cadence®

Setting Constraints and Performing Timing Analysis Using Encounter® RTL Compiler

Product Version 9.1
April 2010

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

Preface

About This Manual

This manual describes timing analysis in RTL Compiler.

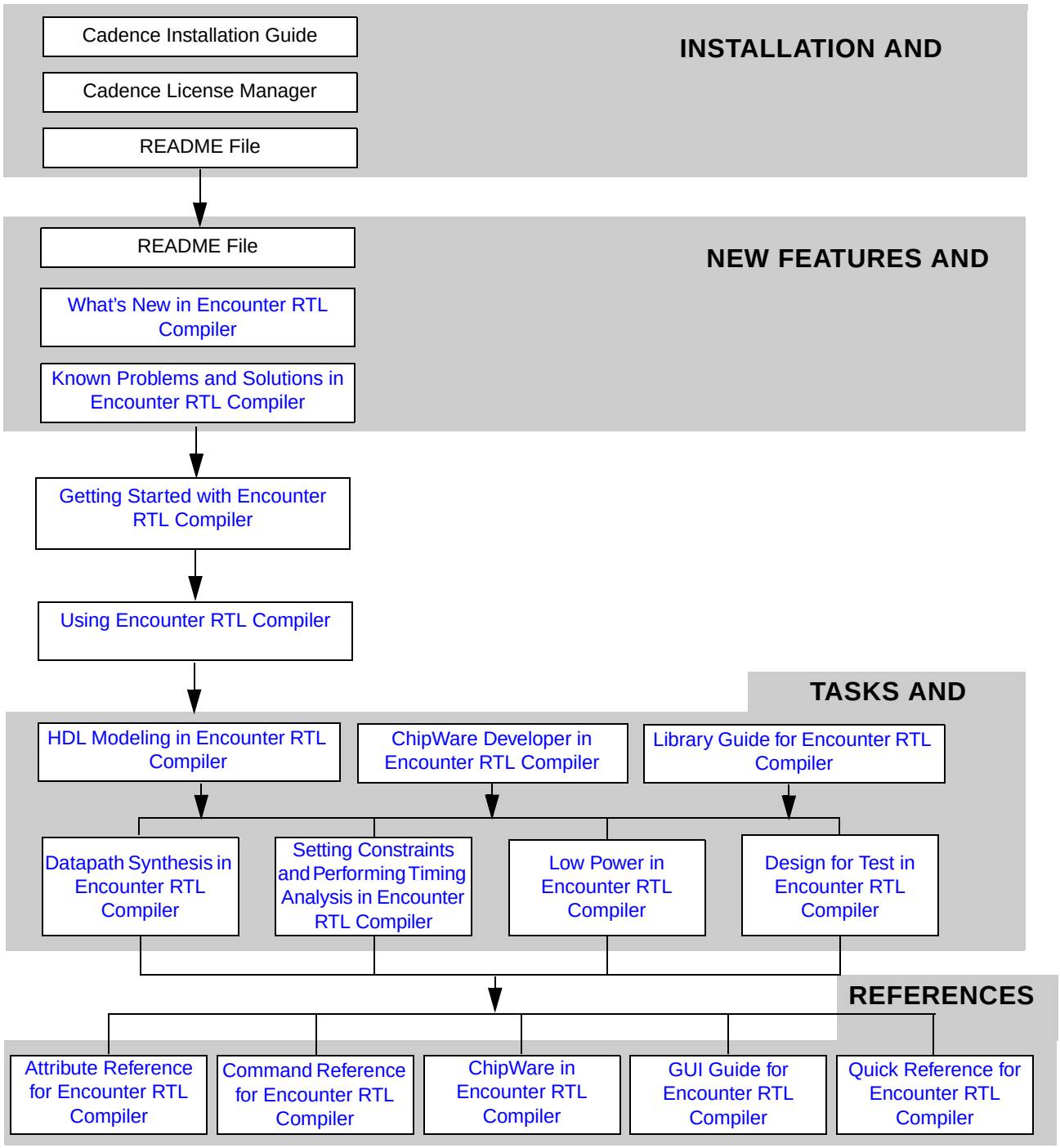
Additional References

The following sources are helpful references, but are not included with the product documentation:

- TclTutor, a computer aided instruction package for learning the Tcl language: http://www.msen.com/~clif/TclTutor.html.
- TCL Reference, *Tcl and the Tk Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company
- IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std.1364-1995)
- IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std. 1364-2001)
- IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1987)
- IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1993)

Note: For information on purchasing IEEE specifications go to http://shop.ieee.org/store/ and click on *Standards*.

How to Use the Documentation Set



Reporting Problems or Errors in Manuals

The Cadence® Help online documentation lets you view, search, and print Cadence product documentation.

➤ Access Cadence help by typing `cdnshelp` from your Cadence tools hierarchy.

Contact Cadence Customer Support to file a PCR if you find

- An error in the manual
- Any missing information in a manual
- A problem using the Cadence Help documentation system

Customer Support

Cadence offers live and online support, as well as customer education and training programs.

Cadence Online Support

The Cadence® online support website offers answers to your most common technical questions. It lets you search more than 40,000 FAQs, notifications, software updates, and technical solutions documents that give you step-by-step instructions on how to solve known problems. It also gives you product-specific e-mail notifications, software updates, service request tracking, up-to-date release information, full site search capabilities, software update ordering, and much more.

For more information on Cadence online support go to:

http://support.cadence.com

Other Support Offerings

- **Support centers**—Provide live customer support from Cadence experts who can answer many questions related to products and platforms.
- **Software downloads**—Provide you with the latest versions of Cadence products.
- **Education services**—Offers instructor-led classes, self-paced Internet, and virtual classroom.

- **University software program support**—Provides you with the latest information to answer your technical questions.

For more information on these support offerings go to:

http://www.cadence.com/support

Messages

From within RTL Compiler there are two ways to get information about messages.

- Use the `report messages` command.

  For example:

  ```
  rc:/> report messages
  ```

  This returns the detailed information for each message output in your current RTL Compiler run. It also includes a summary of how many times each message was issued.

- Use the `man` command.

  Note: You can only use the `man` command for messages within RTL Compiler.

  For example, to get more information about the “TIM-11” message, type the following command:

  ```
  rc:/> man TIM-11
  ```

If you do not get the details that you need or do not understand a message, either contact Cadence Customer Support to file a PCR or email the message ID you would like improved to:

rc_pubs@cadence.com

Man Pages

In addition to the Command and Attribute References, you can also access information about the commands and attributes using the man pages in RTL Compiler. Man pages contain the same content as the Command and Attribute References. To use the man pages from the UNIX shell:

1. Set your environment to view the correct directory:

   ```
   setenv MANPATH $CDN_SYNTH_ROOT/share/synth/man
   ```

2. Enter the name of the command or attribute that you want either in RTL Compiler or within the UNIX shell. For example:

   - `man check_dft_rules`
   - `man cell_leakage_power`

You can also use the `more` command, which behaves like its UNIX counterpart. If the output of a manpage is too small to be displayed completely on the screen, use the `more` command to break up the output. Use the spacebar to page forward, like the UNIX `more` command.

```
rc:/> more man synthesize
```

Command-Line Help

You can get quick syntax help for commands and attributes at the RTL Compiler command-line prompt. There are also enhanced search capabilities so you can more easily search for the command or attribute that you need.

Note: The command syntax representation in this document does not necessarily match the information that you get when you type `help` *`command_name`*. In many cases, the order of the arguments is different. Furthermore, the syntax in this document includes all of the dependencies, where the help information does this only to a certain degree.

If you have any suggestions for improving the command-line help, please e-mail them to:

rc_pubs@cadence.com

Getting the Syntax for a Command

➤ Type the `help` command followed by the command name.

For example:

```
rc:/> help path_delay
```

This returns the syntax for the `path_delay` command.

Getting the Syntax for an Attribute

➤ Type the following:

```
rc:/> get_attribute attribute name * -help
```

For example:

```
rc:/> get_attribute max_transition * -help
```

This returns the syntax for the `max_transition` attribute.

Searching for Attributes

➤ Get a list of all the available attributes by typing the following command:

```
rc:/> get_attribute * * -help
```

➤ Type a sequence of letters after the `set_attribute` command and press `Tab` to get a list of all attributes that contain those letters. For example:

```
rc:/> set_attr li
ambiguous "li": lib_lef_consistency_check_enable lib_search_path libcell
liberty_attributes libpin library library_domain line_number
```

Searching For Commands When You Are Unsure of the Name

You can use help to find a command if you only know part of its name, even as little as one letter.

■ If you only know the first few letters of a command, then you can get a list of commands that begin with that letter.

For example, to get a list of commands that begin with “ed”, you would type the following command:

```
rc:/> ed* -h
```

■ Type a single letter and press `Tab` to get a list of all commands that start with that letter.

For example:

```
rc:/> c <Tab>
```

This returns the following commands:

```
ambiguous "c": cache_vname calling_proc case catch cd change_names
check_dft_rules chipware clear clock clock_gating clock_ports close cmdExpand
command_is_complete concat configure_pad_dft connect_scan_chains continue
cwd_install ...
```

- You can also type a sequence of letters and press `Tab` to get a list of all commands that start with those letters.

  For example:

  ```
  rc:/> path_ <Tab>
  ```

  This returns the following commands:

  ```
  ambiguous "path_": path_adjust path_delay path_disable path_group
  ```

Documentation Conventions

Text Command Syntax

The list below defines the syntax conventions used for the RTL Compiler text interface commands.

`literal`	Nonitalic words indicate keywords that you must type literally. These keywords represent command, attribute or option names
`arguments and options`	Words in italics indicate user-defined arguments or options for which you must substitute a name or a value.
|	Vertical bars (OR-bars) separate possible choices for a single argument.
`[ ]`	Brackets denote options. When used with OR-bars, they enclose a list of choices from which you can choose one.
`{ }`	Braces denote arguments and are used to indicate that a choice is required from the list of arguments separated by OR-bars. You must choose one from the list `{ argument1 | argument2 | argument3 }` Braces, used in Tcl command examples, indicate that the braces must be typed in.
`...`	Three dots (...) indicate that you can repeat the previous argument. If the three dots are used with brackets (that is, `[argument]...`), you can specify zero or more arguments. If the three dots are used without brackets (`argument...`), you must specify at least one argument, but can specify more.
#	The pound sign precedes comments in command files.

1

Specifying Timing and Design Constraints

Overview of Timing and Design Constraints

Figure 1-1 on page 25 shows the timing analysis flow. This chapter focuses on setting timing and design constraints. Timing and design constraints describe the “design intent” and the surrounding constraints, including synthesis, clocking, timing, environmental, and operating conditions.

Set these constraints on startpoints and endpoints to make sure that every path is properly constrained to obtain an optimal implementation of the RTL design. A path begin point is from either an input port or a register clock pin, while an endpoint is either an output port or a register data pin.

Use these constraints to

- Describe different attributes of clock signals, such as the duty cycle, clock skew, and the clock latency
- Specify input and output delay requirements of all ports relative to a clock transition
- Apply environmental attributes, such as load and drive strength to the top-level ports
- Set timing exceptions, such as multicycle paths and false paths

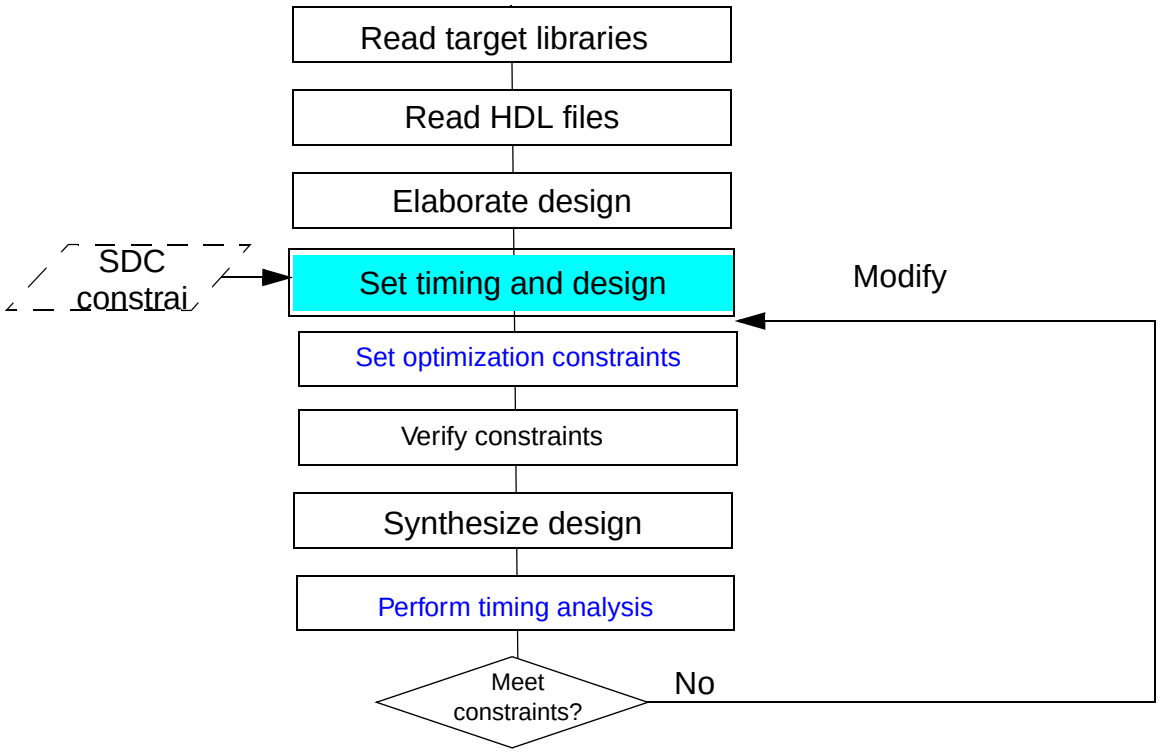
The SDC equivalent command is provided for each constraint. SeeImporting and Exporting SDC Constraints, Using SDC Commands Interactively within RTL Compiler, Mixing RTL Compiler Commands in an SDC File, and DC to RTL Compiler Equivalency Commands for detailed information.

Note: Some of the SDC examples in this chapter contain native RTL Compiler commands. Although this is not standard SDC, it is permissible within RTL Compiler.

Along with the Supported SDC Commands, RTL Compiler supports the `-mode` option for SDC commands that define clocks, external delays, and exceptions (mode specific constraints). For more information, see Chapter 5, “Performing Multi-Mode Timing Analysis.”

RTL Compiler uses picoseconds and femtofarads as timing units. It *does not* use nanoseconds and picofarads.
By default, the time and load units when using the `read_sdc` command, and when using SDC commands interactively, are the units of the first technology library. See Setting SDC Time Units That Are Consistent With RTL Compiler Units for more information.

Figure 1-1 Timing Analysis Flow in Single Mode



A mode signifies the functional or operational behavior of a design. Modes are controlled by a set of constraints that constrain the design and drive timing analysis. For information on performing timing analysis in multi-modes, see Chapter 5, “Performing Multi-Mode Timing Analysis.”

Using SDC Commands

Importing and Exporting SDC Constraints

RTL Compiler supports reading and writing SDC Tcl constraints.

➤ To import SDC constraints, type the following command:

```
rc:/> read_sdc filename
```

The `read_sdc` command supports files that have been compressed with GNU Zip. The following command will unzip the file and then import the SDC constraints:

```
rc:/> read_sdc sdc_constraints.gz
```

➤ To export SDC constraints, type the following command:

```
write_sdc [-dc] [-version {1.1|1.3|1.4|1.5|1.5rc}] [design] [> filename]
```

Default: `1.4`

- ❑ If you specify the `-dc` option, then the constraints are written out in the Design Compiler format instead of Tcl.
- ❑ If you do not specify a design, then RTL Compiler exports the SDC constraints of the current design.
- ❑ If you do not specify a *filename*, then the constraints are written to standard out.
- ❑ If you specify the `.gz` format, then the SDC constraints are compressed with GNU Zip. The following example compresses the exported SDC constraints file:

  ```
  rc:/> write_sdc top > top_constraints.gz
  ```

Using SDC Commands Interactively within RTL Compiler

Interactively use SDC commands at the RTL Compiler prompt by adding the "`dc::`" prefix before any SDC command.

- To use SDC commands with the `dc::` prefix, use the `dc::`*`sdc_command`* form. For example:

  ```
  rc:/> dc::set_input_delay 0.5 [dc::all_inputs]
  ```

- To get help for a SDC command, use the `dc::` prefix and the `-help` option to return the syntax for a specific SDC command. For example:

  ```
  rc:/> dc::set_clock_latency -help
  ```

- To apply mode specific constraints with the `dc::` prefix in multi-mode timing analysis, use the `-mode` option with SDC Mode-Specific Timing Commands. For example:

  ```
  rc:/> dc::set_input_delay -mode mode_a 0.5 [dc::all_inputs]
  ```

  For more information, see Chapter 5, "Performing Multi-Mode Timing Analysis."

Important

When you are mixing SDC and RTL Compiler commands, be aware that the units for capacitance and delay are different. For example, the following SDC `set_load` command expects the load in pF but the RTL Compiler `get_attribute` command will return the load in fF:

```
rc:/> dc::set_load [get_attribute load slow/INVX1/A] \
[dc::all_outputs]
```

This causes the capacitance set on all outputs to be off by a factor of 1000.

Setting SDC Time Units That Are Consistent With RTL Compiler Units

By default, the time and load units when using the `read_sdc` command, and when using SDC commands interactively, are the units of the first technology library.

- Override this behavior using the SDC `set_time_unit` (see dc::set_time_unit) and the `set_load_unit` (see dc::set_load_unit) commands.

  Once the SDC units are changed, the new units apply to all subsequent SDC commands on that particular design. These commands apply to the current SDC design, which can be set using the SDC `current_design` command.

For example, when using the `set_time_unit` command with the `-picoseconds` and `the -nanoseconds` options:

- The following example forces the SDC time units to 1 p.s.:

```
rc:/> dc::set_time_unit -picoseconds
```

- The following example forces the SDC time units to 1 n.s.:

```
rc:/> dc::set_time_unit -nanoseconds
```

The SDC `set_load_unit` command works in a similar manner using the `-picofarads` and `-femtofarads` options.

➤ If you want consistency between RTL Compiler's internal units of picoseconds (ps) and femtofarads (fF) and the SDC file that you load, then use the following commands:

```
rc:/> dc::set_time_unit -picoseconds
rc:/> dc::set_load_unit -femtofarads
```

Mixing RTL Compiler Commands in an SDC File

Mixing SDC and native RTL Compiler commands is possible. As shown in Example 1-1, the SDC file mixes RTL Compiler and SDC commands.

Example 1-1 Mixing SDC and Native RTL Compiler Commands

```
set CLK "Clk";
set CLK_PERIOD [expr 2 * 1000.0] ;
define_clock -name $CLK -domain csi -period $CLK_PERIOD [find /des* -port \
*${CLK}*];
set DEFAULT_CLK_IN_DELAY  [expr 0.63 * $CLK_PERIOD];
set_input_delay -clock $CLK [expr $DEFAULT_CLK_IN_DELAY / 1000.0]
[get_ports "xxin*"]
```

See Supported SDC Commands and Unsupported SDC Commands for more information.

Supported SDC Commands

Note: If the `read_sdc` command issues an error when loading a SDC file, then the file may contain unsupported commands. You can edit the SDC file to remove the unsupported commands.

➤ To get help for supported SDC commands, use the `-help` option. For example:

```
rc:/> dc::get_lib_cells -help
```

General

General Commands	Options and Arguments
{arg1 arg2 ...}	
current_instance	see dc::current_instance on page 226
expression	*arg1 arg2 ...*
list	*arg1 arg2 ...*
set variable_name	*value*
set_hierarchy_separator	see dc::set_hierarchy_separator on page 302
source	*file_names*

Object Access

Object Access Commands	Options and Arguments
add_to_collection	
	Note: PT-only
all_clocks	see dc::all_clocks on page 214
all_connected	**Note:** PT-only
all_inputs	see dc::all_inputs on page 215
all_outputs	see dc::all_outputs on page 217
all_registers	seedc::all_registers on page 218
current_design	see dc::current_design on page 225
	Note: non-SDC

	To cd to a particular subdesign, use the follwoing procedure:
	`proc wa_current_design {name} {`
	`cd /des*`
	`set subd [find /des* -subdesign $name]`
	`cd $subd }`

current_instance	seedc::current_instance on page 226

Object Access Commands	Options and Arguments
`foreach_in_collection`	
	Note: PT-only
`get_cells or get_cell`	see dc::get_cell on page 228
	Note: PT-only
`get_clocks or get_clock`	see dc::get_clock on page 231
	Note: PT-only
`get_designs or get_design`	seedc::get_design on page 233
	Note: PT-only
`getenv`	see cdc::getenv on page 257
`get_generated_clocks`	see dc::get_generated_clocks on page 235
	Note: PT-only
`get_libs or get_lib`	see dc::get_lib on page 238
	Note: PT-only
`get_lib_cells`	see dc::get_lib_cell on page 240
	Note: PT-only
`get_lib_pins`	seedc::get_lib_pin on page 242
	Note: PT-only
`get_nets or get_net`	see `dc::get_net` on page 247
	Note: PT-only
`get_pins or get_pin`	see `dc::get_pin` on page 252
	Note: PT-only
`get_ports or get_port`	see `dc::get_port` on page 255
	Note: PT-only
`get_references`	`[-hierarchical] [-quiet] [-regexp] [-nocase] [-exact][-filter` *`filter_expr`*`]` *`patterns`*
	Note: Non-SDC

Object Access Commands	Options and Arguments
group_path	see dc::group_path on page 258
remove_from_collection	**Note:** PT-only
remove_ideal_net	see dc::remove_ideal_net on page 266
remove_ideal_network	see dc::remove_ideal_network on page 267
remove_input_delay	see dc::remove_input_delay on page 268
remove_output_delay	see dc::remove_output_delay on page 269

Design Rule Constraints

Design Rule Commands	Options and Arguments
set_max_capacitance	see dc::set_max_capacitance on page 317
set_max_fanout	seedc::set_max_fanout on page 322
set_max_transition	see dc::set_max_transition on page 325

Timing Constraints

Timing Command	Options and Arguments
create_clock	see *dc::create_clock* on page 220
create_generated_clock	see *dc::create_generated_clock* on page 222
	Specify the -add and the -master_clock options together.
group_path	see dc::group_path on page 258
set_case_analysis	see *dc::set_case_analysis* on page 270
set_clock_gating_check	see dc::set_clock_gating_check on page 271
	set dc::enable_set_clock_gating_check 1
set_clock_groups	see dc::set_clock_groups on page 273
set_clock_latency	see *dc::set_clock_latency* on page 275
set_clock_sense	see dc::set_clock_sense on page 278
set_clock_skew	see *dc::set_clock_skew* on page 280
	[-ideal] [-minus_uncertainty *minus_value*] *clocks ports*
set_clock_transition	see *dc::set_clock_transition* on page 282
set_clock_uncertainty	see dc::set_clock_uncertainty on page 283
set_data_check	see dc::set_data_check on page 286

Timing Command	Options and Arguments
set_disable_timing	[-from *from_pin_name*] [-to *to_pin_name*] *cells*|*pins*|*lib_cells*|*lib_pins* **Note:** You can specify the -from option without specifying the -to option.
set_dont_touch	see dc::set_dont_touch on page 289 object_list designs current_design cells nets lib_cells [true | false] *Default*: true Default behavior applies it to the current_design if no arguments are specified. **Note:** Non-SDC
set_dont_touch_network	see dc::set_dont_touch_network on page 290
set_dont_use	see dc::set_dont_use on page 291 **Note:** Non-SDC
set_drive	see dc::set_drive on page 292
set_driving_cell	see dc::set_driving_cell on page 294
set_equal	see dc::set_equal on page 297
set_fanout_load	see dc::set_fanout_load on page 301
set_ideal_network	see dc::set_ideal_network on page 304
set_ideal_net	see dc::set_ideal_net on page 303
set_input_delay	see dc::set_input_delay on page 305
set_input_transition	see dc::set_input_transition on page 308
set_load	see dc::set_load on page 311
set_logic_dc	see dc::set_logic_dc on page 314
set_logic_one	seedc::set_logic_one on page 315
set_logic_zero	see dc::set_logic_zero on page 316
set_max_dynamic_power	see dc::set_max_dynamic_power on page 321
set_max_leakage_power	see dc::set_max_leakage_power on page 323

Timing Command	Options and Arguments
set_max_time_borrow	see dc::set_max_time_borrow on page 324
set_operating_conditions	see dc::set_operating_conditions on page 334
set_opposite	see dc::set_opposite on page 335
set_output_delay	see dc::set_output_delay on page 336
set_port_fanout_number	see dc::set_port_fanout_number on page 342
set_unconnected	see dc::set_unconnected on page 346
set_wire_load_mode	see dc::set_wire_load_mode on page 348
set_wire_load_model	seedc::set_wire_load_model on page 349
set_wire_load_selection_group	see dc::set_wire_load_selection_group on page 351

Timing Exceptions

Timing Exception Commands	Options and Arguments
set_false_path	see dc::set_false_path on page 298
set_max_delay	see dc::set_max_delay on page 318
set_min_delay	see dc::set_min_delay on page 327

set_multicycle_path	see dc::set_multicycle_path on page 331
	Note: If you have two multicycle paths with different path multipliers, then RTL Compiler uses the tightest specification for SDC set_multicycle_path constraints and the last specification for RTL Compiler multicycle_path constraints. In the following example, the first set multicycle_path specification will be used because it is the tightest:
	`rc:/> dc::set_multicycle_path -setup -from [get_ports {in2}] \`
	`-to [get_pins {temp3_reg/D}] 3.0`
	`rc:/> dc::set_multicycle_path -setup -from [get_ports {in2}] \`
	`-to [get_pins {temp3_reg/D}] 5.0`

Unsupported SDC Commands

- `derive_clocks`
- `filters`
- `filter_collection`
- `find`
- `get_references`
- `get_unix_variable`
- `remove_attribute`
- `set_annotated_check`
- `set_capacitance`
- `set_connection_class`
- `set_critical_range`
- `set_drive_resistance`
- `set_fix_multiple_port_nets`
- `set_ideal_latency`
- `ste_ideal_transition`
- `set_local_link_library`
- `set_max_area`
- `set_min_capacitance`
- `set_min_delay`

  Note: The `read_sdc` command can read in the `set_min_delay` command and write it back out, but it will have no effect in RTL Compiler's timing analysis, power analysis, and optimization.

