **} |**
placeholder

path_options ::= ( -**from_port** | -**from_pin** | -**from_instance** | -**from_waveform**
**{** object_identifier **}** ) |
( -**from_port** | -**from_pin** | -**from_instance** | -**from_waveform {** object_identifier **}** ) |
( { -**through_port** | -**through_pin** | -**through_instance** | -**through_net**
**{** object_identifier **}** } )

### A.1.9 Parasitics Commands BNF . . . . . .

driver_cell ::= **driver_cell** [ -**library** *library*_identifier ] -**cell** *cell*_identifier
[ -**instance** *instance*_identifier ] [ -**to** *port*_identifier ]
( [ -**from** *port*_identifier ] [ -**rise_slew** *slew*_rvalue ] [ -**fall_slew** *slew*_rvalue ] )
( [ -**multiplier** multiplier ] | [ -**parallel** *driver*_unsigned_number ] ) [ -**rise** | -**fall** ]
[ -**early** | -**late** ] -**ports** port_list

driver_resistance ::= **driver_resistance** [ -**early** | -**late** ] [ -**rise** | -**fall** ] -**ports** port_list
*resistance*_rvalue

external_sinks ::= **external_sinks** -**ports** port_list *sinks*_unsigned_number

external_sources ::= **external_sources** -**ports** port_list *sources*_unsigned_number

fanout_load ::= **fanout_load** -**ports** port_list *load*_unsigned_number

fanout_load_limit ::= **fanout_load_limit** -**ports** port_list *load_limit*_unsigned_number

port_capacitance ::= [ -**early** | -**late | -typ** ] [ -**pin_load** | -**wire_load** | -**lumped_load** ]
-**ports** port_list *capacitance*_rvalue_list

port_capacitance_limit ::= **port_capacitance_limit** -**ports** port_list *load_limit*_rvalue

port_wire_load ::= **port_wire_load** [ -**library** *library*_identifier ] -**ports** port_list
*wire_load_model*_identifier

wire_load_model ::= **wire_load_model** [ -**library** *library*_identifier ]
[ -**instances** instance_list ] *wire_load_model*_identifier

# Annex B

(informative)

# Bibliography

[C1] Cadence Design Systems, Inc., “Command Reference Manual (Ambit strawman)

[C2] Cadence Design Systems, Inc., “General Constraint Format Specification”, Version 2.0, January 26, 1999.

[C3] Mentor Graphics Corporation, “SST Velocity Reference Manual”, Version 3.2., 1999.

[C4] Open Library API (OLA) specification, Version 1.0.2, May 1999.

[C5] IEEE (1003.2) ISO/IEC 9945-2 “Portable Operating System Interface (POSIX) Part 2: Shell and Utilites”, Section 2, 1993.

[C6] Barry N. Taylor, “Guide for Use of the International System of Units”,1995

# Annex C

(informative)

# Analyzed and Rejected Functionality

This annex houses the features and commands that were discussed, analyzed, and ultimately rejected for inclusion in this version of the standard. This functionality is provided here as an archive so that the reader can understand the rationale for not including it into the specification and to potentially re-visit in future versions of the standard.

## C.1 Merging Constraint Values

The DC-WG considered whether or not the value of merging constraints (and potentially allowing incremental constraints) was great enough to include in the first phase of this specification. An example of merging is handling the creation of a new constraint value based on minimum and maximum values for a matching constraint.

**Removed from after section**: 3.3.7.3 “Unsetting and Resetting Constraint Values”

**Justification for rejection**: the complexity that merging implies is very great while the return on investment is not known. For example, it is not known whether designers even want this feature.

## C.2 The persistent_comment Command

The purpose of the *persistent_comment* command was to allow comments to be passed through to a DCDL file that is written out of a tool. Normal comments, represented by the # character, are ignored during reading and thus would be lost during the writing of DCDL.

**Removed from after section**: 4.6 “include”

**Justification for rejection**: due to the potential of design transformations, keeping track of the exact location within the output DCDL file for the comment would be very difficult for most design tools to support. If the designer wanted persistent comments and was working with a tool that could support this concept, the *extend_dcdl* command could be used to prototype this feature.

## C.3 The tool_domain Command

The purpose of the *tool_domain* command was to specify a region of DCDL commands that applied to a particular tool or tool type. This is similar to the pragma concept.

**Removed from after section**: 4.6 “include”

**Justification for rejection**: a user can accomplish this functionality using several other techniques. The *include* command could be used or embedding DCDL in an extension language that had case statements would accomplish this task.

## C.4 Tags

Tags apply when modeling the constraints at the boundary of a hierarchical block or a chip. In most tools, there can be different arrival (required) times at an input (output) pin, one for each of the clock signals associated with registers in other blocks which are in the cone of logic ending (starting) at the input (output) pin.

With tags, there may be several different arrival (required) times with respect to the same clock, and a unique tag is used to distinguish between them. The value of this is in modeling false or multi-cycle exceptions which span the boundary of the block. Suppose there are two partial paths, A and B, starting outside the block and ending at an input pin, and two partial paths, C and D, starting at the input pin and ending inside the block. If the full path consisting of A and C is false, but the other three paths (A+D, B+C, B+D) are not, the normal approaches to describing arrival times and setup/hold arcs break down, because they describe the worst case across all partial paths.

Tags are used to distinguish between different classes of partial paths. In the example, A and B would be modeled as two separate arrival times with different tags. Then a false path constraint would be specified for the combination of A's tag and path C.

**Removed from after section**: 7.5 "Common Timing Command Conventions"

**Justification for rejection**: tags are not well-understood at this point and only just now being investigated by EDA vendors and designers. It is also not known if designers desire this feature. It is likely that tags will be re-visited in a later version of this specification.

# Annex D

(informative)

# DCDL Relationship to OLA

This annex provides a mapping between DCDL commands and related Open Library API (OLA) functions.

**Table 4-4—Related DCDL and OLA Commands**

| DCDL Command | Related OLA Function(s) | Notes |
|---|---|---|
| functional_mode | | If *-all* is used, that is an error. |
| operating_point | *** calc mode *** | If *-vaule* is used, that is an error. |
| operating_process | *** calc mode *** | If *-vaule* is used, that is an error |
| operating_range | appGetCurrentOpRange | |
| operating_temperature | *** calc mode *** | If *-vaule* could be optional depending on lib. |
| operating_voltage | *** calc mode *** | If *-vaule* could be optional depending on lib. |
| power_regime | dpcmGetRailVoltageRange | |
| temperature_regime | | |

The DCDL operating condition commmands can represent correlated or uncorrelated values (refer to page 65) based on process. This corresponds to OLA *DCM_ProcessVariations*. The application would set the *PROCESS_VARIATION* field in the standard structure to:

*DCM_NoVariation* for uncorrelated variation

*DCM_MinEarly_MaxLate* for correlated variation

*DCM_MaxEarly_MinLate_EdgesSame* or *DCM_MaxEarly_MinLate_EdgesOpposite* for common path pessimism removal in clock paths with correlated variation

## B



## C



## D

## P



## R



## S



## T



## U



## V



## W