- `set_min_porosity`
- `set_min_transition`
- `set_propagated_clock`
- `set_resistance`
- `set_wire_load_min_block_size`
- `suppress_message`
- `unsuppress_message`

Note: RTL Compiler will not issue multiple warnings about the same unsupported SDC. In that case, only one warning will be issued:

```
Warning : One or more SDC commands failed and these constraints were not applied. [SDC-209]
        : You can examine the failed commands or save them to a file by querying the Tcl
variable $::dc::sdc_failed_commands.
```

Writing out Unsupported SDCs that Were Originally Read-In

On some occasions unsupported SDCs, which were originally read in, are omitted when writing out the SDCs. Check for error messages when using the `read_sdc` command. If an error occurs when applying an SDC constraint, then the intent of that constraint may be lost. If a constraint is not understood, then RTL Compiler reports an error.

All unsupported commands will be treated as failed commands. RTL Compiler will not stop when it sees an unsupported command, but issues a warning message. For example, the following warning message is issued when a failure is encountered at the end of a file:

```
Warning : Could not interpret SDC command. [SDC-202]
        : (corners.sdc:84) set_driving_cell -rise
```

The following warning message is issued if any commands failed pointing you to the variable that contains a list of all the failed commands encountered:

```
Warning : One or more SDC commands failed and these constraints were not applied. [SDC-209]
        : You can examine the failed commands or save them to a file by querying the Tcl
variable $::dc::sdc_failed_commands.
```

- Use the `read_sdc -stop_on_errors` command to specify that RTL Compiler stop reading a file or script when an error is encountered.

Supported Liberty timing_type Values

- `combinational`
- `combinational_rise`
- `combinational_fall`
- `falling_edge`
- `minimum_period`
- `nochange_high_high`
- `nochange_high_low`
- `nochange_low_high`
- `nochange_low_low`
- `non_seq_setup/hold`
- `recovery_rising`
- `recovery_falling`
- `rising_edge`
- `setup_rising`
- `setup_falling`
- `skew_rising/falling`
- `three_state_disable`
- `three_state_enable`

The following `timing_type` values are not supported:.

- `hold_rising`
- `hold_falling`
- `preset`
- `clear`
- `removal_rising`

DC to RTL Compiler Equivalency Commands

The following tables provide a list of the supported Synopsys Design Compiler (DC) commands and the equivalent RTL Compiler (RC) commands.

Applying Constraints

Synopsys Design Compiler	Encounter RTL Compiler
all_clocks	find / -clock *
all_designs	find /designs -design *
all_fanin	fanin
all_fanout	fanout
all_inputs	dc::all_inputs
	all des inps
all_outputs	dc::all_outputs
	all des outs
all_registers	dc::all_registers
	all des seqs
cd	cd
change_names	change_names
check_timing	report timing -lint
create_clock	define_clock
create_generated_clock	supported SDC command
current_design	cd
current_instance	dc::current_instance
derive_constraints	derive_environment

Synopsys Design Compiler	Encounter RTL Compiler
filter	filter
get_attribute	get_attribute
get_cells	find / -instance
get_clocks	find / -clock
get_designs	find / -design -subdesign
get_lib_cells	find / -libcell
get_lib_pins	find / -libpin
get_libs	find / -library
get_nets	find / -net
get_pins	find / -pin
get_ports	find / -port
group_path	path_group
help	help
include	include
list_instances	find / -instance *
list_libs	find / -library *
ls	ls
quit	exit or quit
read_lib	set_attribute library
read_sdc	read_sdc
read_verilog	read_hdl
read_vhdl	read_hdl -vhdl
remove_cell	rm [find / -instance inst]
remove_clock	rm [find / -clock clk]
remove_constraint	rm [find / -exception excep]
set_case_analysis	set_att timing_case_logic_value
set_attribute	set_attribute

Synopsys Design Compiler	Encounter RTL Compiler
set_case_analysis_sequential_propagation	set_attribute case_analysis_sequential_propagation
set_clock_latency -source -early	set_attribute clock_source_early_latency
set_clock_latency -source -late	set_att clock_source_late_latency
set_clock_latency -early	set_attribute clock_network_early_latency
set_clock_latency -late	set_attribute clock_network_late_latency
set_clock_uncertainty	set_att clock_hold_uncertainty
	set_att clock_setup_uncertainty
set_disable_timing	set_attribute disabled_arcs
set_dont_touch	set_attribute preserve
set_dont_touch_network	set_attribute inherited_preserve
set_dont_use	set_attribute avoid
set_drive_cell	set_attribute external_driver
set_drive	set_attribute external_resistance
set_false_path	path_disable
set_fanout_load	set_attribute fanout_load
set_input_delay	external_delay -input
set_load	set_attribute external_pin_cap
set_max_capacitance	set_attribute max_capacitance
set_max_transition	set_attribute max_transition
set_multicycle_path	multi_cycle
set_operating_conditions	set_attribute operating_conditions
set_output_delay	external_delay -output
set_port_capacitance	set_attribute external_pin_cap

Synopsys Design Compiler	Encounter RTL Compiler
set_port_fanout_number	set_att external_wireload_fanout
set_wire_load_mode	set_attribute wireload_mode
set_wire_load_model	set_attribute wireload_model
set_wire_load_model_selection_group	set_attribute wireload_selection
sub_designs_of	get_attribute subdesign *hier_instance_name*
sub_instances_of	get_attribute instances *subdesign_name*
suppress_message	suppress_message
uniquify	edit_netlist uniquify

Analyzing Results

Synopsys Design Compiler	Encounter RTL Compiler
drive_of	get_att drivers *net_object*
report_area	report area
report_constraint	no equivalent
report_design	report design_rules
report_disable_timing	report timing -exceptions
report_hierarchy	report hierarchy
report_net_fanout	no equivalent
report_path_group	report timing -cost_group
report_timing	report timing
report_wire_load	no equivalent

Saving the Design

Synopsys Design Compiler	Encounter RTL Compiler
write	write_hdl
write_rtl	write_hdl
write_sdc	write_sdc
write_script	write_script

DC to RTL Compiler Commands with Varying Default Units

Synopsys Design Compiler	Encounter RTL Compiler
create_clock	define_clock
set_clock_latency -source -early	set_attribute clock_source_early_latency
set_clock_latency -source -late	set_att clock_source_late_latency
set_clock_latency -early	set_attribute clock_network_early_latency
set_clock_latency -late	set_attribute clock_network_late_latency
set_clock_transition	set_attribute slew_rise set_attribute slew_fall
set_clock_uncertainty	path_adjust
set_clock_skew	path_adjust
set_driving_cell	set_attribute external_driver_input_slew *fall* set_attribute external_driver_input_slew *rise*

Specifying Clock Information

Clocks are devices used to provide timing and synchronization information. You must specify one or more clocks for sequential designs. You can specify the following clock information:

- Period and waveform

  Global reference signal for all signals in the design

- Insertion delay (latency)

  Delay from the clock source

- Uncertainty (skew and jitter)

  Amount of skew or variation in the arrival of clock edges

- Transition time (slew time)

  Amount of time it takes for a signal to change from one logic state to another

Specify clock information for all clocks used or referenced in the design. This section describes the following clock information:

Generate reports to view the timing constraints and the results of applying them. See Generating Post-Synthesis Timing Reports on page 134.

Defining a Clock

A *clock* is a periodic waveform with a fixed frequency that is used to sequence the operation of a circuit. A clock sets the time allowed for signals to propagate between sequential elements.

In RTL Compiler, a clock waveform is a periodic signal with one rising edge and one falling edge per period. Clock waveforms may be applied to design objects, such as input ports, clock pins of sequential cells, external clocks (also known as virtual clocks), mapped cells, or hierarchical boundary pins.

➤ Define clocks using the `define_clock` command.

Note: If you define two different clocks on the same pin, then RTL Compiler will accept both of them and perform timing analysis by considering the worst-case scenario.

Clocks you define have waveforms that ignore the delay effects of the clock network. See Specifying Clock Latency (Insertion Delay) on page 49 for more information.

Important

RTL Compiler automatically assigns a cost group for every defined clock that was created either through the `create_clock` or the `create_generated_clock` command in an SDC file. However, RTL Compiler does *not* automatically create a `cost_group` for each clock defined if the RTL Compiler `define_clock` command is used. By default, RTL Compiler will focus on the worst slack path in each of these cost groups and devote equal optimization efforts to each of these paths. This can result in an inefficient use of optimization cycles. See Creating Path Groups and Cost Groups on page 98 for more information.

Clock waveforms, which are not applied to objects in your design, are external clocks and are only used as references for external delay values.

To define clocks in a multi-mode environment, see Chapter 5, “Performing Multi-Mode Timing Analysis.”

Defining the Clock Period

The *clock period*, also known as cycle-time, describes the interval at which the waveform repeats itself. Sequential logic devices can only change outputs once a period, usually during a positive or a negative edge.

To determine the timing constraints for paths between different clocks, RTL Compiler must know the exact relationship between the rising and falling edges of the two waveforms. By default, RTL Compiler assumes that the rising edge of a clock waveform is at the start of the period and the falling edge is halfway through the period.

- Specify the clocks using the `define_clock` command.

  The SDC equivalent command is `create_clock`.

  The `define_clock` command returns the directory path to the clock object.

- Define a clock of 100MHz that should be applied to all clock input ports of the design:

  ```
  define_clock -period 10000 -name 100MHz [clock_ports]
  ```

  where `clock_ports` returns the input ports of your design that are clock inputs.

- Save the directory path to the clock that is defined (100MHz) in variable `clock1` for future reference:

  ```
  rc:/> set clock1 [define_clock -period 10000 -name 100MHz [clock_ports]]
  ```

- Search for this clock later using the `find` command:

  ```
  rc:/> find / -clock 100MHz
  ```

The following example defines two external clocks, as shown in Figure 1-2 on page 46. One clock, called `100MHz`, has a period of `10,000` picoseconds. The other clock, called `300MHz`, applies to all clock ports and goes through three cycles for every one of the first clock's:

```
rc:/> define_clock -name 100MHz -period 10000
rc:/> define_clock -name 300MHz -period 10000 -divide_period 3 [clock_ports]
```

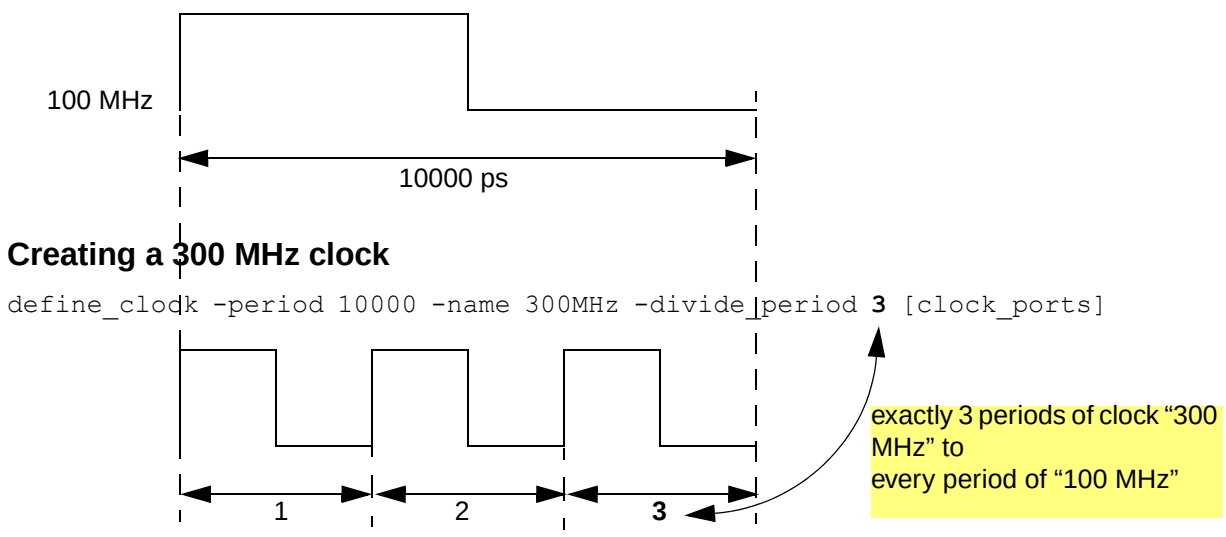
The period for the `300MHz` clock is 10,000/3 picoseconds.

Note: There is no SDC equivalent for the `-divide_period` option.

RTL Compiler recognizes that there are exactly three periods of clock `300MHz` to every period of clock `100MHz`. If you specified a period of `3,333` picoseconds for the `300MHz` clock, then RTL Compiler computes a different relationship between the clocks (`3,333` periods of one clock to 10000 periods of the other) and the timing analysis of the design would be different.

The 100 MHz and 300 MHz clocks are shown in Figure 1-2.

Figure 1-2 Creating 100 and 300 MHz Clocks



Defining the Rising and Falling Edges

The *rising edge* is a transition from 0 to 1 or a positive edge and the *falling-edge* is a transition from 1 to 0 or a negative edge. Data is loaded when a transition or edge occurs on the clock input.

Setting the rising and falling edges lets you precisely define when the clock transitions occur. In some cases, a precise definition is required for RTL Compiler to compute the correct timing constraints for paths that are launched by one clock and captured by another.

➤ Define the rising and falling edges for a clock using the `-rise`, `-fall`, `-divide_rise`, and `-divide_fall` options with the `define_clock` command.

The following examples show how to use these options.

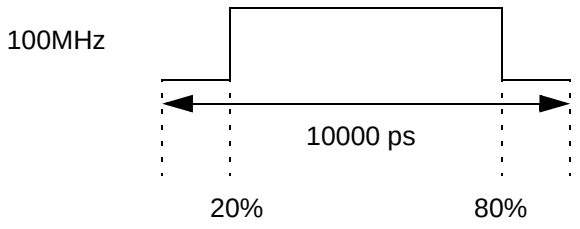
■ The following command specifies a 100 MHz clock to have a rising edge after 20 percent of the period and a falling edge after 80 percent, as shown in Figure 1-3 on page 47:

```
rc:/> define_clock -name 100MHz -period 10000 -rise 20 -fall 80
```

The following is the SDC equivalent command:

```
rc:/> dc::create_clock -name 100Mhz -period 10 -waveform {2 8}
```

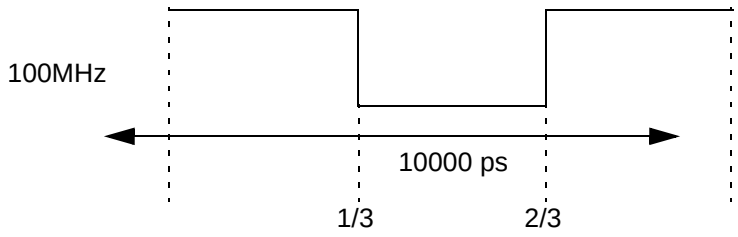
Figure 1-3 Defining the Rising and Falling Edges



- The following command specifies that the falling edge is one third and the rising edge is two thirds of the way through the period, as shown in Figure 1-4:

```
rc:/> define_clock -name 100MHz -period 10000 -rise 2 -divide_rise 3 -fall 1
-divide_fall 3
```

Figure 1-4 Changing the Rise and Fall Times



Creating Clock Domains

A *clock domain* is a group of clocks that are synchronous to each other, allowing timing analysis to be performed between these clocks.

RTL Compiler only computes timing constraints among clocks in the same clock domain. Paths between clocks in different domains are unconstrained by default.

➤ Constrain these paths using the `path_delay` command.

When clocks are defined in different clock domains they are considered to be asynchronous, that is, not constrained relative to each other. RTL Compiler unconstrains different clock domains automatically; you do not need to specify the false paths for the clocks.

Note: If you manually specify false paths and the primary input clock pins and the clock objects have the same names, then use the `find` command with the `-clock` option to refer

to the clock objects and not the clock pins. Otherwise, the ambiguity will cause RTL Compiler to synthesize the paths, since it does not recognize them as false paths.

If a clock domain is not specified, then RTL Compiler will assume all the clocks are in the same domain.

➤ By default, RTL Compiler assigns clocks to `domain_1`, but you can create your own domain name using the `-domain` argument with the `define_clock` command.

The following example demonstrates how to create two different clocks and assign them to two separate clock domains:

```
rc:/> define_clock -domain domain1 -name clk1 -period 720 [find / -port SYSCLK]
rc:/> define_clock -domain domain2 -name clk2 -period 720 [find / -port CLK]
```

The SDC equivalent commands are:

```
rc:/> dc::create_clock -name "clk1" -period 0.72 -waveform {0.0 0.36} \
    [get_ports SYSCLK]
rc:/> dc::create_clock -name "clk2" -period 0.72 -waveform {0.0 0.36} \
    [get_ports CLK]
rc:/> dc::set_false_path -from [get_clocks clk1] -to [get_clocks clk2]
rc:/> dc::set_false_path -from [get_clocks clk2] -to [get_clocks clk1] ...
```

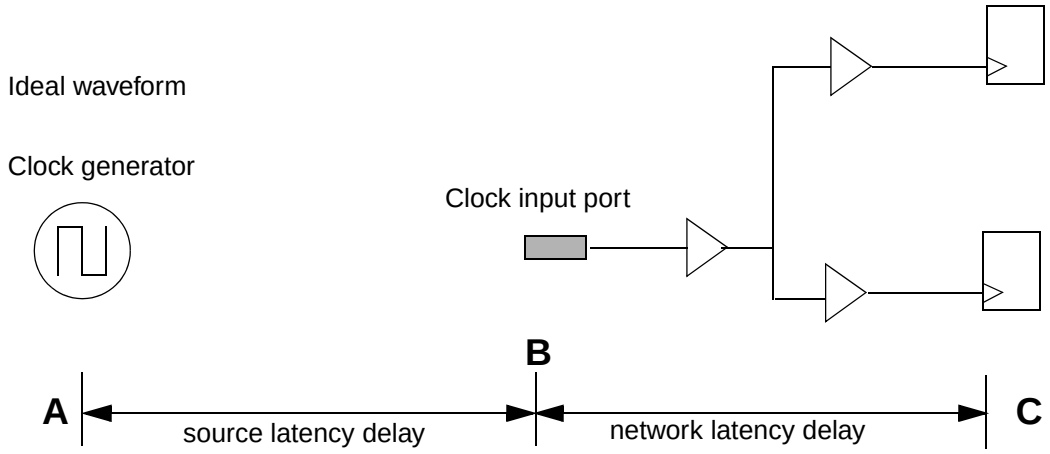
To define a cost group between two clock domains, see Cost Group Examples on page 100

Specifying Clock Latency (Insertion Delay)

*Clock latency (*or *insertion delay)* is the sum of the clock source latency and the clock network latency.

- *Clock source latency* is the time it takes for the clock signal to propagate from the point where the ideal clock waveform is applied to the point in the circuit where the clock is defined (clock input port of the design). Referring to Figure 1-5, the time it takes for the clock signal to get from point A to point B.
- *Clock network latency* (or clock tree delay) is the time it takes for the clock signal to propagate from the clock input port of the design to the first sequential element. Referring to Figure 1-5, the time it takes for the clock signal to get from point B to point C.

Figure 1-5 Clock Latency



RTL Compiler computes timing constraints for a path using the maximum clock latency at the launching clock edge and the minimum clock latency at the capturing clock edge. This produces the tightest possible timing constraints.

➤ Specify clock network latency on a waveform using the `clock_network_early_latency` and the `clock_network_late_latency` attributes. The following example sets the latency to `150` picoseconds:

```
rc:/> set_attribute clock_network_late_latency 150 clk1
```

The SDC equivalent command is:

```
rc:/> dc::set_clock_latency -max .15 clk1
```

Note: The values set on a pin both override the latency values for all clock signals that propagate through the pin and apply to all sequential elements in the transitive fanout.

➤ Specify clock source latency using the clock_source_early_latency and clock_source_late_latency attributes. The following example sets the latency to 150 picoseconds:

```
rc:/> set_attribute clock_source_early_latency 150 clk1
```

The SDC equivalent command is:

```
rc:/> dc::set_clock_latency -source -min .15 clk1
```

Note: You can specify whether RTL Compiler either includes or ignores these latency values with the external_delay command.

Specifying Clock Skew

Clock skew, also called uncertainty, is the maximum difference between the arrival of clock signals at registers in one clock domain or between domains. Clock uncertainty is a general method to add pessimism. A positive uncertainty contributes to pessimism, while a negative uncertainty has no meaning physically, but it makes all slack numbers optimistic. Negative uncertainty can be used to adjust conservative constraints into more realistic ones, which is convenient when the design has paths starting in one clock domain and ending in another clock domain.

You must describe the delay effects of the clock network because the clocks you defined using the define_clock command have waveforms that do not account for this delay.

Skew affects the propagation time. Using the clock's specification, timing analysis determines the amount of time allowed for propagation between registers. The clock skew and flip-flop setup time are subtracted from the clock period to arrive at the time left for propagation. A path that takes longer than this propagation time violates the path's required timing.

Specify clock skew using one of the following methods.

■ Use the following attributes to specify the setup and hold arrival times:

  ❑ Specify the uncertainty in the arrival times of the capturing edges for the clock in late mode (setup) timing analysis using the clock_setup_uncertainty attribute.

  Default: 0 0 {R F} picoseconds

  For example, the following command sets the rise setup uncertainty to 100 picoseconds and the fall setup uncertainty to 50 picoseconds:

  ```
  rc:/> set_attribute clock_setup_uncertainty {100 50} clk1
  ```

  ❑ Specify the uncertainty in the arrival times of the capturing edges for the clock in early-mode (hold) timing analysis using the clock_hold_uncertainty attribute.

  Default: no_value no_value

The following is the SDC equivalent command:

```
rc:/> dc::set_clock_uncertainty -hold .1 clk
```

Note: Using the `clock_setup_uncertainty` and the `clock_hold_uncertainty` attributes are useful in specifying *intra clock* uncertainty, which is uncertainty with respect to one clock. To specify *inter clock* uncertainty, use the SDC `set_clock_uncertainty` command.

- Use the following command to specify clock skew information:

  `clock_uncertainty`

Specifying the Clock Transition

The *clock transition* is when a signal changes from one logic state to another, while *slew* is the amount of time it takes for the signal to change from low to high or from high to low.

➤ Specify the minimum rise, minimum fall, maximum rise, and maximum fall slew values, respectively, in picoseconds using the `slew` attribute.

Default: 0 ps

For example, the following command sets a 100 minimum rise value, a 110 minimum fall value, a 110 maximum rise value, and a 120 maximum fall value on `clock1`:

```
rc:/> set_attribute slew {100 110 110 120} \
/designs/example_design/timing/clock_domains/domain_1/clock1
```

RTL Compiler ignores the minimum values but they can be passed to downstream tools. The slew can affect both the delay through the sequential devices and the setup requirements within them.

Note: If you specify different slew values, then you must do so as a Tcl list (within braces).

The SDC equivalent command is:

```
rc:/> dc::set_clock_transistion .1 clk
```

Defining Externally Generated Clocks

An *external clock*, also known as a virtual clock, is an externally generated clock source that is not associated with any object in the design.Virtual clocks are needed when you want to synthesize a combinational logic block.

Define virtual clocks for signals that interface to external clocked devices. Use a virtual clock as a reference for specifying input and output delays relative to a clock.

➤ Specify the timing for an external clock using the `define_clock` command, as shown in following examples:

```
rc:/> define_clock -period 3000 -name vclock1
rc:/> define_clock -period 2000 -name vclock2
```

The SDC equivalent commands are:

```
rc:/> dc::create_clock -period 3 -name vclock1
rc:/> dc::create_clock -period 2 -name vclock2
```

The `vclock1` clock has a period of `3000` picoseconds and the `vclock2` clock has a period of `2000` picoseconds.

Defining Multiple Clocks on the Same Point

➤ To avoid conflicting clock specifications when using the SDC `create_generated_clock` command, use the `-add` option.

For example, the following clocks conflict because they have the same source points:

```
rc:/> dc::create_generated_clock -name sport1_sclk_generated_out -domain \
      DSP_CLK -source\ vox160/falc_core/pll_top_inst/CLKBUFX20_dsp1_clk/Y \
      divide_by 4 vox160/dsp2_top/logic_mod/sp_mod/p1/sck1/sclko_reg/q
rc:/> dc::create_generated_clock -name sport2_clk_out1 -domain DSP_CLK \
      -source vox160/falc_core/pll_top_inst/CLKBUFX20_dsp2_clk/Y \
      -divide_by 4 vox160/dsp2_top/logic_mod/sp_mod/p1/sck1/sclko_reg/q
```

Define the second clock in the file or to all generated clocks using the `-add` option; otherwise, the first clock defined will have its source points removed by the second clock.

If you define multiple clocks on the same endpoint but they are `false_path` to each other, then RTL Compiler will always use the worst case scenario. For example, if you create a different clock on each input of a mux, but the select line is unconstrained, then RTL Compiler will choose the fastest clock. This also applies when you define multiple clocks on the same pin, such as a programmable divider.

Specifying Timing Setup Checks

RTL Compiler will automatically put a setup check of 0 picoseconds on logic that have the clock and data inputs gated together and for which the clock controls the output.

In the following three figures, the setup check will be added to the first two gates but not the third since the clock does not control the output.

Figure 1-6 and Gate with clk Controlling Output Value

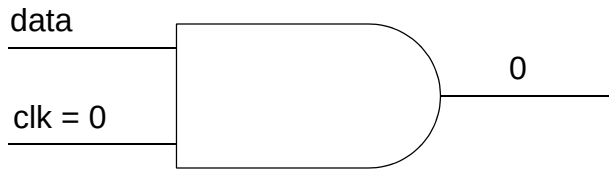


Figure 1-7 or Gate with clk Controlling Output Value

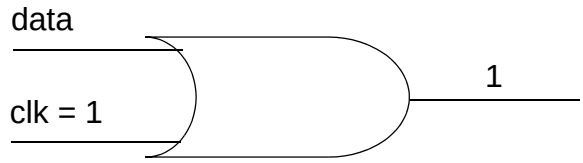
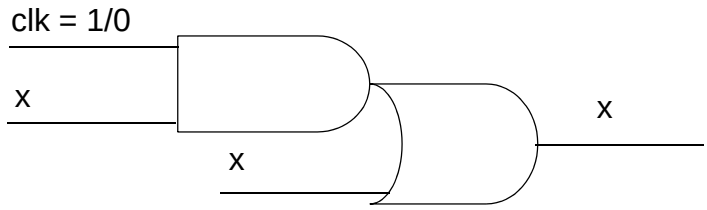


Figure 1-8 andor Gate where clk Cannot Deterministically Determine Output Value



Removing Clocks

When a clock object is removed, external delays that reference it are removed, and timing exceptions referring to the clock are removed if they cannot be satisfied without the clock.

- Remove clocks using the rm command. If you have defined a clock and saved the object variable; for example as clock1, then you can remove the clock object as shown in the following example:

  ```
  rc:/> rm $clock1
  ```

  The SDC equivalent command is:

  ```
  rc:/> dc::remove_clock $clock1
  ```

  The following example shows how to remove the clock if you have not saved the clock object as a variable:

  ```
  rc:/> rm [find / -clock clock_name]
  ```

  The SDC equivalent command is:

  ```
  rc:/> dc::remove_clock [find / -clock clock_name]
  ```

Finding Clock Ports

A *clock port* is a bidirectional signal that acts as either an input to or an output from an electronic circuit and is the clock definition point.

- After defining the clocks, list all the clock input ports in your design using the clock_ports command:

  ```
  rc:/> clock_ports [design]
  ```

  There is no SDC equivalent command for the clock_ports command.

Note: RTL Compiler only returns a list of input ports that connect directly to a sequential clock pin. Clock ports that go through gating elements are not counted by the clock_ports command.

Tip

If you embed the clock_ports command within a define_clock command, then the clock waveform will be applied to all clock input ports of your design. For example, the following command applies a clock period of 3,000 picoseconds to all clock ports:

```
rc:/> define_clock -period 3000 -name clock1 [clock_ports]
```

Reporting Clocks

- Use the `report clocks` command to generate a report on the clocks of the current design.
- Use the `report_clocks -generated` command to report generated clocks.

  See Generating Clock Reports on page 122 for more information.

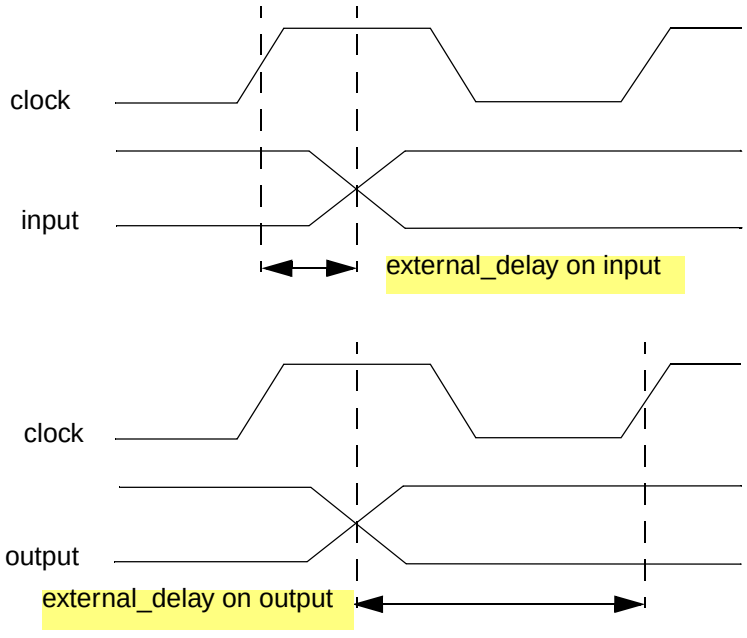
Defining Input and Output Delays

Input delay is the arrival time of external paths at an input port relative to a clock edge, while *output delay* represents the delay of an external timing path from an output port to a register input of the external block. You can define the input and output delays for ports in your design relative to an edge of a previously defined clock.

Delays are set relative to a defined clock waveform. Setting external delays on ports or pins without defining the reference clock waveform is only useful if the port will be constrained by a `path_delay` timing exception, which is rare. A warning is issued when using the `report timing -lint` command if a port has no external delays that reference clocks.

External delays are most often specified on top-level ports of your design. Timing is specified as either an input or output delay, and is specified relative to a clock edge, as shown in Figure 1-9.

Figure 1-9 External Delays on Input and Output Signals



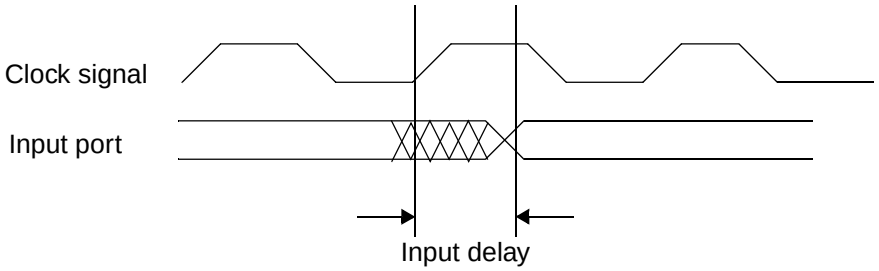
To define input and output delays in a multi-mode environment, see Chapter 5, “Performing Multi-Mode Timing Analysis.”

Setting Input Delays

As shown in Figure 1-10, *input delay* is the delay between a launching clock edge and the time when an input port becomes stable.

You can apply delays to individual bits of multi-bit ports because each bit is accessible individually within the directory structure of the design.

Figure 1-10 Clock/Input Port Timing



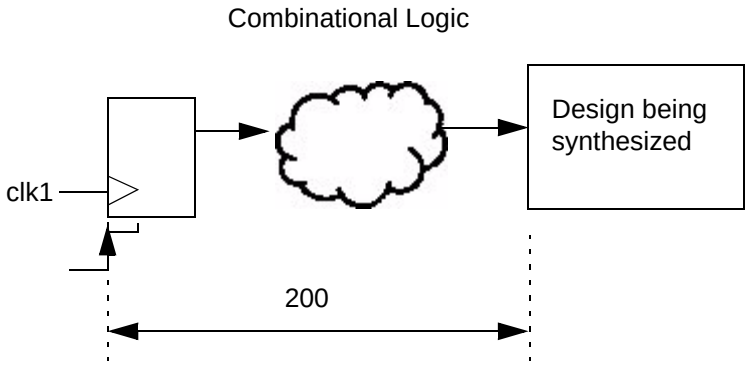
- For example, the following commands create a clock and set the input delay, named `in_con`, to `200` picoseconds, as shown in Figure 1-11 on all input ports:

```
rc:/> define_clock –name clock1 –period 2500
rc:/> external_delay –clock [find / -clock clock1] –input 200 -name in_con \
    [find /des* -port ports_in/*]
```

The equivalent SDC commands are:

```
rc:/> dc::create_clock -period 2.5 -name clock1
rc:/> dc::set_input_delay .2 -clock clock1 [dc::all_inputs]
```

Figure 1-11 Input Delay



Removing an External Delay for a Specific port

You can specify the external_delay with respect to different clocks at any particular port. RTL Compiler keeps all the external_delay constraints if they are referenced to different clocks. If they are referenced to the same clock, then the last one overrides the previous one.

1. To override timing constraints for any input signals with a different timing delay with reference to a different clock, you must first remove the original external_delay, then redefine the external delay. For example, if you want to remove the external_delay to a default clock at a certain port, then the following command removes the original external delay, then re-defines the external delay to all other ports except that port:

```
external_delay -input 500 -clock default {a b c}  ==> this delay is in_delay_1
```

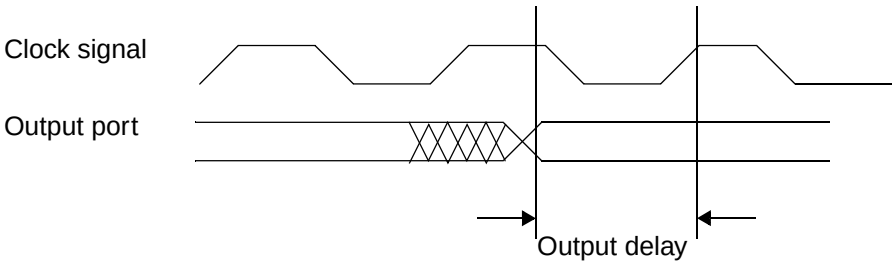
2. Next, the following commands remove the in_delay_1 and redefine the external_delay without port a:

```
rc:/> rm in_delay_1
rc:/> external_delay -input 500 -clock default {b c}
```

Setting Output Delays

External output delay is the delay between an output becoming stable and a capturing edge of a clock, as shown in Figure 1-12.

Figure 1-12 External Output Delay



➤ Constrain ports and pins within your design using the external_delay command. For example, the following commands create a clk2 clock, and set the output delay named out_con to 400 picoseconds, as shown in Figure 1-12 on all output ports:

```
rc:/> define_clock –name clock2 –period 2500
rc:/> external_delay -clock [find / -clock clock2] –output 400 -name out_con \
    [find /des* -port ports_out/*]
```

The SDC equivalent is:

```
rc:/> dc::create_clock -period 2.5 -name clock2
rc:/> dc::set_output_delay .4 -clock clock2 [dc::all_outputs]
```

Setting External Driver and Load Constraints

RTL Compiler constrains boundaries through the `external_driver`, `external_pin_cap`, and `max_fanout` attributes.

Use the `external_driver` attribute to:

- Find the name of the library cell driving the port
- Find the output pin of the library cell
- Assign the driven cell information to the input ports

➤ Control the input drive strength by specifying a particular pin from your technology library with the `external_driver` attribute. The following example forces the `Z` pin from the `AN2` libcell to drive all input ports in the `sheridan` design:

```
rc:/> set_attribute external_driver [find [find / -libcell AN2] -libpin Z] \
        /designs/sheridan/ports_in/*
```

The SDC equivalent command is:

```
rc:/> dc::set_driving_cell
```

See Modifying the External Driver on page 60 to control the modeling of the external driver even further.

➤ Remove driving cell from clock pins using the following command:

```
rc:/> set_attribute external_driver " " [find / -port CK_TD]
```

Modifying the External Driver

- To specify the exact input pin to use when calculating the slew and delay from the external driver, set the external_driver_from_pin attribute to the input libcell pin. For example:

  ```
  set_attribute external_driver_from_pin [find [find / -libcell AN2] -libpin A]
  ```

  RTL Compiler uses the worst case input pin if none is specified.

  This is equivalent to using the SDC `set_driving_cell` command with the `-from_pin` option.

- Specify an input slew other than 0 into the external driver input pin using the external_driver_input_slew attribute.

  This is equivalent to using the SDC `set_driving_cell` command with the `-input_transition_rise` and the `-input_transition_fall` options.

- Specify RTL Compiler to ignore the design rule violations as seen by the external driver pin using the ignore_external_driver_drc attribute. This is equivalent to the SDC `set_driving_cell` with the `-no_design_rule` option.

- Specify the number of parallel driving pins with a value greater than or equal to 0 using the external_non_tristate_drivers attribute. The default value is 0.

- Specify the external capacitive load on a port using the external_pin_cap attribute. The specified load will be calculated in femtofarads. The following example specifies an external capacitive load of `2` femtofarads on all output ports in the `gia` design:

  ```
  rc:/> set_attribute external_pin_cap 2 /designs/gia/ports_out/*
  ```

  The SDC equivalent command is:

  ```
  rc:/> dc::set_load
  ```

Specifying the Maximum Fanout

- Specify the maximum fanout for the net connected to a port using the max_fanout attribute. The resolution is 1/1000. For example, the following command sets the maximum fanout on all input and output ports to `10`:

  ```
  rc:/> set_attribute max_fanout 10 /designs/*
  ```

  The SDC equivalent command is:

  ```
  rc:/> dc::set_max_fanout
  ```

Specifying Operating Conditions

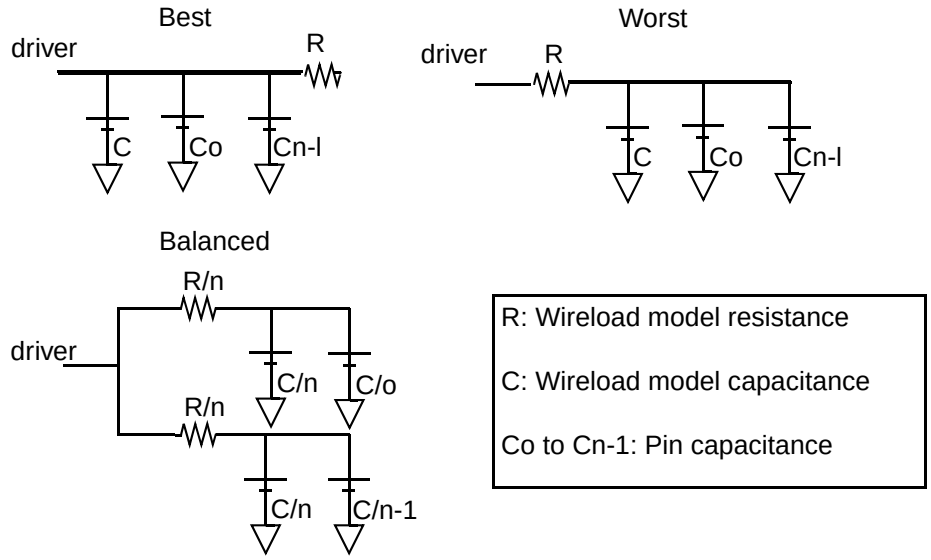
Describing the Operating Condition Attribute Values

Operating conditions describe the process, voltage, temperature (PVT), and tree type values under which the design must operate and for which the library cells are characterized.

- Process-derating value is related to the scaling of devices from fabrication variations. Using a process number less than the nominal value usually results in smaller delays.
- Temperature is the ambient-temperature value. Using a higher chip temperature usually results in larger delays. The temperature is influenced by the chip's power dissipation, cooling, package type, and air temperature.
- Voltage is the supply-voltage value and using a higher supply voltage usually results in smaller delays.
- Tree type is the interconnect-model (see Figure 1-13 on page 62). This value defines an RC tree topology for use in estimating net capacitance and resistance.

  The tree types are used to help calculate the RTL Compiler wire delay in the design. Think of this as the time of flight for the signal to go from the driving cell's output pin to the receiving cell's input pin. Figure 1-13 shows how RC is modeled in the three tree types and what this means for the delay through the wire.

Figure 1-13 Tree Types



- `best_case_tree` estimates only the wire capacitance because the load pin is modeled adjacent to the driver. The capacitance is modeled before any resistance of the net at the output pin of the driving cell. In this case, the RC delay is 0 since the C's do not need to be charged through a resistance (the driving cell being the *charger*).
- `worst_case_tree` estimates both the wire capacitance and wire resistance because the load pin is modeled at the end of the wire. This puts the capacitance for each load modeled at the end of the wire, and each *leg* of the wire is assumed to have the entire R value. Delay is calculated as R*C, so the delay to any load is:

  (Total R for the net) * (C of that load)
- `balanced_tree` estimates both the wire capacitance and wire resistance equally because all the load pins are modeled on equal branches of the interconnect wire. A balanced tree puts 1/nth of the R on each *leg* of the tree. This assumes that all the wires will be the same length in the layout. In this case, the delay is:

  (R/n) * (C of the load)

Overriding the Technology Library Default Operating Conditions

While the technology library has default operating conditions, you can override them. The new values are used to scale the timing calculations.

- Override default values by using the operating_conditions attribute to specify one of several conditions that might be available in the libraries.

  Make sure that you specify the attribute on the root level ("/").

  For example, suppose you want to perform synthesis using worst-case conditions. To find the available operating conditions, type the following command:

  ```
  rc:/> find /lib* -operating_condition *
  ```

- If your technology library offers a worst-case condition that meets your needs, then type the following command:

  ```
  rc:/> set_attribute operating_conditions worst_case /
  ```

  The following is the SDC equivalent command:

  ```
  rc:/> dc::set_operating_conditions worst_case
  ```

  For information about the available operating conditions in your technology library, consult your library vendor or foundry.

Getting a List of Available Operation Conditions from the Technology Library

- Get a list of the available operating conditions in a technology library before setting operating condition values using the get_attribute or `ls -attribute` commands.

  The `get_attribute` command is used to obtain information about a specific attribute while the `ls -attribute` command is used to list all the available attribute values.

  For example, the following command returns the operating condition temperature for the operating condition named `slow` in the technology library:

  ```
  rc:/> get_attribute temperature /libraries/techlib/operating_conditions/slow

  125
  ```

  The following example lists all the attribute values for the `slow` operating condition:

  ```
  rc:/> ls -attribute /libraries/techlib/operating_conditions/slow

  /libraries/slow/operating_conditions/slow (operating_condition)
  Attributes:
  ```

```
liberty_attributes = process 1 temperature 125 tree_type balanced_tree voltage
1.62
process = 1
temperature = 125 degrees
tree_type = balanced_tree
voltage = 1.62 volts
```

- Use the `-help` option with the `get_attribute` command to list all the available objects on which you can return attribute values:

```
rc:/> get_attribute -help
```

Selecting Default Operating Conditions from Different Libraries

If you have two operating conditions from two separate libraries, as shown in Example 1-2, and no operating condition is specified, then RTL Compiler picks the `default_operating_condition` from the first library specified in the `library` attribute list.

Example 1-2 Two Operating Conditions From Two Libraries

```
opcond1:
  operating_conditions("WC>MILV"){
    process : 1.0 ;
    temperature : 125 ;
    voltage : 2.7 ;
    tree_type : "worst_case_tree" ;
  }
  default_operating_conditions : LVWC3V ;

opcond2:
  nom_process     : 3;
  nom_temperature : 125;
  nom_voltage     : 2.7;
  operating_conditions(MACRO_LIB_worst) {
      process     : 3;
      temperature : 125;
      voltage     : 2.7;
      tree_type   : balanced_tree;
  }
```

If there are no `default_operating_conditions` in the libraries, then RTL Compiler creates an internally `_nominal_` operating condition with the following characteristics in the first library:

```
/libraries/lsi_10k/operating_conditions/_nominal_
(operating_condition)
    All attributes:
      liberty_attributes = process 1.0 temperature 25.0 voltage 5.0
      process = 1
      temperature = 25 degrees
      tree_type = balanced_tree
      voltage = 5 volts
```

Setting Separate Operating Condition Attribute Values

You can set the process, temperature, tree_type, and voltage attribute values separately. When you change the attributes of an operating condition, RTL Compiler calculates the new slew and delay based on the k derating factors. Set the values within a *reasonable* range to get a useful timing report because RTL Compiler may need to extrapolate beyond the values for which the library was characterized.

Set the attributes on a specified operating condition that will be used for timing analysis.

- Set the process value using the `process` attribute.

  For example, the following command sets the process attribute value to `1.4` for the `worst_case` operating condition:

  ```
  rc:/> set_attr process 1.4 [find /libraries -operating_condition worst_case]
  ```

- Set the temperature value using the `temperature` attribute.

  For example, the following command sets the `temperature` attribute value to `115` for the `worst_case` operating condition:

  ```
  rc:/> set_attr temperature 115 [find /libraries -operating_condition
  worst_case]
  ```

- Set the `tree_type` value using the `tree_type` attribute.

  For example, the following command sets the `tree_type` attribute value to `balanced_tree` for the `worst_case` operating condition:

  ```
  rc:/> set_attr tree_type balanced_tree [find /libraries -operating_condition
  worst_case]
  ```

- Set the voltage value using the `voltage` attribute.

  For example, the following command sets the voltage attribute value to `1.2` for the `worst_case` operating condition:

  ```
  rc:/> set_attr voltage 1.2 [find /libraries -operating_condition worst_case]
  ```

Specifying Wire-load Models

A *wire-load model* is the net resistance and capacitance (RC) model used for timing analysis, and it provides an estimate of the RC load of nets calculated from fanouts.

Wire-load models are used to calculate the loading effect of interconnect delays in the design.

By default, if an area-based wire-load model selection is supported and enabled or a default wireload model is specified, then RTL Compiler extracts this information from the technology library. You can specify a different wire-load model if needed.

Setting the Wire-load Mode

The wire-load mode can be one of the following:

- `top`

  Uses the wire-load model of the top-level design for all the nets in all the subdesigns. Hierarchical boundary pins are not counted as fanouts.

- `enclosed`

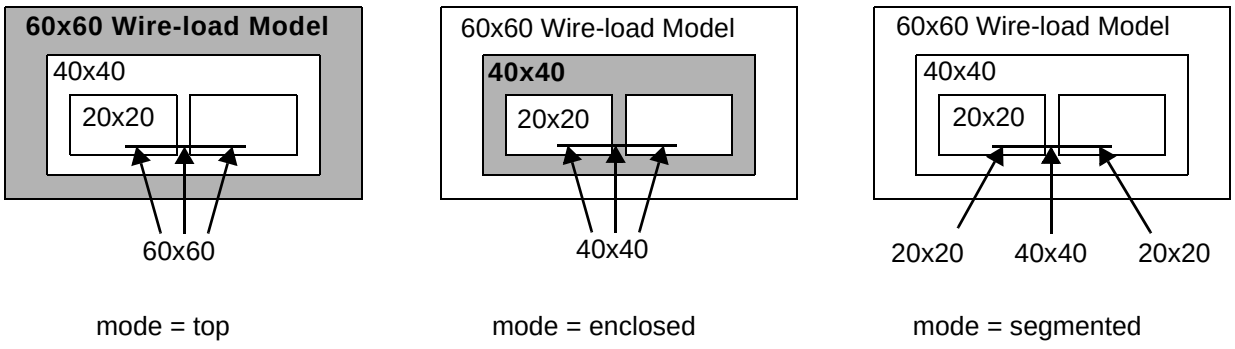
  Uses the wire-load model of the smallest block that fully encloses the net to compute the load of the net. Hierarchical boundary pins are not counted as fanouts.

- `segmented`

  Divides nets that cross hierarchical boundaries into segments, with one segment for each level of hierarchy. Separate load values are computed for each segment (counting the hierarchical boundary pins as individual fanouts) and the load values are added together.

Figure 1-14 shows the wire-load modes for the three wire-load model names.

Figure 1-14 Wire-load Modes



➤ Set the wire-load mode using the following command:

```
rc:/> set_attribute wireload_mode mode /
```

The SDC equivalent command is:

```
rc:/> dc::set_wire_load_mode mode
```

Finding Available Wire-load Models

➤ Report all available wire-load models using the following command:

```
rc:/> find /lib* -wireload *
```

You must load the technology library using the library attribute before running this command.

➤ Use the report area command to report the wire-load model used for each hierarchy.

➤ Get the wireload model specifically for each design and subdesign using the following commands:

```
rc:/> get_attribute force_wireload design_name
rc:/> get_attribute force_wireload subdesign_name
```

If you know the instance name, then use the following command:

```
rc:/> get_attribute force_wireload [get_attribute subdesign instance_name]
```

Specifying a Wire-load Model

➤ Specify the wire-load model to be used during synthesis by setting the value of the force_wireload attribute to a wire-load model in the technology library on a particular design.

The following example specifies the 1x1 wire-load model on the chekov design:

```
rc:/> set_attribute force_wireload 1x1 chekov
```

The SDC equivalent command is:

```
rc:/> dc::set_wire_load_model 1x1 chekov
```

You must elaborate the design using the elaborate command before you can specify wire-load models. The wire-load model is defined in the technology library.

➤ Specify a custom wire-load model from a library that contains multiple libraries using the following command:

```
rc:/> set_attribute force_wireload [find [find / -library lib_name] \
    -wireload wireload_model] [find / -design design_name]
```

The SDC equivalent command is:

```
rc:/> dc::set_wire_load_model [find [find / -library lib_name] \
    -wireload wireload_model] [find / -design design_name]
```

For example, the following command specifies that the CU_LARGE wire-load model from the library nlc18_cwlm be used with the design alu.

```
rc:/> set_attribute force_wireload /lib*/nlc18_cwlm/wireload_models/CU_LARGE \
    /des*/alu
```

The SDC equivalent command is:

```
rc:/> dc::set_wire_load_model -name CU_LARGE -library nlc18_cwlm
```

Use a custom wire-load model by overriding the default wire-load model from the technology library using the following command:

```
rc:/> set_attribute force_wireload wireload_model design_name
```

The SDC equivalent command is:

```
rc:/> dc::set_wire_load_model wireload_model design_name
```

- Retrieve the wire-load model for a particular design using the `wireload` read-only attribute:

```
rc:/> get_attribute wireload top
/libraries/techlib/wireload_models/1x1
```

- List the contents of a particular wire-load model using the `-attribute` and `-long` options of the `ls` command:

```
rc:/> ls -a -l /libraries/CDE/wireload_models/CDE13_Aggressive
```

In the above example, `CDE` is the library name and `CDE13_Aggressive` is the wire-load model.

Using Unnamed Wire-load Models

If RTL Compiler encounters an unnamed wire-load model (a wire-load model without a header) in the technology library, then it assigns the wire-load model a name using the following convention:

```
unnamed_wlm_<lib_file_name>_<row>_<col>_<id>
```

- *lib_file_name* is the name of the library file
- *row* and *col* point to the location within the library file by row and column
- *id* is a numerical identifier (usually “1”)

The named wire-load model is then processed as if you had assigned it a name.

Specifying a Zero Wire-load Model

- Use the `none` argument with the `force_wireload` attribute to prevent the use of wire-load models. You can specify a zero wire-load model on a particular subdesign or on the entire design.

The following example prevents wire-load models from being used on the entire design:

```
rc:/> set_attribute wireload_mode top /
rc:/> set_attribute force_wireload none /designs/*
```

The following are the SDC equivalent commands:

```
rc:/> dc::set_wire_load_mode top
rc:/> dc::set_wire_load_model -name none
```

Note: Use these two commands if you want to perform zero wire-load timing analysis on the design.

Changing Wire-load Selection Groups

By default, RTL Compiler uses the wire-load selection group as defined in the technology library.

➤ Choose a different selection group using the following attribute:

```
rc:/> set_attribute wireload_selection wc_group /
```

The following is the SDC equivalent command:

```
rc:/> dc::set_wire_load_selection_group group_name
```

Make sure that you specify the attribute on the root-level ("/"). The `wc_group` value must be defined in the technology library.

➤ List the available wire-loads and wire-load selection groups using the `ls` command, as shown in the following example:

```
rc:/> ls /libraries/tutorial/wireload_models
/libraries/tutorial/wireload_models:
./      AL_LARGE    AL_MEDIUM   AL_SMALL    CU_LARGE    CU_MEDIUM   CU_SMALL
rc:/> ls /libraries/tutorial/wireload_selections
/libraries/tutorial/wireload_selections:
./                   ALUMINUM            COPPER
```

Assigning a Different Wireload to the Input and Output Ports of a Design

When the wires to and from ports within a module are longer than a typical wire between logic elements, you can assign a different wireload to the input and output ports of the design from the rest of the design.

➤ Specify the wire-load model to use for this port using the `external_wireload_model` attribute.

For example, the following command applies the wireload model named *example* to all the outputs of a design:

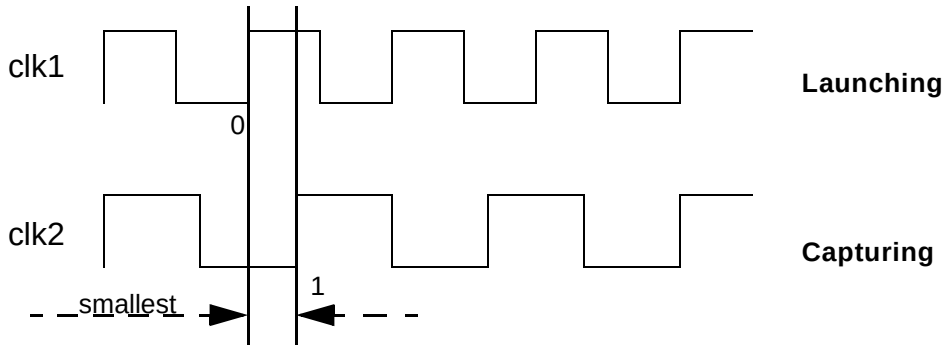
```
rc:/> set_attribute external_wireload_model [find / -wireload example] \
/designs/*/ports_out/*
```

Setting Timing Exceptions

A *timing exception* is an exception to RTL Compiler's default (single-cycle) timing behavior. For each place in the design that does not conform to the default behavior, you must specify a timing exception. Timing exceptions are timing paths in a design that you either do not want RTL Compiler to optimize or you want to optimize with more relaxed constraints.

By default, for designs that have only one clock, RTL Compiler uses one clock cycle as the interval between the launching clock edge and the receiving clock edge for timing analysis. For designs with paths that are between two different clock domains, RTL Compiler uses the smallest non-negative interval between the launch edge of `clk 1` and the receiving edge of `clk 2` for timing analysis, as shown in Figure 1-15. These intervals are also used to determine whether optimization was successful.

Figure 1-15 Defining Timing Constraints



To override this default behavior, specify a timing exception.

Before performing timing analysis, you must specify a set of timing constraints that define the synchronous behavior of the design. Apply timing exceptions to describe exceptions to the synchronous design.

There are three kinds of timing exceptions:

- A false path indicates that the particular timing path does not propagate a signal. A false path timing exception specifies an *infinite* number of clock cycles as the interval time. Any path in a design that you do not want RTL Compiler to time and optimize must be specified as a false path. By default, timing paths that cross clock domains are automatically false pathed. See Specifying a False Path on page 73.
- Multicycle paths, which require more than one clock cycle to propagate data, specify a *finite* number of clock cycles for the interval. See Overriding the Default Timing Constraint for Multi-Cycle Paths on page 75.

- Paths delays are timing constraints for paths that meet the specified criteria. See Specifying Path Delays on page 77.

Timing Exception Priority

RTL Compiler sets the priority on timing exceptions in the following order:

- false paths
- path delays
- multicycle paths

You can check the priority attributes in the following directory:

```
"/des*/*/timing/exceptions
```

See Reporting Timing Exceptions on page 139 for information on how to use the report timing command to report timing exceptions.

See Handling Timing Exceptions on page 197 for troubleshooting tips on timing exception issues.

To define exceptions in a multi-mode environment, see Chapter 5, "Performing Multi-Mode Timing Analysis."

Specifying a False Path

A *false path* is a path that will never be used, thus, it does not need to meet timing requirements. Signals that activate test modes are examples of false paths. Avoid timing violations by setting false path exceptions.

If a timing path does not propagate a signal, you can declare it as a functional false path. RTL Compiler will then not work on this path for timing.

➤ Use the `path_disable` command to disable timing analysis for specified paths. A path that satisfies this exception does not have a timing constraint set on it and hence, is not considered to be timing critical.

- Use the `-from` option to specify a clock object, input port of a design, and clock pins of flip-flops.

  For example, the following command performs a path disable from all the start points that fan out to `reg1/D`:

  ```
  rc:/> path_disable -from [fanin -startpoint reg1/D] -to reg1/D
  ```

- Use the `-to` option to specify a clock object, output ports of a design, and input D pins of flip-flops.

  The following command performs a path disable on all the endpoints to which `reg1/CK` fans out:

  ```
  rc:/> path_disable -to [fanout -endpoint reg1/CK] -from reg1/CK
  ```

  The following is the SDC equivalent command:

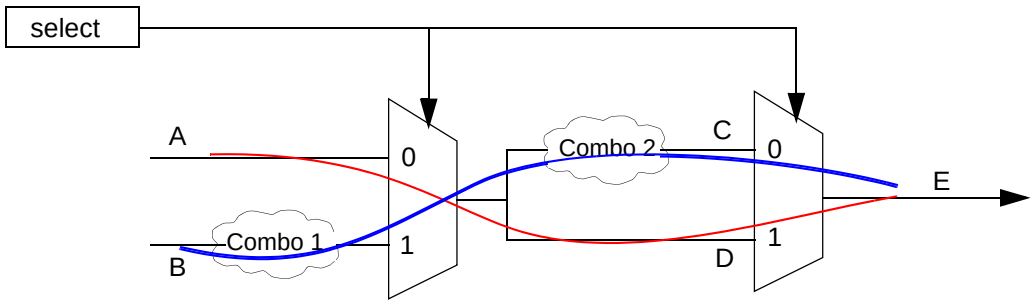
  ```
  rc:/> dc::set_false_path
  ```

  Note: Using the `path_disable` command for flip-flops as the start and endpoints do not need to reference the pins of the flip-flops.

- Use the `-through` option to specify a sequence of points, ports, pins of hierarchical blocks, or pins of sequential cells.

False paths are sometimes associated with multiplexers that are driven by the same select lines ensuring that certain paths through the muxes are not possible. Consider the example shown in Figure 1-16. Since the select line is common, paths `A - D - E` and `B - C - E` are false paths and have to be disabled for timing analysis.

Figure 1-16 False Paths



All `path_disable` exceptions are stored in the following location:

`/designs/`*`design_name`*`/timing/exceptions/path_disables/`*`exception_name`*

- Find stored path-disable exceptions using the `find` command with the `-exception` option.

Note: Setting false paths on hierarchical pins preserves RTL Compiler generated multiplexer/datapath hierarchies. For example, if you have the following SDC constraint:
`set_false_path -from xyz -through [get_pins -hierarchical inst_xyz/*]`
then this causes RTL Compiler to preserve all the artificial multiplexer hierarchies since they

have a constraint sitting on their pins. Try to workaround this issue, by writing the SDC constraint in a different way.

To define false paths in a multi-mode environment, see Chapter 5, “Performing Multi-Mode Timing Analysis.”

Overriding the Default Timing Constraint for Multi-Cycle Paths

A *multi-cycle path* is a timing path that does not propagate a signal in one cycle. You can change the default behavior for multi-cycle paths.

By default, RTL Compiler uses single clock cycle timing for all paths, which means that data should reach from the startpoint to the endpoint in one clock cycle.

The default timing constraint corresponds to a launch shift of `0` and a capture shift of 1. Incrementing the launch shift value moves the launch one cycle earlier. Incrementing the capture shift value moves the capture one cycle later. The shift values are not limited to the values shown in Figure 1-15; any integer values may be used.

The `multi_cycle` command creates a timing exception object that overrides the default clock edge relationship for certain paths in the design. The numbers under each rising edge shown in Figure 1-15 indicate the shift values that correspond to that edge.

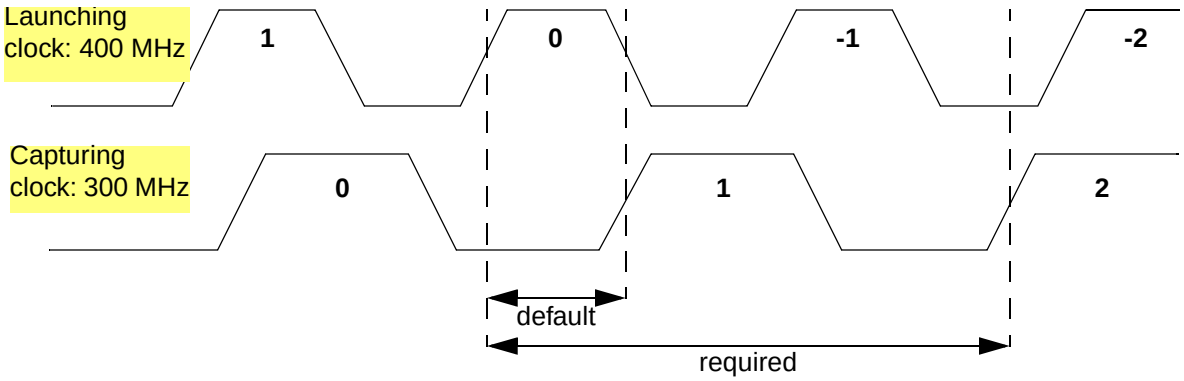
- Specify any pair of edges to use for the timing constraint using the `launch_shift` and `capture_shift` options with the `multi_cycle` command.

  The SDC equivalent command is:

  ```
  rc:/> dc::set_multicycle_path
  ```

For example, consider the two clock waveforms shown in Figure 1-17. The upper clock launches a path at a rising-edge triggered flip-flop and the lower clock captures that path at the D pin of another rising-edge triggered flip-flop. There are four launch clock periods for every three capture clock periods. By default, RTL Compiler computes timing relationships based on the shortest positive difference between rising edges on the two waveforms (shown by the arrow).

Figure 1-17 Launching and Capturing Clocks



The numbers under each rising edge indicate the shift values that correspond to that edge. The default timing constraint corresponds to a launch shift of 0 and a capture shift of 1. Incrementing the launch shift value moves the launch one cycle earlier. Incrementing the capture shift value moves the capture one cycle later. The shift values are not limited to the values shown in the example; any integral value may be used.

➤ Change the timing relationship to use the third rising edge of the launch clock and the third rising edge of the capture clock using the following command:

```
rc:/> multi_cycle -launch_shift -1 -capture_shift 2 \
      -from [find / -clock "400MHz"] -to [find / -clock "300MHz"]
```

The command applies to all paths being launched at the 400 MHz clock and captured at the 300 MHz clock.

To define multicycle paths in a multi-mode environment, see Chapter 5, "Performing Multi-Mode Timing Analysis."

Specifying Path Delays

For information on how to direct timing optimizations for specifying paths, see Modifying Path Constraints on page 103.

To define path delays in a multi-mode environment, see Chapter 5, “Performing Multi-Mode Timing Analysis.”

Creating a Timing Exception Object

In RTL Compiler, objects are general terms for items within the design information hierarchy. A *timing exception object* is a timing exception located in the following RTL Compiler design directory:

`/designs/`*`design_name`*`/timing/exceptions/`

- Use the path_delay command to create a timing exception object that lets you specify the timing constraint for paths that meet the criteria. The command returns the directory path to the object that it creates.
- Create a meaningful handle in the: `/designs/*/timing/exceptions/path_delays` directory by specifying the `-name` option as follows:

  ```
  rc:/> path_delay -delay 5000 -from [find / -port a] -name override
  /designs/alu/timing/exceptions/path_delays/override
  ```

  The SDC equivalent command is:

  ```
  rc:/> dc::set_max_delay
  ```

- Use the `find` command to search for a named timing exception:

  ```
  rc:/> find / -exception override \
       /designs/alu/timing/exceptions/path_delays/override
  ```

- Use the `rm` command to remove a named timing exception:

  ```
  rc:/> rm [find / -exception override]
  Info    : Removing a timing exception. [TIM-307]
          : The exception is 'path_delay/override'
  ```

➤ Save the timing exception for future reference using a Tcl variable. For example:

```
rc:/> set override [path_delay -delay 5000 -from [find / -port a]]
```

RTL Compiler provides an exception name (or handle) if the `-name` option is not specified.

The path delays are located in the following RTL Compiler design directory:

`/designs/`*`design_name`*`/timing/exceptions/path_delays`

Assigning Paths to a Cost Group

A *cost group* is a set of critical paths to which you can apply weights or priorities that the optimizer will recognize. Paths assigned to a cost group are called path groups. See Creating Path Groups and Cost Groups on page 98 for detailed information.

To define path groups in a multi-mode environment, see Chapter 5, “Performing Multi-Mode Timing Analysis.”

Specifying Path Exceptions

➤ Use the `specify_paths` command to create a string that indicates the path of a particular timing exception and as an argument to the `-paths` option to the timing exception commands.

The `specify_paths` command gives you more detailed control when specifying timing exceptions than the `-from`, `-through`, `-to` options alone could provide. For example, the following command specifies that the `path_disable` command should be applied to paths that are launched by the `clk1` clock and start with a rise pin transition as their startpoint:

```
rc:/> path_disable -paths [specify_paths -from_rise_pin clk1]
```

Note: Although the above example specified the `-from_rise_pin` option, it was possible to use a clock object. See the option arguments for a complete list of permissible objects. The following example specifies the `-to_fall_clock` option but uses a pin object:

```
rc:/> path_disable -paths [specify_paths -to_fall_clock ff1/D]
```

The above example specifies that the `path_disable` command should be applied to paths that end at the `ff1/D` pin and are captured by a falling edge of an ideal clock waveform.

Setting Path Exception Priorities

If a timing path satisfies two conflicting timing exceptions, then you can tell RTL Compiler to use one and not the other.

➤ Set a higher priority on the desired exception using the `user_priority` attribute. For example:

```
rc:/> set_attribute user_priority 5 $my_multi_cycle
rc:/> set_attribute user_priority 4 $other_exception
```

Default: `0`

To learn how to make all paths through a certain pin use default timing constraints, see Reporting and Adjusting Path Exceptions on page 198 and Handling Timing Exceptions on page 197.

Breaking the Path of a Specified Pin

➤ Break the path of a specified pin using the following attribute:

```
rc:/> break_timing_paths {true | false | propagate_slews}
```

- Set this attribute to `true` to set external delays on specified pins. They can then function as `-to`, `-through`, or `-from` points in path exception commands. Timing will consider the breakpoint as an endpoint.
- Set the attribute to `false` to prevent the slew values from propagating through the specified pin. The input slew at the driven pins will be `0`.
- If the attribute is set to `propagate_slews`, then the slew will propagate from the driver. The load on the net is unaffected by the `break_timing_paths` attribute.

Setting Data-to-Data Checks

You can also apply setup checks between two data pins. It is similar to clock-to-data setup checks, except that the from-pin (also called the related pin) is not a clock pin, but a data pin.The data-to-data timing check is a zero-cycle relationship between the two data signals. This differs from the usual clock-to-data setup checks where the data edge is usually one cycle away from the clock edge.

Note: Setting data-to-data checks breaks the timing paths at the to-pin.

The data-to-data check can be specified in one of the following ways:

- Using the SDC `set_data_check` command.
- In the library with timing arcs of type `non_seq_setup_rising` and `non_seq_setup` falling

To compute the data-to-data checks, the tool must calculate the minimum delay paths to the from_pin. This is similar to considering the minimum clock latency of the clock for a setup check. The minimum delays are only computed on the paths that reach the from_pin of the data-to-data setup check. On those minimum delay paths, the `has_min_delay` pin and port attributes are set to `true`. You can check the minimum delays with the `min_timing_arcs` and `min_slew` pin attributes.

Note: Only setup data-to-data checks are supported. Hold data-to-data checks are ignored.

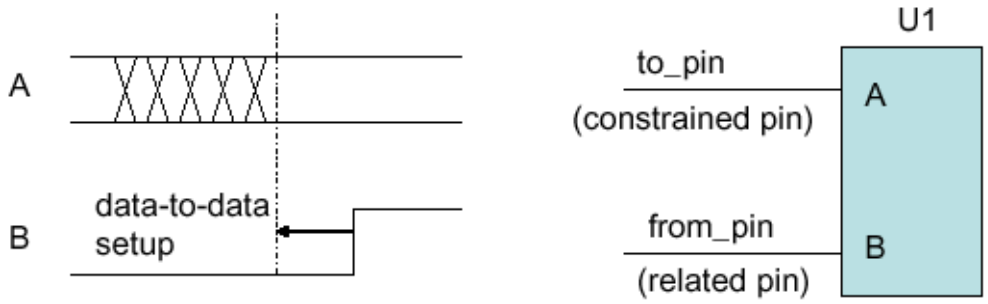
Specifying Data-to-Data Checks with SDC

The following command specifies a data-to-data check of 2.0 from pin `U1/B` to pin `U1/A`

```
rc:/ > dc::set_data_check -from U1/B -to U1/A -setup 2.0
```

The setup between the two data pins `U1/B` and `U1/A` is illustrated in Figure 1-18.

Figure 1-18 Data-to-Data Check



Data-to-data check timing arcs from the library

Data-to-data checks can be specified in your library with timing arcs of type `non_seq_setup_rising` and `non_seq_setup_falling`. In this case, the setup value is computed from the library delay table based on the transition times of both data pins.

To infer these timing arcs from the library, you must set the following root attribute:

```
set_attribute timing_disable_non_sequential_checks true /
```

Note: By default, these timing arcs are not taken into account.

How to enable data-to-data checks on non endpoint pins

By default, data-to-data checks are disabled on from_pins (constrained pins) that are not endpoints of the design. This prevents that timing paths are broken at the to_pin when inserting the data-to-data check.

➤ To change the default behavior use the `tim_ignore_data_check_for_non_endpoint_pins` root attribute.

The default value is `true`.

Set this attribute to `false` to enable the data-to-data checks on all pins, including pins that are not endpoints.

Set this attribute to `sdc_set_data_check_only` to *only* ignore data-to-data checks that are created using the SDC `set_data_check` command and that are set on a constrained pin that is not an endpoint.

Set this attribute to `lib_non_seq_setup_only` to *only* ignore data-to-data checks specified on instances whose library cells have timing arcs of type `non_seq_setup_rising` or `non_seq_setup_falling` and for which the related pin is not an endpoint.

Disabling Data-to-Data Checks

By default, the data-to-data checks specified with the SDC commands are enabled, while the data-to-data checks specified in the library are by default disabled.

➤ To disable all data-to-data checks, set the following attribute **before** loading the design:

```
set_attribute enable_data_check false /
```

Setting Design Rule Constraints

Design constraints consist of design rule constraints (DRCs), design environment constraints, and top-level boundary constraints. RTL Compiler tries to satisfy the design rule constraints, optimize for speed, area, and power while using the environmental constraints and the top-level boundary constraints to describe the *context* of the design.

This section includes the following information:

Note: When setting constraints, constraints set on a design override the library constraints.

Controlling the Use of Design Rule Constraints From the Technology Library

Design rule constraints are imposed on synthesis by the physical limitations of the technology library chosen to implement a design. Design rules include the following three elements:

- Maximum capacitance per net
- Maximum fanout per gate
- Maximum transition time of a signal

These three constraints are used together to ensure that the library limits are not exceeded. You can set looser design rule constraints than the ones listed in the technology library by using the `ignore_library_drc` attribute with the `max_capacitance`, `max_fanout`, and `max_transition` attributes to control the use of design rule constraints from the technology library.

➤ To force RTL Compiler to ignore design rule constraints specified in the technology library, set the `ignore_library_drc` attribute to `true`. For example:

```
rc:/> set_attribute ignore_library_drc true /designs/designname
```

Important

Use the `ignore_library_drc` attribute with care. The `ignore_library_drc` attribute only applies to the top-level module and all subdesigns.

Specifying the Maximum Fanout Limit

Each input pin of every gate in the technology library has a fanout-load attribute. The fanout load increases the capacitance the driver must charge and discharge. The sum of all the fanout loads connected to an output cannot exceed the maximum fanout limit.

Using the `max_fanout` command limits the number of gates driven by an output. Setting a maximum fanout limit on a port specifies that the net connected to that port to have a total fanout load that is less than the value you specify. Setting a maximum fanout limit on a design sets the default maximum fanout for that design. See Generating a Net Report on page 157 for an example net report.

- Specify a maximum fanout design rule limit for all nets in a design or on a port using the `max_fanout` attribute on a top-level block or port:

  ```
  rc:/> set_attribute max_fanout value [design|port]
  ```

  where *`value`* is an integer or *`no_value`*.

  The following is the SDC equivalent command:

  ```
  rc:/> dc::set_max_fanout
  ```

  If this attribute is set, then use the `report design_rules` command to report design rule violations. Even if you have not specified any constraints, rules may still be inferred from the technology library. See DRV Report for max_fanout on page 155 for an example report.

- Return the total fanout cost of a design using the `max_fanout_cost` attribute.

Specifying the Maximum Capacitance Limit

The load on a net is comprised of the fanout and interconnect capacitance. The capacitance of a wire is detailed in the wire-load model.

Setting a maximum capacitance limit on a port specifies that the net connected to that port to have a total capacitance that is less than the value you specify. Setting a maximum capacitance limit on a design sets the default maximum capacitance for that design.

- Specify a maximum capacitance limit for all nets in a design or on a port using the `max_capacitance` attribute on a top-level block or port:

  ```
  rc:/> set_attribute max_capacitance value [design|port]
  ```

  where *`value`* is an integer for the capacitance in femtofarads or *`no_value`*.

  The following is the SDC equivalent command:

  ```
  rc:/> dc::set_max_capacitance
  ```

Note: If this attribute is set, then use the `report design_rules` command to report design rule violations. Even if you have not specified any constraints, rules may still be inferred from port attributes or from the technology library. See DRV Report for max_capacitance on page 154 for an example report.

➤ Return the total capacitance cost of a design using the `max_cap_cost` attribute.

Specifying the Maximum Transition Limit

The transition time is the amount of time it takes for a signal to change from one logic state to another and is a product of the signal-line capacitance and resistance on a wire. The wire-load model details the capacitance and resistance characteristics.

The `max_transisiton` attribute sets a maximum transition limit for the nets attached to specified ports or designs. Setting a maximum transition limit on a port specifies that the port's transition time is less than the value you specify. Setting a maximum transition limit on a design sets the default maximum transition for that design.

➤ Specify a maximum transition design rule limit for all nets in a design or on a port using the `max_transition` attribute on a top-level block or port:

```
rc:/> set_attribute max_transition value [design|port]
```

where `value` is an integer or `no_value`.

The following is the SDC equivalent command:

```
rc:/> dc::set_max_transition 0.5 [get_designs designname]
```

➤ If this attribute is set, then use the `report design_rules` command to report design rule violations. Even if you have not specified any constraints, rules may still be inferred from the technology library.

```
rc:/> report design_rules [design]... [> file]
```

See DRV Report for max_transition on page 153 for an example report.

➤ Return the total transition time of a design using the `max_trans_cost` attribute.

Disabling Timing Arcs

Timing arcs, also called libarcs, define the timing relationships between pins on the same cell or net. Cell arcs describe the cell's internal pin-to-pin timing and are defined in the technology library cell description. Net arcs are defined by the operating conditions, which specify the type of resistance-capacitance (RC) tree model to use.

Timing analysis uses timing arcs to describe the delay from each input to each output, and the constraints between input pins, such as setup and hold, or minimum pulse width.

You can

- Disable cell arcs in your design to break a combinational feedback loop or to remove an undesired path from timing analysis. Disabling a pin disables all timing arcs to that pin.
- Disable a timing arc or pin on a library cell, which causes the specified arcs to be disabled on all instances of that library cell.

➤ Disable or break timing arcs of library cells for synthesis and static timing analysis using one of the following methods:

  - Set the `enabled` attribute to `false` on the RTL Compiler design data structure object called `libarc`, which represents a timing arc within the technology library.
  - Add the libarcs you want to disable to the `disabled_arcs` attribute under the instance.

Note: When setting constraints, the constraints set on a design override the library constraints.

Disabling Timing Arcs for Library Cells

➤ Use the find command to list all the libarc objects on a specified pin of a library cell.

As shown in Example 1-3, the following command lists the libarc objects on the CO output pin of the ADDFXL library cell:

```
rc:/> find ADDFXL/CO -libarc *
```

Example 1-3 libarc Objects

```
/libraries/slow/libcells/ADDFXL/CO/inarcs/A_n92
/libraries/slow/libcells/ADDFXL/CO/inarcs/A_n91
/libraries/slow/libcells/ADDFXL/CO/inarcs/B_n92
/libraries/slow/libcells/ADDFXL/CO/inarcs/B_n91
/libraries/slow/libcells/ADDFXL/CO/inarcs/CI_n90
/libraries/slow/libcells/ADDFXL/CO/inarcs/B_n90
/libraries/slow/libcells/ADDFXL/CO/inarcs/A_n90
```

The libarc objects have the following attributes defined:

■ enabled

The enabled attribute is true by default unless RTL Compiler does not support those timing arcs (for example, borrow arcs in flops).

➤ Set the enabled attribute to false to break timing arcs.

■ from_pin

The from_pin attribute specifies the path to the start pin of the timing arc.

Because the libarc object is located under the to_pin object, there is no need for a separate attribute to specify the termination or destination pin.

■ type

Represents the timing arc type.

For example, the following attributes on the libarc object called A_n92 are listed in Example 1-3:

```
ls -a -l /libraries/slow/libcells/ADDFXL/CO/inarcs/A_n92
 /libraries/slow/libcells/ADDFXL/CO/inarcs/A_n92 (libarc)
    All attributes:
      enabled = true
      from_pin = /libraries/slow/libcells/ADDFXL/
      type = combo_arc
```

- To break the timing arc from the A input pin to the CO output pin for the ADDFXL cell, set the following attribute:

```
rc:/> set_attribute enabled 0 /libraries/slow/libcells/ADDFXL/CO/inarcs/A_n92
```

- To break all timing arcs leading to the CO output pin for the ADDFXL cell, set the following attribute:

```
rc:/> set_attribute enabled 0 /libraries/slow/libcells/ADDFXL/CO/inarcs/*
```

Disabling Timing Arcs for a Specified Instance

The other option for disabling timing arcs affects only the specified instance and not all the references of the library cell.

Note: You can only disable arcs of mapped instances.

Every instance has an attribute called disabled_arcs, which is a Tcl list of *libarc* objects. The list is empty by default, but you can add the *libarc* objects that you want to disable for a given instance.

- Disable or break timing arcs (path delay) of library cells for synthesis and static timing analysis using the disabled_arcs command.

  For example, you have a p6305A instance of the ADDFXL library cell. To disable all timing arcs leading to the CO pin for the p6305A instance, type the following:

```
rc:/> set_attribute disabled_arcs [find ADDFXL/CO -libarc *] \
   [find . -instance p6305A]
```

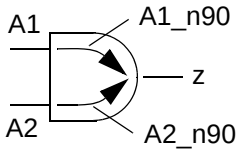
  The following is the SDC equivalent command:

```
rc:/> dc::set_disable_timing [find ADDFXL/CO -libarc *] \
    [find . -instance p6305A]
```

  The following command disables all the timing arcs from the A1 pin to Z in the p0553A instance, which is an AND2D1 cell in the design after mapping to the technology library, as shown in Figure 1-19:

```
rc:/>set_attribute disabled_arcs [find ./lib*/*/libcells/AND2D1 -libarc \
     A1_n90] [find ./ -instance p0553A]
```

Figure 1-19 Disabling an Instance Timing Arcs



Propagating Case Logic Downstream

You can propagate both clock signals and case logic downstream if loop breakers cause an interruption. See Handling Combinational Feedback Loops on page 201 for detailed information.

- Force a specified pin or port to assume the specified logic value for timing analysis using the following attribute:

  ```
  timing_case_logic_value {0 | 1 | no_value}
  ```

You can set this attribute on mapped leaf (combinational) instance pins, hierarchical boundary pins, outputs pins of sequential cells, and on input ports. The timing engine automatically sets the logic constants to their appropriate value.

Specifying Latch Time Borrow Values

Latch Time Borrowing or cycle stealing, is an analysis technique in which a portion of a path borrows timing margin from the next stage of the design, when the next stage is a level-sensitive latch operating in transparent mode.

Latch-based designs often use multi-phase, non-overlapping clocks to control successive registers in a data path. In these cases, you can use time borrowing to lessen the constraints on successive paths.

Designs containing latches are more complex than edge-triggered flip-flop designs because latches are transparent for a finite time during which the D input may stabilize and the logic following the Q output could begin evaluation. Time borrowing is based on this assumption.

In a physical circuit, optimal time borrowing automatically occurs. The logic following the Q output is always updated with the latest value passing from D through the latch, so that when D is stable, the computation for logic following the Q output begins immediately. Time borrowing is an attempt to model this behavior in timing analysis. However, it is always conservative in comparison to the real circuit.

Time borrowing does *not* do the following

- Modify the circuit
- Add a delay to the clock net
- Re-time (move logic from one side of the sequential element to the other)

This section shows how to set the latch analysis mode to `max_borrow` to put all of the slack on one side of the latch.

See Analyzing Latch-Based Designs on page 161 for information on analyzing latch-based designs using time borrowing (cycle stealing) to borrow time to meet timing constraints.

➤ Use the `latch_borrow` attribute to specify latch time borrow values if you do not want to use the dynamically computed ones. The slack on the `D` and Q sides of the latches will be computed assuming the specified borrow value.

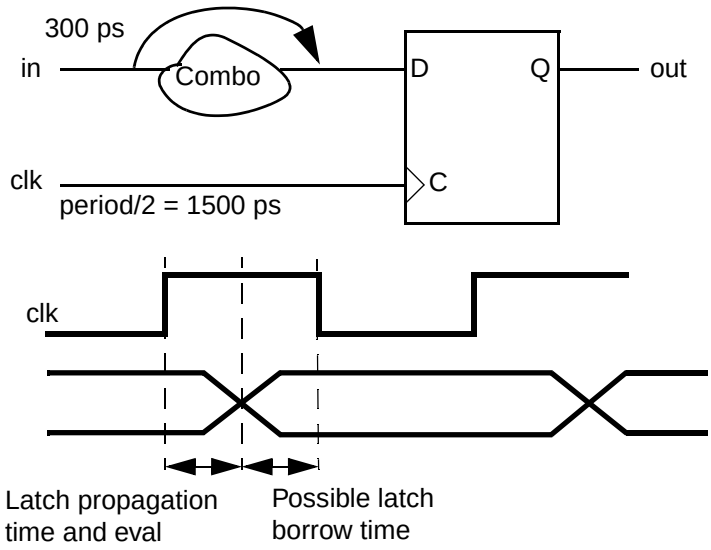
The example in Figure 1-20 assumes that the enable pin of the latch is driven by the defined clock signal and the designated 300 ps will enter the timing calculations. In effect, it means that the latched output is valid 300 ps after the rising edge of the clock.

Without setting the `latch_borrow` attribute, a latch circuit inferred will show a slack of about 1200 ps assuming there is no launch delay on the input port since the timing calculator will assume that the evaluation can arrive till the clock goes low disabling the gate of the latch. A `latch_borrow` of 0 ps setting will cause a timing violation of 300 ps

since the timing engine assumes that the edge launching the input also latches the evaluation. A `latch_borrow` setting of 300 ps clears the violation.

```
rc:/> set_attribute latch_borrow 300 [find ./ -design test_top]
```

Figure 1-20 Latch Borrow Example



- Specify the maximum time that can be borrowed by one cycle from the next cycle to meet timing constraints using the `latch_max_borrow` attribute. For example:

  ```
  rc:/> latch_max_borrow {float | no_value}
  ```

  The SDC equivalent command is:

  ```
  rc:/> dc::set_max_time_borrow
  ```

The specified value must be an integer greater or equal to 0 (decimal values are rounded to the nearest integer) or `no_value`. This attribute is available on latch cells, clocks, clock (enable) pins, data pins, or on a design. If the attribute is set on multiple objects that overlap each other, for example a latch instance and its data pin, then the minimum value will be taken. If the `latch_borrow` attribute has already been set, then the `latch_max_borrow` attribute is ignored.

Handling Ideal Nets

In RTL Compiler you can set ideal nets either explicitly or implicitly. RTL Compiler will never idealize a net based on the number of fanouts.

➤ Set the ideal_driver attribute to true or 1 to explicitly specify ideal nets. Depending on your design requirements, this forces RTL Compiler to ignore transition delays, connect delays, and design rule constraints of a pin or port.

For example, to set an ideal net, type the following command:

```
rc:/> set_attribute ideal_driver 1 pin_name/port_name
```

The following is the SDC equivalent command:

```
rc:/> dc::set_ideal_net
```

Set this attribute only on a primary input or output of an instance. When this attribute is set, RTL Compiler will not

- Insert buffers.
- Check design rule constraints (DRC).
- Perform fixes on the net.
- Upsize or split the driver on the specified pin or port.
- Consider the loading effect on the net.

RTL Compiler implicitly sets an ideal net if the driving pin or port is

- A pin of a sequential element that is an asynchronous set or reset. You can check for this attribute by using the ls -long -attribute command on the pin or libcell.
- A pin of a sequential element that is a clock pin. You can check for this attribute by using the ls -long -attribute command on the pin or libcell.
- Any input pin of a sequential element that has no setup arc. That is, any sequential element that may be asynchronous.

➤ Use the causes_ideal_net attribute to see whether a specified pin is causing its net to become an ideal net.

For example, the following command queries whether the in1[1] input pin is causing its net to become ideal:

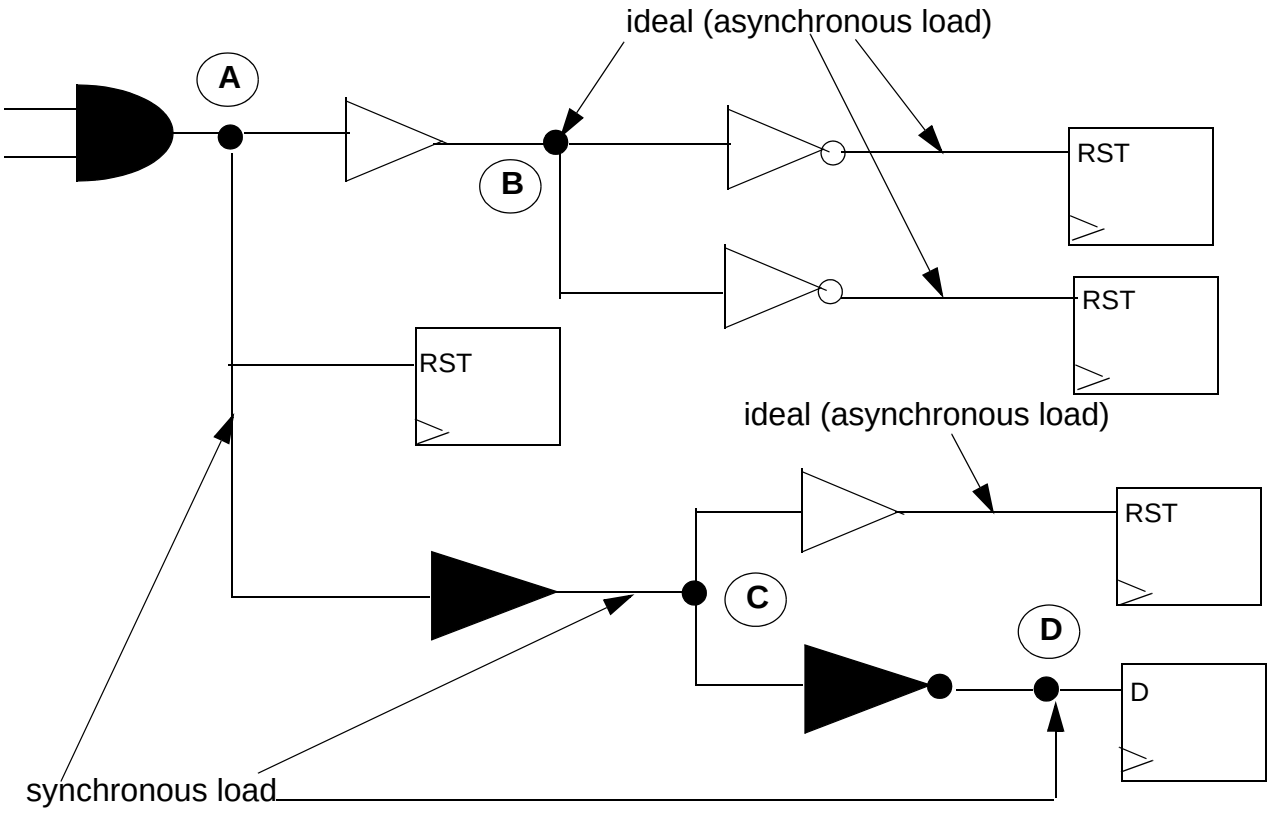
```
get_attribute causes_ideal_net \
    {/designs/trance/instances_hier/inst1/pins_in/in1[1]}
false
```

Nets with Asynchronous and Synchronous Loads

For those nets that have both asynchronous and synchronous loads, RTL Compiler by default will only treat the asynchronous loads as ideal and not the entire net. Only the synchronous loads will be used to compute the total load capacitance of these nets during timing and optimization. For example, if there are 20 asynchronous reset loads on a net and each has a load capacitance of 100 and there is also two D-pin loads and each has a load capacitance of 150, the total load capacitance of the net is:

```
2 * 150  = 300
```

When a net has an inverter or buffer tree that in turn drives the synchronous or asynchronous loads, then any asynchronous pin of a register is considered to be an asynchronous load. Any inverter or buffer that drives *only* other asynchronous loads is itself considered to be an asynchronous load for its driver net. Thus, if a buffer or inverter tree drives asynchronous or synchronous loads, the asynchronous selection process is propagated backwards through the chain of buffers or inverter.



In the diagram above, the drivers with synchronous loads are in black. Other nets have zero capacitance because they do not have any synchronous loads.

As an example, assume the following capacitance values for the diagram above:

- RST = 10
- D = 15
- Buffer = 5
- Inverter = 5

The load at point A would be:

```
0 + 0 + 5 = 5
```

The load at point B would be:

```
0 + 0 = 0
```

The load at point C would be:

```
0 + 5 = 5
```

The load at point D would be 15.

2

Setting Optimization Constraints

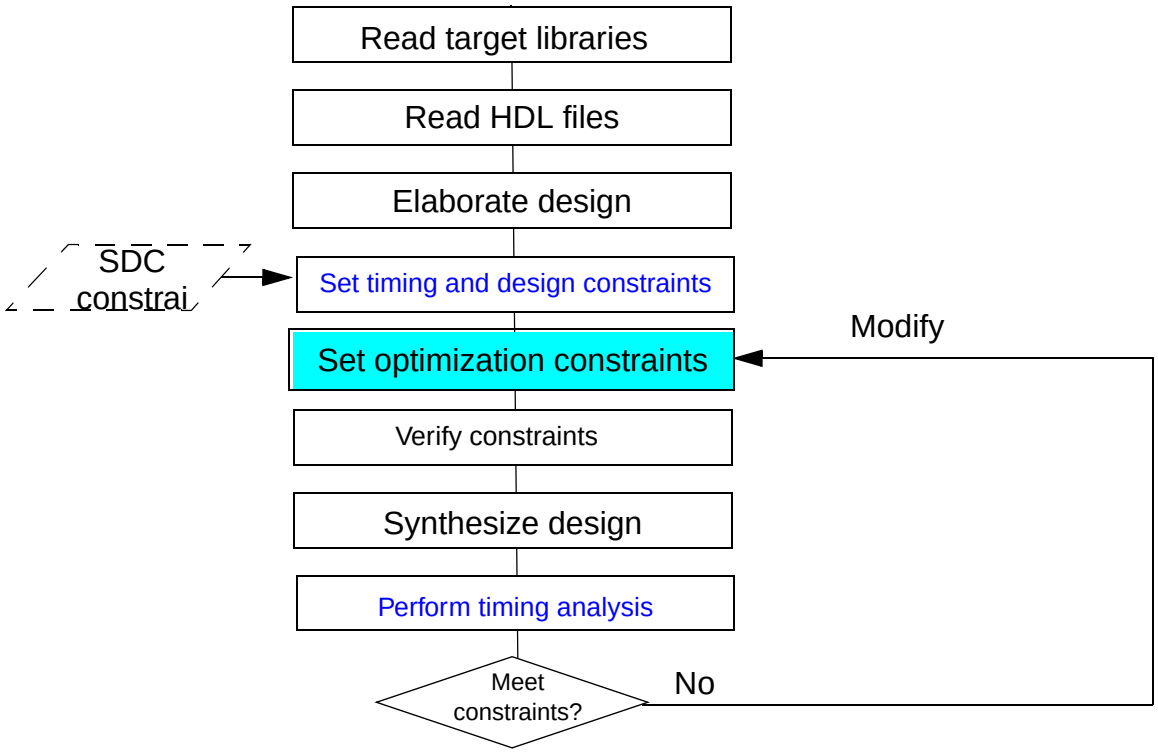
Overview of Optimization Constraints

This chapter describes how to set optimization constraints before synthesis.

In addition to applying design constraints, you can use optimization strategies to get the desired performance goals from synthesis. You can

- Preserve instances and modules
- Group and ungroup instances or subdesigns
- Control boundary optimization of hierarchical instances
- Map to complex sequential cells
- Flatten arrays and memories
- Specify ideal nets
- Disable timing arcs
- Create path groups and cost groups
- Optimize total negative slack
- Make DRC the highest priority

Figure 2-1 Timing Analysis Flow in Single Mode



Creating Path Groups and Cost Groups

Important

By default, RTL Compiler attempts to optimize a timing path with the worst negative slack (WNS). If RTL Compiler is able to optimize the WNS path so that it meets timing, then RTL Compiler optimizes the path with the next WNS. This continues until all paths meet their timing goals. However, if in this process, RTL Compiler is unable to optimize a path so that it meets timing, then it will not move to the next worst path, and RTL Compiler will halt the optimization process.

To work around this default behavior, group timing paths into different cost groups. When multiple cost groups exist, RTL Compiler will optimize the WNS path in each cost group. If it cannot meet the timing goal for the WNS path in a cost group, then RTL Compiler will continue to try and optimize the WNS paths in each of the other cost groups.

A *cost group* is a set of critical paths to which you can apply weights or priorities that the optimizer will recognize. Paths assigned to a cost group are called path groups.

A *path group* is a group of related paths, grouped either by the `define_clock` command or by the `path_group` command.

Important

By default, paths whose endpoints are clocked by the same clock are assigned to the same path group.

Organizing Paths into Groups

Organize timing paths in your design into the following four cost groups:

- Input-to-Output paths (I2O)
- Input-to-Register paths (I2C)
- Register-to-Register (C2C)
- Register-to-Output paths (C2O)

Organizing paths in the design into groups is helpful when generating timing analysis reports. By default, the timing report shows the critical path from each path group. The critical path is the timing path in the design with the greatest amount of negative slack (margin).

The value or weight that you specify for a cost group is not meaningful as an absolute value, but as a ratio to the weight of other cost group paths. For example, if you create three cost groups with weights of 1, 1, and 2, then the cost group with a weight of 2 will get twice the effort from RTL Compiler as the groups with a weight of 1.

The mapper sets a different target, the worst or most critical path for each cost group, and works on it separately. RTL Compiler performs incremental optimization on each path separately. The costing function takes into account the weight of the cost group and multiplies it to the worst slack for scoring.

To set up path groups:

1. Create the cost group using the `define_cost_group` command.
2. Add a specified path group to the cost group using the `path_group` command.

As with all other timing exceptions, at least one of the `to`, `through`, or `from` options is required. The `path_group` exception works well with other exceptions.

➤ To control the focus of optimization, you can specify particular cost and path groups for optimization. For example, it is common to use this method to group all register-to-register timing paths:

```
rc:/> set_all_regs [find / -instance instances_seq/*]
rc:/> define_cost_group -name C2C
rc:/> path_group -from $all_regs -to $all_regs -group C2C -name C2C
```

The effect of creating a single “super group” of all register-to-register timing paths is that, instead of optimizing the worst path for each individual clock-domain cost group separately, RTL Compiler will optimize the single worst path among all of these paths, and after meeting timing on that path, will proceed to the next worst path, and so on. This is often a more efficient method of using CPU resources since the focus is on spending optimization cycles on the worst flop-to-flop paths in the entire design.

This mechanism can be used productively in conjunction with the `path_adjust` command and with total negative slack optimization. However, there are situations in which it does not make sense to group all register to register paths, so this method should not be applied blindly, as with all aspects of timing optimization during synthesis. It is crucial to understand the clocking and timing scheme of the design prior to setting up path and cost groups to control the optimization process.

Cost Group Examples

The following examples show how to use the `define_cost_group` and `path_group` commands.

To set up a cost group called `critical1`:

1. Create the cost group by typing:

   ```
   rc:/> define_cost_group [-weight 2] -name critical1
   ```

   The cost group is kept in `/designs/test/timing/cost_groups/critical1`.

2. Add the path starting from `data_in` to the cost group called `critical1`:

   ```
   rc:/> path_group -from data_in -name exception2 -group critical1
   ```

   The path group is kept in `/designs/top_counter/timing/exceptions/path_groups/exception2`.

   The SDC equivalent is:

   ```
   rc:/> dc::group_path -from data_in -exception_name exception2 -name critical1
   ```

To create more path groups for the `critical1` cost group:

1. Define the starting point and cost group by typing:

   ```
   rc:/> path_group -from l0/ENA -group critical1
   ```

   The SDC equivalent is:

   ```
   rc:/> dc::group_path -from l0/ENA -name critical1
   ```

2. Define the endpoint for the path group by typing:

   ```
   rc:/> path_group -to l0/ENA -group critical1
   ```

   The SDC equivalent is:

   ```
   rc:/> dc::group_path -to l0/ENA -name critical1
   ```

To set the characteristics of a cost group:

1. Get a report of the current characteristics by typing:

   ```
   rc:/designs/test/timing/cost_groups>ls -l -a critical1
   ```

   The following information is displayed.

   ```
   /designs/test/timing/cost_groups/critical1 (cost_group)
   All attributes:
   exceptions =
   weight = 1
   ```

2. Overwrite the weight by typing:

   ```
   rc:/designs/test/timing/cost_groups> set_attribute weight 2 critical1
   ```

The following status message is displayed.

```
Setting attribute of cost_group critical1: 'weight' = 2
```

Defining a Cost Group Between Two Clock Domains

To define a cost group between two clock domains, for example if a design has multiple domains and you want an easy way to find the slack within each clock domain and also the slack in the clock domain crossings, use the following commands.

1. The following example defines a cost group:

   ```
   define_cost_group -name clk_fsample2clk_f2x
   ```

2. The following example binds the `path_group` command to the defined cost group using the `-group` option of the `path_group` command:

   ```
   path_group -from [all des seqs] -clock clk_fsample -to [all des seqs] -clock
   clk_f2x -name clk_fsample2clk_f2  -group clk_fsample2clk_f2x
   ```

➤ To get information for paths for specified cost groups, use the `report timing -cost_group` command. For example:

```
report_timing -cost_group clk_fsample2clk_f2x
```

Decreasing Path Group Runtime and Memory

When you define a `path_group` command, RTL Compiler tries to work on all the groups simultaneously. Runtime and memory will increase when you have too many path groups.

➤ Therefore, if you have critical paths in a particular clock domain that you want to focus on, then specify them as a path group using the following commands:

```
define_cost_group –name clka_reg_reg
path_group –from  [all des seqs] –clock clka
```

When you have hundreds of clocks, do not specify the `I2C`, `C2O`, `C2C`, and `I2O` clocks all at the same time. Instead, there is a more efficient way of putting only the clocks you want in a particular clock group with regards to the clock domain, as shown in Example 2-1.

Example 2-1 Setting Path Groups

```
foreach clock [find / -clock *] {
   set clock [basename $clock]
   set excludeList {*virage90_ram*}
   set seqList [all des seqs -clock ${clock} -exclude ${excludeList}]
   set inList [all des inps -clock ${clock}]
   set outList [all des outs -clock ${clock}]
   if {[llength $seqList] > 0} {
      if {[llength $inList] > 0} {
         define_cost_group -name "I2${clock}"
         path_group -from $inList -to $seqList -group I2${clock} -name I2${clock}
      }
      if {[llength $outList] > 0} {
         define_cost_group -name "${clock}2O"
        path_group -from $seqList -to $outList -group ${clock}2O -name ${clock}2O
      }
      define_cost_group -name "${clock}2${clock}"
      path_group -from $seqList -to $seqList -group ${clock}2${clock} -name
${clock}2${clock}
   }
}
```

Use the `set excludeList` to fit your needs. For example, if you do not want the timing models to be included, then use the `set excludeList {*virage90_ram*}`, as shown in Example 2-1.

See Analyzing a Design Using the Timing Report on page 202 for information on how to troubleshoot critical path types.

Modifying Path Constraints

Direct timing optimizations for specific paths using the `path_adjust` command.

- Use the `path_adjust` command to modify path constraints that were either computed by the timing engine using the launching and capturing waveforms or set explicitly with the `path_delay` command.

For example, in many cases you want to focus on optimizing the register to register timing during synthesis, while not prioritizing the optimizations of input and output paths. The following example uses the `path_adjust` command to conveniently achieve this kind of optimization:

```
rc:/> set all_regs [find / -instance instances_seq/*]
rc:/> path_adjust -from [all_inputs] -to [all_outputs] -delay 500 -name PA_I2O
rc:/> path_adjust -from [all_inputs] -to $all_regs -delay 500 -name PA_I2C
rc:/> path_adjust -from $all_regs -to [all_outputs] -delay 500 -name PA_C2O
rc:/> path_adjust -from $all_regs -to $all_regs -delay -500 -name PA_C2C
```

The first three `path_adjust` commands relax the constraints on all input and output paths by 500 picoseconds, while the fourth command tightens the constraints on all register to register timing paths by 500 picoseconds. A positive delay value relaxes the path timing, while a negative delay value tightens it.

Use the `path_adjust` command with cost groups to direct the optimization focus in RTL Compiler.

The constraints created by the `path_adjust` command are found in the following RTL Compiler design directory:

`/designs/`*`design_name`*`/timing/exceptions/path_adjusts`

See the “RTL Compiler Design Information Hierarchy” chapter in the *Using Encounter RTL Compiler* manual for detailed information on the design hierarchy.

Constraining Input Delays for Different Paths

If your design has input ports that are start points in both input to register (i2r) and input to output (i2o) paths, you can specify timing constraints such that the `i2r` portion of the path uses one input delay, and the `i2o` path uses another by performing the following steps:

1. Add an exception to tighten up the path between the input to output (i2o) path by setting the `external_delay` on the input and output ports.
2. Use the `path_adjust` constraint to tighten the specified paths from the input to the output ports.

Preserving Instances and Modules

Prevent RTL Compiler from performing optimizations on specific objects in the design using the preserve attribute. By default, RTL Compiler will perform optimizations that can result in logic changes to any object in the design. You can prevent any logic changes in a block while still allowing mapping optimizations in the surrounding logic. For example:

- Preserve hierarchical instances using the following command:

  ```
  rc:/> set_attribute preserve true object
  ```

  where *object* is a hierarchical instance name.

- Preserve primitive instances using the following command:

  ```
  rc:/> set_attribute preserve true object
  ```

  where *object* is a primitive instance name.

  The following command preserves the mod1 instance:

  ```
  set_attribute preserve true [find / -inst mod1]
  ```

  If the mod1 instance has mod2 and mod3 instances in its hierarchy, then setting the preserve attribute to true on the mod1 instance will inherit this setting on the mod2 and mod3 instances. In other words, the inherited_preserve attribute will be set to true.

- Preserve modules or submodules using the following command:

  ```
  rc:/> set_attribute preserve true object
  ```

  where *object* is a module or sub-module name.

In RTL Compiler you can preserve and protect your existing mapped logic from being further synthesized or changed. Preserve logic with the preserve attribute on the desired module or instance.

Use the preserve attribute on a mapped sub-design to prevent any logic changes in the block while allowing mapping optimizations to be performed in surrounding logic.

The default value of this attribute is false. Therefore, you must set it to true to preserve the module. The following example preserves the module counter:

```
rc:/> set_attribute preserve true [find / -subdesign counter]
```

The following example preserves the mapped leaf cell instance g1:

```
rc:/> set_attribute preserve true [find / -instance g1]
```

This command will force logic optimization to preserve the current mapping of the g1 instance.

Modifying Preserved Instances

RTL Compiler offers the following additional options to give you more flexibility with preserved instances or modules during synthesis:

- Use the `size_ok` value to preserve an instance (`g1` in the example below), while allowing it to be resized.

  ```
  rc:/> set_attribute preserve size_ok [find / -instance g1]
  ```

- Use the `delete_ok` value to delete an instance (`g1` in the following example), but not to rename, remap, or resize it.

  ```
  rc:/> set_attribute preserve delete_ok [find / -instance g1]
  ```

- Use the `size_delete_ok` value to resize or delete an instance (`g1` in the example below), but not to rename or remap it.

  ```
  rc:/> set_attribute preserve size_delete_ok [find / -instance g1]
  ```

- Use the `map_size_ok` value to resize, unmap, and remap a mapped sequential instance during optimization, but not to rename or delete it.

A `preserve` set on a cell is marked in a timing report, as shown in Example 2-2.

Example 2-2 Timing Report Showing the Preserve Set on an Instance

```
    Pin              Type     Fanout Load Slew Delay Arrival
                                     (fF) (ps)  (ps)    (ps)
-------------------------------------------------------------
(clock clock)       launch                                0 R
(in_del_1)          ext delay                  +0         0 R
in1[9]              in port        1 10.4    0    +0       0 R
stress_from_groupi/in1[9]
  i9/B                                         +0         0
  i9/Y         (P)  nand2          1 20.4   46  +123     123 F
  f9/D         (p)  fflopd                     +0     123
  f9/CK             setup                    0  +100     223 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock)       capture                              10 R
-------------------------------------------------------------
Timing slack :    -213ps (TIMING VIOLATION)
Start-point  : in1[9]
End-point    : stress_from_groupi/f9/D
(p) : Instance is preserved but may be resized
(P) : Instance is preserved
```

Grouping and Ungrouping Objects

RTL Compiler provides a set of commands that let you group or ungroup any existing instances, designs, or subdesigns. Grouping and ungrouping are helpful when you need to change your design hierarchy as part of your synthesis strategy.

- *Grouping* builds a level of hierarchy around a set of instances.
- *Ungrouping* flattens a level of hierarchy.

Grouping Objects

If your design includes several subdesigns, then you can group some of the subdesign instances into another single subdesign for placement or optimization purposes using the `edit_netlist group` command.

For example, the following command creates a new subdesign called `CRITICAL_GROUP` that includes instances `I1` and `I2`.

```
rc:/> edit_netlist group -group_name CRITICAL_GROUP [find / -instance I1] \
        [find / -instance I2]
```

The module name for this new hierarchy is CRITICAL_GROUP. The instance name for this new hierarchy is `CRITICAL_GROUP`i and it will be placed in the following directory:

```
/designs/top_counter/instances_hier/CRITICAL_GROUPi
```

Ungrouping Objects

To flatten a hierarchy in the design, use the `ungroup` command:

```
rc:/> ungroup instance
```

where *`instance`* is the name of the instances to be ungrouped.

If you need to ungroup the design hierarchy `CRITICAL_GROUP` (which contains instances `I1` and `I2`), then use the `ungroup` command along with the instance name. For example:

```
rc:/> ungroup [find / -instance CRITICAL_GROUPi]
```

RTL Compiler will not ungroup any preserved instance in the hierarchy. For more information on preserving instances, see “Preserving Instances and Modules” on page 104.

Setting Boundary Optimization

RTL Compiler performs boundary optimization for all hierarchical instances in the design during synthesis. Boundary optimization moves logic up in the hierarchy. You can control boundary optimization during synthesis using the boundary_opto attribute. For example, if you instantiate an inverter in a lower level module whose output drives an output at the top level, then setting the `boundary_opto` command to `true` will move the inverter to the top level in the synthesized netlist if it improves QOR. If you set the `boundary_opto` command to `false`, then the inverter remains at the level it was instantiated in.

However, If you want to keep the inverter at the lower level module and still perform boundary optimization on that module, set the `preserve` attribute on the inverter to `true`.

Examples of boundary optimizations include:

- Constant propagation across hierarchies

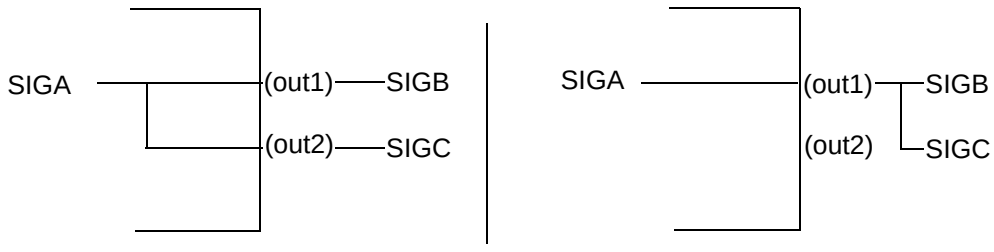
  Both through input ports and output ports

- Rewiring of equivalent signals across hierarchy

  If two outputs of a module are identical, then RTL Compiler may disconnect one of them and use the other output to drive the fanout logic for both.

RTL Compiler may also rewire signals that are functionally equivalent, as illustrated in Figure 2-2. The figure on the left shows the `SIGA` signal fanning to two output ports, `out1` and `out2`. These are connected to the `SIGB` and `SIGC` signals, respectively. The figure on the right shows that RTL Compiler can change that signal so that the `SIGB` and `SIGC` signals are both connected to `out1`.

Figure 2-2 Rewiring Equivalent Signals



If you cannot perform top-down formal verification on the design, then turn off boundary optimization for sub-blocks that will be individually verified.

➤ To disable boundary optimization on the subdesign, type the following command:

```
rc:/> set_attribute boundary_opto false [find /des* -subdesign name]
```

Changing the `boundary_opto` attribute to `false` prevents RTL Compiler from doing the following optimizations:

- Propagating constants into or out of a module
- Rewiring feed-through signals
- Rewiring complimentary signals through modules
- Deleting cells that are connected to the specified `boundary_opto false` module if they are not read outside of the module, and deleting cells connected to the inputs of a specified `boundary_opto false` module if they are not read inside of the module

Note: Setting the `boundary_opto` attribute to `false` should be the same as if RTL Compiler had perfect timing characterization and compiled the `boundary_opto false` module at the top-level, for example without logical context.

Deleting Unused Sequential Instances

➤ By default, RTL Compiler removes flip-flops and logic if they are not transitively fanning out to output ports. To prevent this, use the `delete_unloaded_seqs` attribute:

```
rc:/> set_attribute delete_unloaded_seqs false [subdesigns or /]
```

Setting this attribute may have a negative impact on area and timing.

RTL Compiler optimizes sequential instances that transitively do not fanout to primary outputs. This information is generated in the log file, as shown below. This is especially relevant if you see unmapped points in formal verification.

```
Deleting 2 sequential instances. They do not transitively drive any primary
outputs:
   ifu/xifuBtac/xicyBtac/icyBrTypeHold1F_reg[1] (floating-loop root), ifu/
xifuBtac/xicyBtac/icyBrTypeHold1T_reg[1]
```

➤ To prevent constant `0` propagation through flip-flops, set the `optimize_constant_0_flops` attribute to `false`. The default value of this attribute is `true`.

```
rc:/> set_attribute optimize_constant_0_flops false /
```

➤ To prevent constant `1` propagation through flip-flops, set the `optimize_constant_1_flops` attribute to `false`. The default value of this attribute is `true`.

```
rc:/> set_attribute optimize_constant_1_flops false /
```

➤ To prevent constant propagation through latches, set the optimize_constant_latches attribute to false. The default value of this attribute is true.

```
rc:/> set_attribute optimize_constant_latches false /
```

Optimizing Total Negative Slack (TNS)

The timing of a circuit is measured in terms of two metrics: worst negative slack (WNS) and total negative slack (TNS). The slack at a timing point is the difference between the required and arrival times of the signal at that timing point. A negative slack indicates that the signal is arriving later than the required time. The WNS is the worst slack among all timing endpoints, while *TNS* is the sum of all the worst negative slacks per endpoint.

➤ Return the sum (or total) of all worst negative slacks of all endpoints in the design (or in a cost group) using the tns attribute. For example:

```
rc:/> get_attribute tns /designs/foo
```

If the sum is positive, then a value of 0 will be reported. For example, assume a design has two A and B endpoints whose worst slack is -10 and +10, respectively. In this case, the tns value of the design returns -10 because the slack value of +10 is treated as zero slack.

To simultaneously produce the smallest possible design and the best possible timing results on critical paths, RTL Compiler, by default, sacrifices timing slack to minimize area on non-critical paths. However, these non-critical paths may still violate timing. You can direct RTL Compiler to optimize non-critical endpoints for timing and to sacrifice timing slack only on non-violating endpoints by using TNS optimization.This usually results in a longer runtime.

➤ Optimize all violating paths for timing by setting the following attribute to true:

```
rc:/> set_attribute tns_opto true /
```

Setting this attribute to true will optimize all violating paths for timing, but it may also result in a netlist with a larger instance count, area, or both than if it were left to its default value of false.

For example, if you have a simple design with two timing endpoints called E1 and E2 during optimization, then RTL Compiler considers the following two circuit implementations:

1. E1 slack -300ps

E2 slack -250ps

Design area is 100

2. E1 slack -300ps

   E2 slack -100ps

   Design area is 150

By default, RTL Compiler considers the first implementation the best, because the critical slack (-300) is the same and the circuit is smaller (100 versus150).

However, if the `tns_opto` attribute is set to `true`, then the second implementation is the better implementation method because the sum of the negative slacks across all endpoints is better (-400ps versus -550ps).

Creating Hard Regions

Use the `hard_region` attribute to specify hierarchical instances that are recognized as hard regions in your floorplan during logic synthesis and to preserve pins and subports.

Place and route tools operate better if your design has no buffers between regions at the top level. To accommodate this, specify hard regions before mapping.

To create hard regions, follow these steps:

1. Specify the hard region, for example `pbu_ctl`, by typing:

   ```
   set_attribute hard_region 1 [find / -instance pbu_ctl]
   ```

   Primary inputs and outputs are also treated as hard regions for this purpose.

2. Run the `synthesize -to_mapped` command.

The regular boundary optimization related controls are also applicable to hard regions.

Making Design Rule Constraints (DRC) the Highest Priority

By default, RTL Compiler tries to fix all DRC violations but not at the expense of timing. If DRCs are not being fixed, then it could be because of infeasible slew issues on input ports or infeasible loads on output ports. You can force RTL Compiler to fix DRCs, even at the expense of timing, with the `drc_first` attribute.

➤ To ensure DRCs get fixed, even at the expense of timing, set the following attribute:

```
rc:/> set_attribute drc_first true
```

By default, this attribute is `false`, which means that DRCs will not be fixed if it introduces timing violations.

The design rule constraints are optimized in the following order:

1. `max_transition`
2. `max_capacitance`
3. `max_fanout`

Note: This attribute applies only to incremental optimization.

3

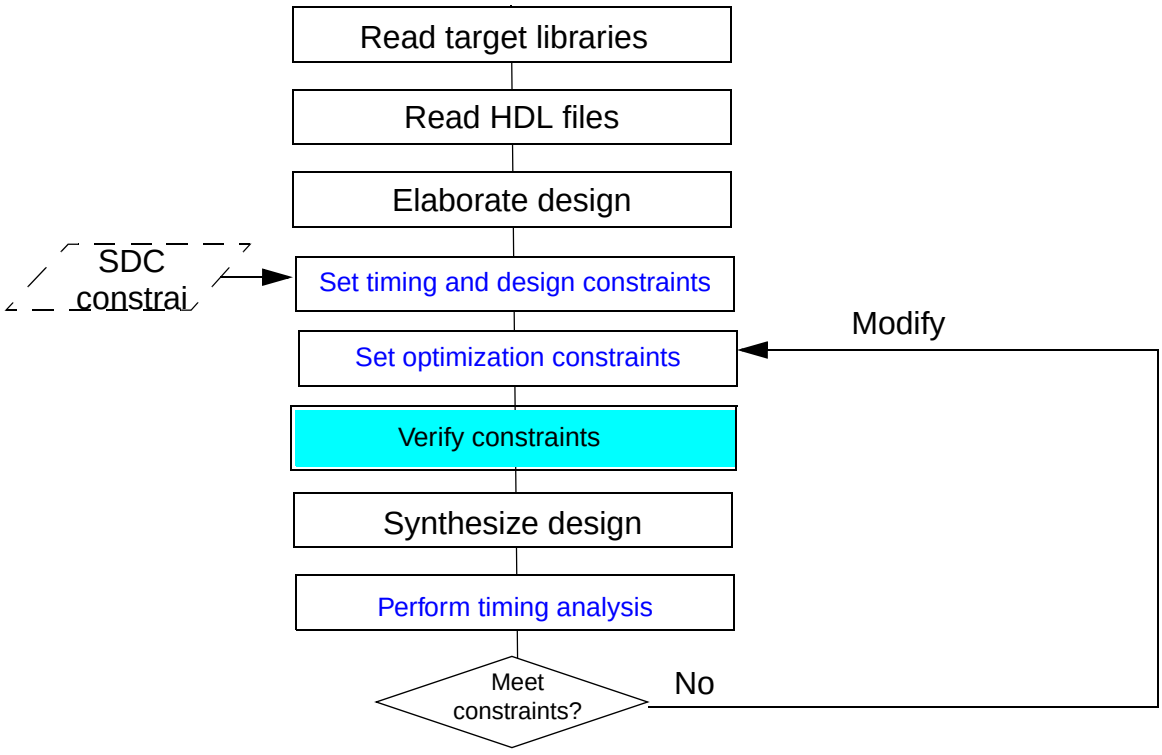
Verifying Constraints

Overview of Verifying Constraints

After defining the clocks and setting the design and timing constraints, verify the constraints before synthesis. Verifying the constraints helps identify potential problems or timing assertions in the design that can make timing analysis incomplete or inaccurate, such as undefined clocks, unconstrained ports, unconstrained clock pins of registers, undefined input arrival times, or undefined output constraints.

Paths that are not constrained correctly may not appear in timing violation reports when you perform timing analysis. For this reason, generate pre-synthesis timing reports to check a new design or a design with new constraints.

Figure 3-1 Timing Analysis Flow in Single Mode



Generating Pre-Synthesis Timing Reports

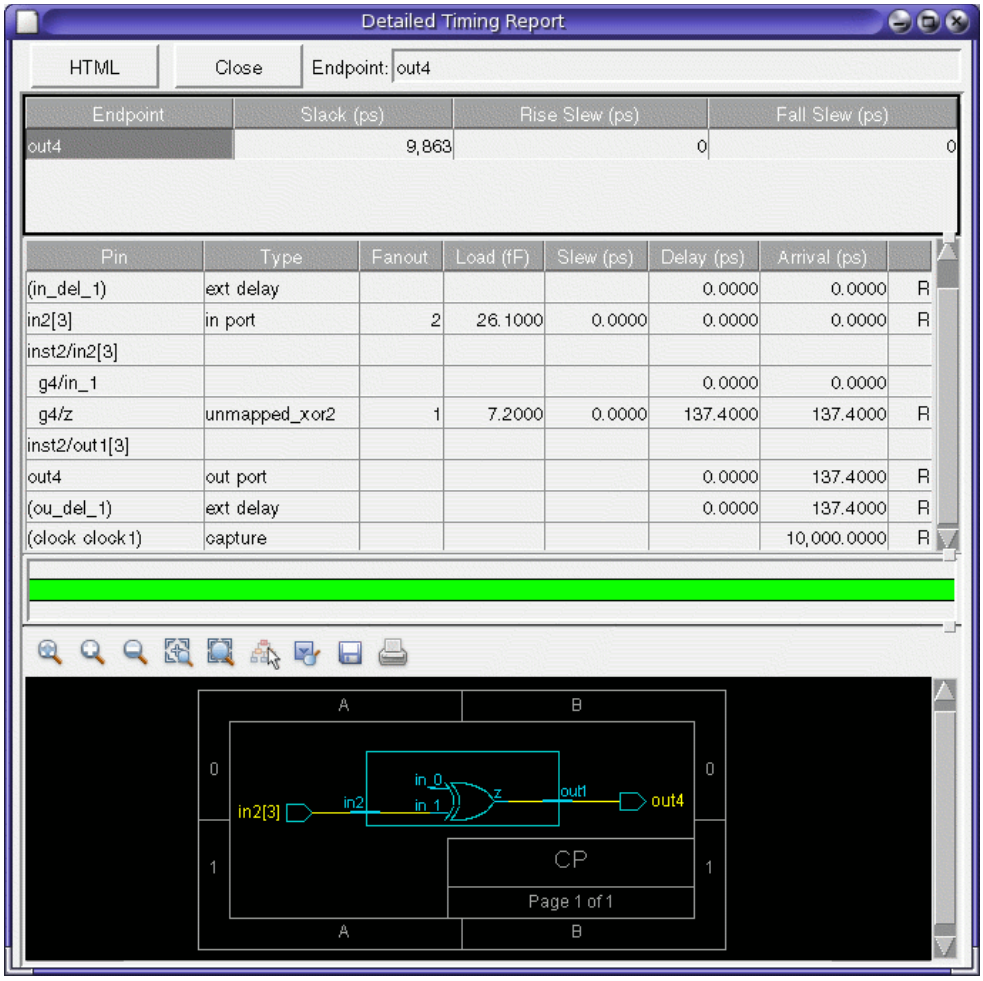
Generating timing reports before synthesis saves time because they can catch errors, such as improper constraint values or missing constraints. The most common report commands used to verify constraints before synthesis are:

- `report timing` or `report timing -endpoint`

  Use to generate a default timing report on the timing of the current design and a detailed view of the most critical path in the current design. However, this report may not be as accurate because the netlist is in an unmapped state; thus, the slack numbers may not correlate with the slack numbers after synthesis.

- `gui_report_timing`

  Use the GUI to visualize the timing report along with its pre-synthesis schematic.



- `report timing -lint`

  Use this command immediately after reading in SDC and timing constraints to catch some of the worst constraint problems, such as incorrect constraints, missing information, exceptions that either override each other, are incorrectly specified, or unsatisfied. It also reports missing clocks, unconstrained ports, and unconstrained clock pins of registers. Use this command to get information on loop-breaking cells. See Handling Combinational Feedback Loops on page 201 for detailed information.

  See Checking the Constraints Using the report timing -lint Command on page 118 for detailed information on constraint checks.

- `report timing -exception` or `report timing -cost_group`

  Use these commands depending on the outcome of the `report timing -lint` report. For example, if the `-lint` report shows an exception that cannot be satisfied, because another exception has a higher priority, then it can useful to run the `report timing -exception` command to see why there are such conflicting exceptions.

  See Tips for Reporting Timing Issues on page 196 for more information.

To generate timing reports in a multi-mode environment, see Chapter 5, “Performing Multi-Mode Timing Analysis,”

Generating a Default Timing Report

➤ To generate a timing report, `cd` into the appropriate design directory and type the following command:

```
rc:/designs/top> report timing
```

The timing report provides the following information:

- Type of cell: flip-flop, or, nor, and so on
- The cell's fanout and timing characteristics: load, slew, and total cell delay
- Arrival time for each point on the most critical path

If timing is not met, then the *Timing Slack* is reported with a minus (-) number and a *TIMING VIOLATION* is written out.

RTL Compiler generates the timing report down to the gate and net level.

Customizing the Timing Report

➤ Use the `exceptions` or `cost_group` options to generate timing reports for any of the previously set timing exception names, or for the set of path group names defined by the `cost_group` command. Use these options to help generate custom timing reports for the paths that you previously assigned to cost groups.

```
rc:/designs/top> report timing -exceptions <exception_name>
```

or

```
rc:/designs/top> report timing -cost_group <cost_group_name>
```

For more information on creating cost groups, see Creating Path Groups and Cost Groups on page 98.

➤ Use the `-lint` option to generate timing reports at different stages of synthesis.

```
rc:/designs/top> report timing -lint
```

This report gives you a list of possible timing problems due to over constraining the design or incomplete timing constraints, such as not defining all the multicycle paths or false paths. See Checking the Constraints Using the report timing -lint Command on page 118 for detailed information.

Checking the Constraints Using the report timing -lint Command

The `report timing -lint` command checks for the following constraint issues.

Exceptions

- Conflicting exceptions

  Occurs when an exception cannot be satisfied because another exception has a higher priority, such as timing exceptions that are satisfied by some paths but the paths that satisfy them also satisfy an exception with a higher priority.

  Use the `report timing -exception` command to see why there are such conflicting exceptions.

  If there are conflicting timing exceptions in your design, then RTL Compiler reports them as follows:

  ```
  The following timing exceptions are not currently affecting timing in the
  design. Either no paths in the design satisfy the exception's path
  specification, or all paths that satisfy the path specification also satisfy
  an exception with a higher priority. Improve runtime and memory usage by
  removing these exceptions if they are not needed.To see if there is a path in
  the design that satisfies the path specification for an exception, or to see
  what other exception is overriding an exception because of priority, use the
  following command:

  report timing -paths [eval [get_attribute paths <exception>]]
  ```

- Suspicious exceptions

  Occurs when there are exceptions that are not satisfied by any paths, are never used, or conflict with others. If an exception is not satisfied by any path in the design, then it is reported. See Reporting Path Exceptions on page 145 for more information.

  If you have suspicious exceptions in your design, then RTL Compiler reports them as follows:

  ```
  The following timing exceptions have -from points that are not valid timing
  startpoints or -to points that are not valid timing endpoints. If the exception
  is intended to constrain paths that pass through these points, use the -through
  option instead when creating the timing exception.If these points were
  specified inadvertantly, re-write the exception to not include them. Currently
  these exception points are being ignored.
  ```

- Multi-Cycle exceptions shifting different paths by different amounts

  Occurs when a multi-cycle exception is being used to shift paths by different amounts. For example, shifting one path by 10 n.s. and another by 8 n.s. if the clock periods are different at the flops involved. This indicates that possibly the exception is being applied more broadly than intended and RTL Compiler reports them as follows:

  ```
  Exception: /designs/top/timing/exceptions/multi_cycles/mcA
  ```

```
  Shifts these paths by 200.0 ps:
    report timing -exception /designs/top/timing/exceptions/multi_cycles/mcA \
      -from /designs/top/timing/clock_domains/domain_1/ck100 \
      -to /designs/top/timing/clock_domains/domain_1/ck200
  And shifts these paths by 300.0 ps:
    report timing -exception /designs/top/timing/exceptions/multi_cycles/mcA \
      -from /designs/top/timing/clock_domains/domain_1/ck100 \
      -to /designs/top/timing/clock_domains/domain_1/ck300

Exception: /designs/top/timing/exceptions/multi_cycles/mcB
  Shifts these paths by 100.0 ps:
    report timing -exception /designs/top/timing/exceptions/multi_cycles/mcB \
      -from /designs/top/timing/clock_domains/domain_1/ck100 \
      -to /designs/top/timing/clock_domains/domain_1/ck200
  And shifts these paths by 300.0 ps:
    report timing -exception /designs/top/timing/exceptions/multi_cycles/mcB \
      -from /designs/top/timing/clock_domains/domain_1/ck300 \
      -to /designs/top/timing/clock_domains/domain_1/ck200
```

External Delays

- Missing external delays

  Primary inputs have input external delays. This is the delay arriving at this input port with respect to a certain clock, which is an internal or external clock. Output ports have output external delays, which are required times for ports with respect to a clock. If missing, then RTL Compiler reports them, as shown in Example 3-1.

Example 3-1 Reporting Missing External Delays

```
============================================================
The following sequential clock pins have no clock waveform driving them.No timing
constraints will be derived for paths leading to or from these cells.
  delay_pipeline_reg[0][0]/CLK (DFFSR/CLK)
  delay_pipeline_reg[0][1]/CLK (DFFSR/CLK)
  delay_pipeline_reg[0][2]/CLK (DFFSR/CLK)
  ... 435 other clock pins in this category. Use the -verbose option for more details
The following primary inputs have no clocked external delays.As a result the timing
paths leading from the ports have no timing constraints derived from clock
waveforms.The 'external_delay' command is used to create new external delays.
  /designs/filter16opt/ports_in/clk
  /designs/filter16opt/ports_in/clk_enable
  /designs/filter16opt/ports_in/filter_in[0]
  ... 16 other pins in this category. Use the -verbose option for more details.


The following primary outputs have no clocked external delays.As a result the timing
paths leading to the ports have no timing constraints derived from clock waveforms.
The 'external_delay' command is used to create new external delays.
  /designs/filter16opt/ports_out/filter_out[10]
  /designs/filter16opt/ports_out/filter_out[20]
  /designs/filter16opt/ports_out/filter_out[19]
  ... 19 other pins in this category. Use the -verbose option for more details.
```

Clocks

- Missing or undefined clocks

  Occurs when sequential clock pins do not have clocks defined or propagated to them, as well as sequential clock pins that are either unconnected or driven by a logic constant. As a result, the sequential element (flop or latch) would have no timing constraint, since it does not have a clock to launch and capture.

- Suspicious clocking,

  Occurs when you have unclocked registers, multiple-clocked registers, or strange clock frequencies. If you have suspicious clocks in your design, then RTL Compiler reports them as follows:

  - `The following sequential clock pins are either unconnected or driven by a logic constant.`
  - `The following sequential data pins are driven by a clock signal.`

- The following sequential clock pins have no clock waveform driving them. No timing constraints will be derived for paths leading to or from these cells.
- The following sequential clock pins have multiple clock waveforms from a single clock domain driving them. Timing analysis will use all of the waveforms to determine the most pessimistic timing constraints. Use the PROPAGATED_CLKS_ATTR_NAME pin attribute to analyze which clocks are being used at each clock pin.
- The following generated clocks have no clock waveform to generate from.
- There are paths in the design that are constrained by two different clocks, and the relationship between the periods is not an integer multiple. Verify that the periods of the clocks were defined correctly. Use the following command to report a path for which this occurs: report timing -from <startpoint> -to <endpoint>. You can access the clock period using the period and the divide_period clock attributes on the clock object.

Multiple Drivers

Occurs when you have multiple driven nets in your design and at least one of the drivers is non-tristate. RTL Compiler reports them as follows:

```
The following nets have multiple drivers and at least one of the drivers is non-
tristate.
```

Combinational Loops

- Suspicious logic, such as non-tristate parallel drivers or combinational loops
- Loop-breaking cells

  Occurs when the design contains loop-breaking cells that were inserted automatically to eliminate combinational feedback. Since timing paths have been arbitrarily broken, then timing results for this design may not be reliable. See Handling Combinational Feedback Loops on page 201 for detailed information.

  When you have combinational loops in your design, then RTL Compiler reports them as follows:

  ```
  The design contains loop-breaking cells that were inserted automatically to
  eliminate combinational feedback. Since timing paths have been arbitrarily
  ```

```
broken, timing results for this design should not be trusted. The loop-breaking
cells are:
```

Conflicting Case Constraints

Occurs when constants have conflicting values on ports or pins. RTL Compiler reports them as follows:

```
The following port <port_names> or pin <pin_names> have conflicting case constants
defined on them. Check the value of the'timing_case_logic_value' attribute defined
on these pins. These are different than the propagated case constants at these pins.
To see propagated constant values for these pins, either unset the
'timing_case_logic_value'attribute on these pins or the
'timing_case_computed_value' attribute on its driver pin.
```

Unconstrained Points

Reports unconstrained ports, and unconstrained clock pins of registers.

Multimode

Occurs when you have timing paths for pins in different timing modes.

See “Breaking the Path of a Specified Pin on page 79” for more information.

Generating Clock Reports

- Use the `report clocks` command, as shown in Example 3-2 to display information about clocks, including the clock period, rise, fall, domain, setup uncertainty, latency, and clock ports or sources in the current design.

  Use the `-generated` option to report generated clock information, and use the `-ideal` option to report an ideal clock - clock relationship.

To report clocks in a multi-mode environment, see Chapter 5, “Performing Multi-Mode Timing Analysis,”

Example 3-2 Clock Report

```
==========================================================================
 Clock Description
 -----------------
 Clock                                                  No of
 Name    Period  Rise    Fall     Domain  Pin/Port  Registers
--------------------------------------------------------------
 CLK1    4000.0   0.0   2000.0   domain_1   Clk             5
 CLK2    2000.0   0.0   1000.0   domain_1   C               0
 CLK3    3000.0   0.0   1500.0   domain_2   Clk1            5
 CLK4    6000.0   0.0   3000.0   domain_2   C1              0
 Clock Network Latency / Setup Uncertainty
 -----------------------------------------
        Network   Network   Source   Source      Setup         Setup
Clock   Latency   Latency  Latency  Latency  Uncertainity  Uncertainity
 Name    Rise      Fall      Rise     Fall       Rise          Fall
--------------------------------------------------------------------------
CLK1      140.0     140.0    150.0    150.0         100.0         100.0
CLK2      120.0     120.0    120.0    120.0         110.0         110.0
CLK3      100.0     100.0    100.0    100.0         100.0         100.0
CLK4        0.0       0.0      0.0      0.0           0.0           0.0
 Clock Relationship (with uncertainity & latency)
 ------------------------------------------------
  From    To     R->R     R->F     F->R     F->F
-------------------------------------------------
  CLK1   CLK1   3900.0   1900.0   1900.0   3900.0
  CLK1   CLK2   1840.0    840.0   1840.0    840.0
  CLK2   CLK1   1950.0   1950.0    950.0    950.0
  CLK2   CLK2   1890.0    890.0    890.0   1890.0
  CLK3   CLK3   2900.0   1400.0   1400.0   2900.0
  CLK3   CLK4   2800.0   2800.0   1300.0   1300.0
  CLK4   CLK3   3100.0   1600.0   3100.0   1600.0
  CLK4   CLK4   6000.0   3000.0   3000.0   6000.0
```

Generating a Port Report

➤ Generate timing information on the ports in the design using the `report port` command.

By default, the report gives information on port direction, external delays, driver, slew, fanout load, pin capacitance, and wire capacitance for the ports.

For example, the following options for the `report port` command reports the external delay information on the `ck1`, `e_out[6]`, and `ena` ports, as shown in Example 3-3.

- Use the `-delay` option to report external delay information, which is rise and fall delay and the external delay object.
- Use the `-driver` option to report the external driver name and the rise and fall values (slew) of the ports.
- Use the `-load` option to report the external fanout load and the pin and wire capacitance values of the ports.
- Use the *`port`* option to specify the port, and use the *`file`* option to specify the name of the file to which to write the report.

Example 3-3 report port

```
============================================================
  ...
  Module:                 test
  Technology library:     tutorial 1.0
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
============================================================
         External Delays
                          Rise   Fall  Ext Delay
   Port     Dir  Clock   Delay  Delay   Object
-------------------------------------------------------------
 ck1       in     CLK2  500.0  500.0  in_del_2
                  CLK1  700.0  700.0  in_del_1
 e_out[6]  out          300.0  300.0  outrxt1
                  CLK2  300.0  300.0  ou_del_2
                  CLK1  200.0  200.0  ou_del_1
 ena       inout  CLK2  500.0  500.0  in_del_2
                  CLK1  700.0  700.0  in_del_1
 ena       inout          300.0  300.0  outrxt1
                  CLK2  300.0  300.0  ou_del_2
                  CLK1  200.0  200.0  ou_del_1

          External Loads
                  Pin  Wire Fanout
   Port     Dir   Cap   Cap  Load
-------------------------------------------------------------
 ck1       in     10.0  0.5    4.0
 e_out[6]  out     0.0  0.5    4.0
 ena       inout  20.0  0.7

         External Driver/Slew
                                          Timing
                  External   Slew  Slew  Case   Ideal
   Port     Dir    Driver    Rise  Fall  Value  Driver
-------------------------------------------------------------
 ck1       in    fflopd_ckn 100.0 100.0      1  false
 e_out[6]  out                               false
 ena       inout ivt1       100.0 100.0      false
 ena       inout                            false
```

Generating a Design Report

➤ Check a design for undriven and multi-driven ports and pins, unloaded sequential elements and ports, unresolved references, constant connected ports and pins and any assign statements using the following command:

```
check_design [-undriven] [-unloaded] [-multidriven] [-unresolved] [-constant]
[-assigns] [-all] [<design>] [> file]
```

For example, the following command reports all the information (assigns, constants, multi-driven, undriven, unloaded, and unresolved) on the design:

```
rc:\> check_design -all
```

By default, if you do not specify an option, then the `check_design` command reports a summary table with this information, as shown in Example 3-4.

Example 3-4 check_design Summary Table

```
Summary
----------------------------
         Name          Total
----------------------------
Unresolved References     0
Empty Modules             1
Unloaded Port(s)          0
Unloaded Pin(s)           1
Assigns                   2
Undriven Port(s)          2
Undriven Pin(s)           4
Multidriven Port(s)       0
Multidriven Pin(s)        0
Constant Port(s)          0
Constant Pin(s)           2
```

Example 3-5 shows the `check_design -all` command report.

Example 3-5 check_design -all Report

```
Unresolved References & Empty Modules
-------------------------------------
No unresolved references in design 'm1'
design 'm1' has the following empty module(s)
sub
Total number of empty modules in design 'm1' : 1
Unloaded Pin(s), Port(s)
------------------------
design 'm1' has the following unloaded sequential elements:
/designs/m1/instances_seq/bx_reg
Total number of unloaded sequential elements in design 'm1' : 1
No unloaded port in 'm1'
Assigns
-----------------------
Encountered an assign statement at subport '/designs/m1/instances_hier/sub/subports_out/
out[1]' in subdesign sub
Encountered an assign statement at subport '/designs/m1/instances_hier/sub/subports_out/
out[0]' in subdesign sub
Total number of assign statements in design 'm1' : 2
Undriven Port(s)/Pin(s)
-----------------------
The following combinational pin(s) in design 'm1' are undriven
/designs/m1/instances_comb/g22/pins_in/A
/designs/m1/instances_comb/g24/pins_in/A
Total number of combinational undriven pins in design 'm1' : 2
The following sequential pin(s) in design 'm1' are undriven
/designs/m1/instances_seq/bx_reg5/pins_in/D
/designs/m1/instances_seq/bx_reg7/pins_in/D
Total number of sequential undriven pins in design 'm1' : 2
The following port(s) in design 'm1' are undriven
/designs/m1/ports_out/co
Total number of undriven port(s) in design 'm1' : 2
 Multidriven Port(s)/Pin(s)
--------------------------
No multidriven combinational pin in 'm1'
No multidriven sequential pin in 'm1'
No multidriven ports in 'm1'
  Constant Pin(s)
  ---------------
No constant combinational pin(s) in design 'm1'
design 'm1' has the following constant input sequential pin(s)...
```

Supported .lib Timing Checks

Timing checks verify that the timing constraints are upheld. For more information, see the *Timing Library Format Reference*.

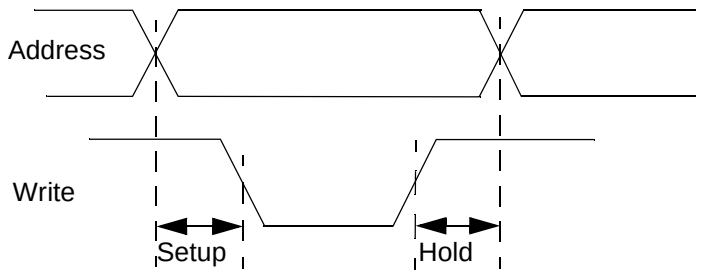
RTL Compiler supports the following Liberty timing checks:

- `nochange_high_high`
- `nochange_high_low`
- `nochange_low_high`
- `nochange_low_low`
- `recovery_rising`
- `recovery_falling`
- `setup_rising`
- `setup_falling`

Nochange

The *nochange* timing check shown in Figure 3-2 is a signal check relative to the width of a control pulse. A “setup” period is established before the start of the control pulse and a “hold” period after the pulse. The signal checked against the control signal must remain stable during the setup period which is the entire width of the pulse and the hold period. A *nochange* timing check is often used to model the timing of memory devices when address lines must remain stable during a write pulse with margins both before and after the pulse.

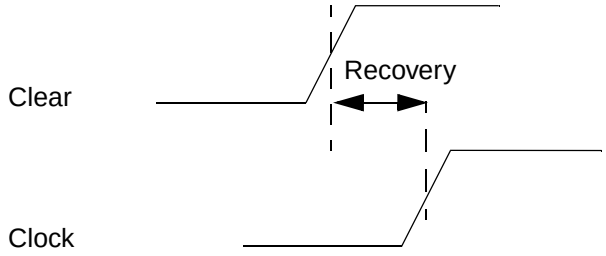
Figure 3-2 nochange Timing Check



Recovery

The *recovery* timing check shown in Figure 3-3 specifies a limit for the time allowed between the release of an asynchronous control signal from the active state and the next active clock edge. One example is a limit for the time between the release of the clear and the next active edge of the clock of a flip-flop. If the active clock edge occurs too soon after the release of the clear, then the state of the flip-flop becomes uncertain. The output can have the value set by the clear or the value clocked into the flip-flop from the data input.

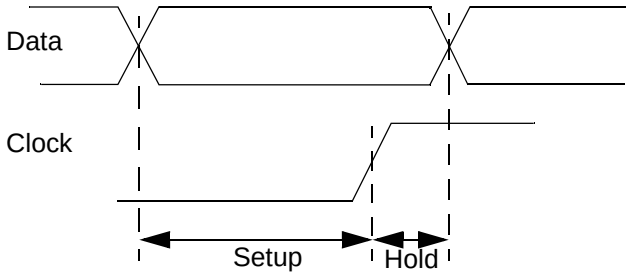
Figure 3-3 Recovery Timing Check



Setup

A *setup* check compares a data edge to the clock edge and specifies limit values for a setup time. In a flip-flop, the setup time is the time during which a data signal must remain stable before the clock edge. Any change to the data signal within this interval results in a timing violation. Figure 3-4 shows a positive setup time—one occurring before the active edge of the clock.

Figure 3-4 Positive Setup and Hold



4

Performing Timing Analysis

Overview of Timing Analysis

The purpose of timing analysis is to make sure the design meets the design goals after synthesis. Timing analysis identifies problem areas in the design and helps you determine how to solve these problems.

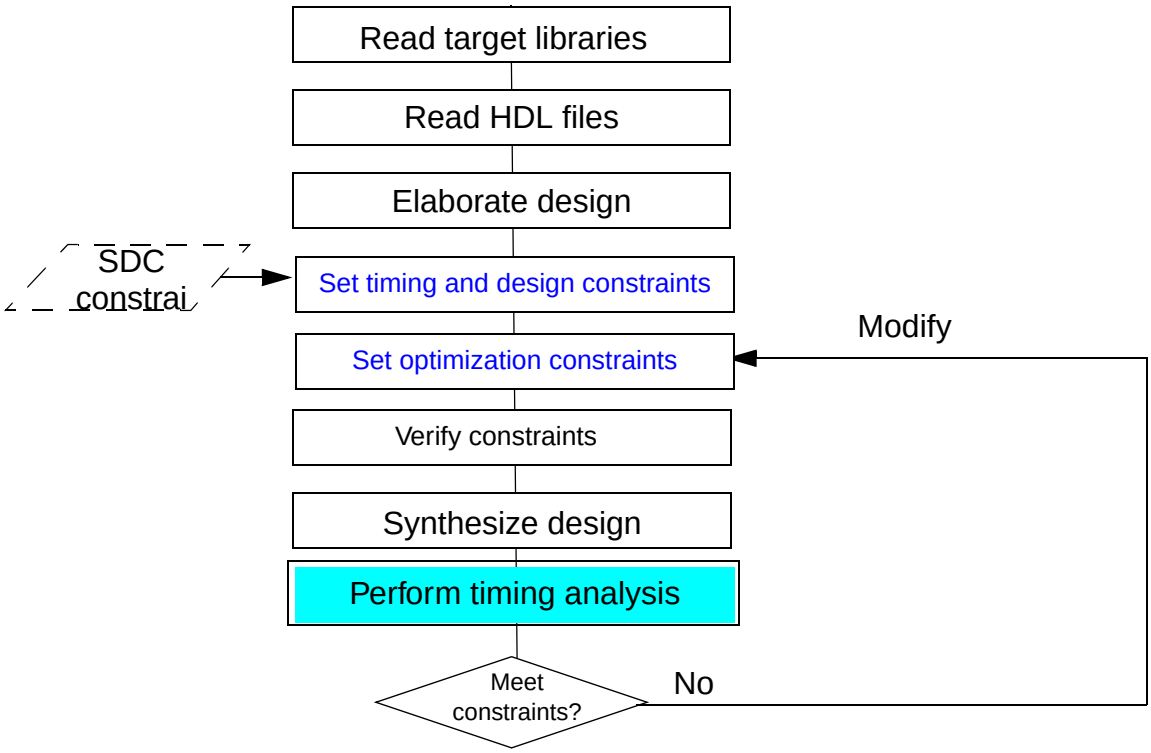
After synthesizing a design, generate post-synthesis reports to analyze the synthesis results, such as timing of the current design, area of each component in the current design, and gate selection. For information on verifying a new design or new constraints before synthesis, see Generating Pre-Synthesis Timing Reports on page 115.

Analyzing the timing compares the actual path delays with the required path delays of the design. Timing analysis computes gate and interconnect delay, traces critical paths, then uses the critical path values to create timing reports. This helps you identify constraint violations.

This chapter describes how to perform post-synthesis timing analysis in RTL Compiler and describes the factors that impact a timing report, such as the wire-load model, load, transition, and fanout.

See Chapter 6, “Troubleshooting Timing Analysis Issues,” for tips on reporting timing problems and how to analyze or “debug” a design using the timing report.

Figure 4-1 Timing Analysis Flow in Single Mode



Generating Post-Synthesis Timing Reports

This section describes how to generate timing reports after synthesis using the report timing command options.

See Generating a Default Timing Report on page 117 for information on how to generate a timing report before synthesis to report the worst critical path in the design.

See Checking the Constraints Using the report timing -lint Command on page 118 for more information on how to use the timing report to to verify any new constraints.

To generate reports in a multi-mode environment, see Chapter 5, “Performing Multi-Mode Timing Analysis,”

Using the report timing Command Options

➤ Use the `report timing` command options to generate different types of reports after synthesis on the timing of the current design:

```
report timing [-endpoints] [-lint] [-full_pin_names]
[-num_paths integer] [-slack_limit integer]
[-worst integer]
[-from {instance|external_delay|clock|port|pin}...]
[-through {instance|port|pin}...]...
[-to {instance|external_delay|clock|port|pin}...]
[-paths string] [-exceptions exception...]
[-cost_group cost_group] [> file]
```

The RTL, shown in Example 4-1, is used to explain the following `report timing` example reports. The following pre-synthesis constraints were applied:

- Clock constraints
- External delays
- Timing exceptions: `path_disable`, `path_delay`, `path_adjust`, `path_group`, and `multi_cycle`

Different attributes interact with the timing engine, such as:

- Specifying ideal drivers and nets
- Specifying specific borrow values for latches and clocks launched and captured at latches

Example 4-1 RTL Used in report timing Examples

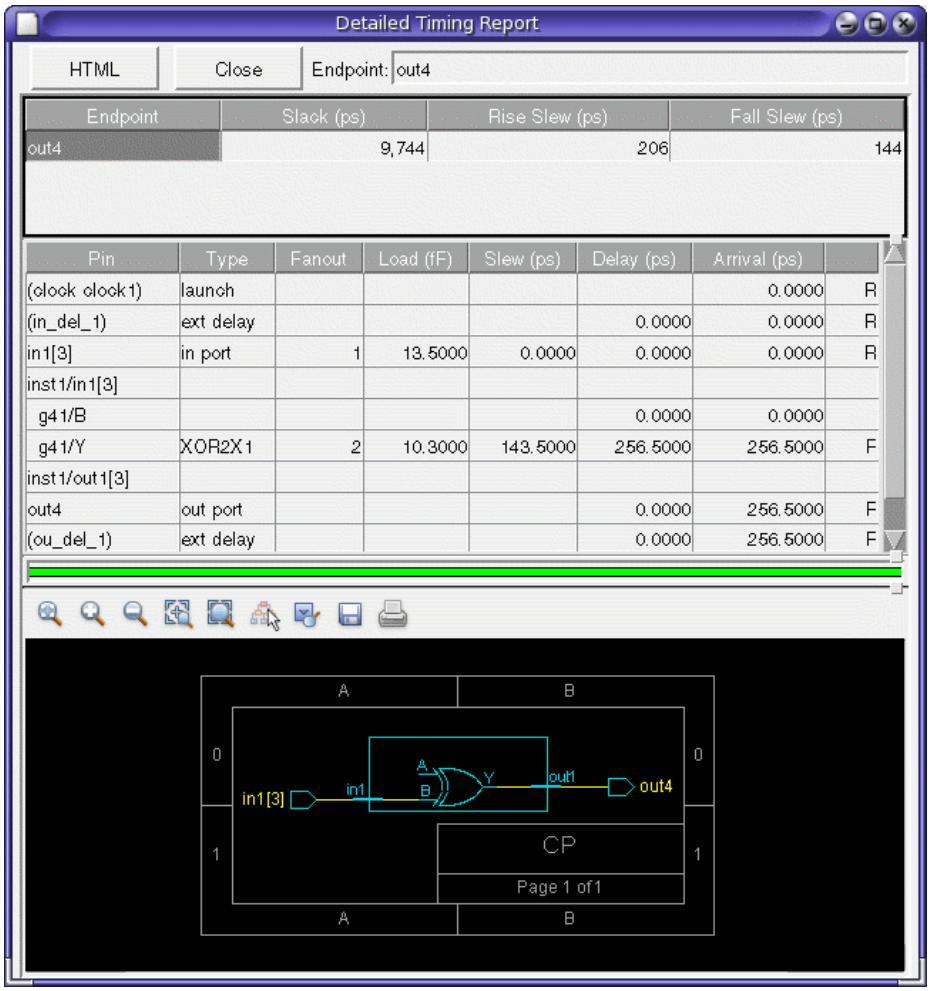
```
module timing_example(a, b, clk, y);
  input  [3:0] a, b;
  input  clk;
  output [3:0] y;

  reg [3:0] stage_a, stage_b, stage_y;

  always @(posedge clk) begin
      stage_a = a;
      stage_b = b;
  end

  always @(posedge clk) begin
      stage_y = stage_a + stage_b;
  end

  assign y = stage_y;
endmodule
```

Using the Timing Reports from the GUI

➤ Use the `gui_report_timing` command from the prompt to see the timing report from the GUI. You can also see the post-synthesis schematic.

```
rc:/> gui_report_timing
```

You will see a new window like the following:



Reporting the Timing Value Between Two Points in a Design

➤ Use the `from` and `to` options to report the timing value between two points in the design. The timing points in the report are called out with the `<<<` indicator.

The following command generates a post-synthesis timing report, as shown in Example 4-3.

```
rc:/> report timing -from stage_a_reg_0 -to stage_y_reg_3
```

Example 4-2 on page 137 shows how this report looks before synthesis. The report shows the unmapped gates before they are replaced with mapped library elements. The report also shows the header, launch and capture clocks, the timing slack, and the timing startpoint and endpoint.

Example 4-2 report timing Showing Unmapped Gates

```
================================================================================
  Generated by:           RTL Compiler (RC) Version
  Generated on:           Date
  Module:                 add
  Technology library:     tutorial 1.0
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
================================================================================
    Pin                      Type            Fanout  Load  Slew   Delay   Arrival
                                                     (fF)  (ps)    (ps)    (ps)
--------------------------------------------------------------------------------
(clock clk)                  launch                                          0 R
stage_a_reg_0/clk     <<<                                   0               0 R
stage_a_reg_0/q  (u)         unmapped_d_flop     3   70.8     0    +164     164 R
add_14_25/A[0]
  g2/in_0                                                            +0     164
  g2/z            (u)        unmapped_nand2      1   23.6     0    +136     300 F
  g5/in_0                                                            +0     300
  g5/z            (u)        unmapped_nand3      3   70.8     0    +196     496 R
  g25/in_0                                                           +0     496
  g25/z           (u)        unmapped_nand2      1   23.6     0    +136     632 F
  g26/in_0                                                           +0     632
  g26/z           (u)        unmapped_nand2      1   23.6     0    +136     768 R
  g35/in_0                                                           +0     768
  g35/z           (u)        unmapped_xnor2      1   23.6     0    +136     905 R
add_14_25/Z[3]
stage_y_reg_3/d       <<<    unmapped_d_flop                          +0     905
stage_y_reg_3/clk            setup                                 0    +100    1005 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                  capture                                         500 R
--------------------------------------------------------------------------------
Timing slack :    -505ps (TIMING VIOLATION)
Start-point  : stage_a_reg_0/clk
End-point    : stage_y_reg_3/d
u: Net has unmapped pin(s).
```

Both reports show the header, launch and capture clocks, the complete path and slack, and the timing startpoint and endpoint. However, the difference between generating the report after synthesis is that the unmapped gates are replaced with mapped library elements.

Example 4-3 Post-Synthesis Timing Report

```
====================================================================
  Generated by:          RTL Compiler (RC) Version
  Generated on:          Date
  Module:                add
  Technology library:    tutorial 1.0
  Operating conditions:  typical_case (balanced_tree)
  Wireload mode:         enclosed
====================================================================
    Pin                    Type    Fanout  Load  Slew   Delay   Arrival
                                           (fF)  (ps)    (ps)    (ps)
--------------------------------------------------------------------
(clock clk)                launch                                  0 R
stage_a_reg_0/CK     <<<                          0               0 R
stage_a_reg_0/Q            fflopd      2   25.7    17    +148     148 R
g82/A                                                    +0     148
g82/Y                      nand2       3   45.9    52    +130     278 F
g81/A                                                    +0     278
g81/Y                      inv1        1   10.5    15     +84     363 R
g66/B                                                    +0     363
g66/Y                      nand2       1   10.5    45    +121     484 F
g62/B                                                    +0     484
g62/Y                      nand2       1   15.5    46    +120     604 R
g59/A                                                    +0     604
g59/Y                      xor2        1   20.5    58    +146     751 R
stage_y_reg_3/D      <<<   fflopd                           +0     751
stage_y_reg_3/CK           setup                    0    +100     851 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                capture                                 500 R
--------------------------------------------------------------------
Timing slack :    -351ps (TIMING VIOLATION)
Start-point  : stage_a_reg_0/CK
End-point    : stage_y_reg_3/D
```

This report includes the following information:

- Prints out header information, including the wire-load model, operating conditions, and the library used
- Describes the launching and capturing clocks

- Shows hierarchical instances and paths inside: `add_14_25/..`
- Reports the worst path amongst all the paths `-from stage_a_reg* -to stage_y_reg*`
- Includes annotations, such as the following:
  - `(u)` shows unmapped pins
  - `<<<` shows path points set using the `-from`, `-to`, `-through` options or the `specify_paths` command
  - `(i)` shows an ideal driver

    See Handling Ideal Nets on page 91 for more information.
  - `(*)` shows a zero slack borrow limit
  - `(@)` shows an annotated cap
  - `(b)` shows broken timing paths
- Shows the number of fanouts for each driver. This is the structural fanout instead of the wireload fanout, which is the number of leaf load pins on a net.
- Shows the total load (or capacitance) at each segment of the timing path in femto Farads
- Shows the slew, delay (interconnect delay at load pins, or gate or arc delay for a gate), and cumulative arrivals
- Shows the final slack and corresponding start- and end-points

Reporting Timing Exceptions

➤ Use the `-from` and `-to` options with the `report timing` command to report timing exceptions, such as `path_disable`, `path_delay`, `path_adjust`, `multi_cycle`, `path_group`, and `cost_group`.

- `path_disable`: Disables the timing analysis for specified paths. A path that satisfies this exception does not have a timing constraint set on it, hence, the path is not considered timing critical. The following commands generate a timing report with `path_disable`, as shown in Example 4-4:

```
rc:/> path_disable -from {stage_a_reg* stage_b_reg*} -to stage_y_reg_*
/designs/add/timing/exceptions/path_disables/dis_1
rc:/> report timing -from stage_a_reg_0 -to stage_y_reg_3
```

Example 4-4 Disabling Timing Analysis for Specified Paths

```
Tracing clock networks.
Levelizing the circuit.
Applying wireload models.
Computing net loads.
Computing delays.
Computing arrivals and requireds.
============================================================================
  Generated by:           RTL Compiler (RC) version
  Generated on:           Date
  Module:                 add
  Technology library:     tutorial 1.0
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
============================================================================
    Pin                    Type    Fanout  Load  Slew   Delay   Arrival
                                           (fF)  (ps)    (ps)    (ps)
----------------------------------------------------------------------------
(clock clk)                launch                                    0 R
stage_a_reg_0/CK     <<<                              0              0 R
stage_a_reg_0/Q            fflopd      2   25.7    17    +148     148 R
g82/A                                                    +0     148
g82/Y                      nand2       3   45.9    52    +130     278 F
g81/A                                                    +0     278
g81/Y                      inv1        1   10.5    15     +84     363 R
g66/B                                                    +0     363
g66/Y                      nand2       1   10.5    45    +121     484 F
g62/B                                                    +0     484
g62/Y                      nand2       1   15.5    46    +120     604 R
g59/A                                                    +0     604
g59/Y                      xor2        1   20.5    58    +146     751 R
stage_y_reg_3/D      <<<   fflopd                           +0     751
stage_y_reg_3/CK           setup                     0    +100     851 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                capture                                   R
----------------------------------------------------------------------------
Exception    : 'path_disables/dis_1'
Timing slack :  UNCONSTRAINED
Start-point  : stage_a_reg_0/CK
End-point    : stage_y_reg_3/D
```

- `path_delay`: Overrides the default constraints and clocks for specified paths. The following commands generate a timing report with `path_delay` constraints, as shown in Example 4-5:

```
rc:/ path_delay -from {stage_a_reg* stage_b_reg*} -to stage_y_reg_* -delay 100
/designs/add/timing/exceptions/path_delays/del_1
rc:/ report timing -from stage_a_reg_0 -to stage_y_reg_3
```

Example 4-5 Timing Report with path_delay Constraints

```
 Tracing clock networks.
        Levelizing the circuit.
        Applying wireload models.
        Computing net loads.
        Computing delays.
        Computing arrivals and requireds.
================================================================================
  Generated by:           RTL Compiler (RC) version
  Generated on:           Date
  Module:                 add
  Technology library:     tutorial 1.0
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
================================================================================
    Pin                    Type       Fanout  Load  Slew   Delay   Arrival
                                             (fF)  (ps)    (ps)    (ps)
--------------------------------------------------------------------------------
stage_a_reg_0/CK     <<<                                    0              0 R
stage_a_reg_0/Q            fflopd         2   25.7    17    +148     148 R
g82/A                                                      +0     148
g82/Y                      nand2          3   45.9    52    +130     278 F
g81/A                                                      +0     278
g81/Y                      inv1           1   10.5    15     +84     363 R
g66/B                                                      +0     363
g66/Y                      nand2          1   10.5    45    +121     484 F
g62/B                                                      +0     484
g62/Y                      nand2          1   15.5    46    +120     604 R
g59/A                                                      +0     604
g59/Y                      xor2           1   20.5    58    +146     751 R
stage_y_reg_3/D      <<<   fflopd                             +0     751
stage_y_reg_3/CK           setup                     0     +100     851 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                           path_delay                                 100
--------------------------------------------------------------------------------
Exception    : 'path_delays/del_1'     100ps
Timing slack :    -751ps (TIMING VIOLATION)
Start-point  : stage_a_reg_0/CK
End-point    : stage_y_reg_3/D
```

- `path_adjust`: Adjusts the path constraints by a specified amount. A positive adjustment loosens the constraint while a negative adjustment tightens it. Path adjusts are cumulative. The following report timing command reports `path_adjust` constraints, as shown in Example 4-6.

```
rc:/> path_adjust -from {stage_a_reg* stage_b_reg*} -to stage_y_reg_* -delay
100
rc:/> path_adjust -from {stage_a_reg*} -to stage_y_reg_* -delay 5
rc:/> report timing -from stage_a_reg_0 -to stage_y_reg_3
```

Example 4-6 Path Adjusts Timing Report

```
================================================================================
  Generated by:           RTL Compiler (RC) version
  Generated on:           Date
  Module:                 add
  Technology library:     tutorial 1.0
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
================================================================================
    Pin                    Type        Fanout  Load  Slew   Delay   Arrival
                                               (fF)  (ps)    (ps)    (ps)
--------------------------------------------------------------------------------
(clock clk)                launch                                        0 R
stage_a_reg_0/CK     <<<                                 0               0 R
stage_a_reg_0/Q            fflopd          2   25.7    17    +148     148 R
g82/A                                                        +0     148
g82/Y                      nand2           3   45.9    52    +130     278 F
g81/A                                                        +0     278
g81/Y                      inv1            1   10.5    15     +84     363 R
g66/B                                                        +0     363
g66/Y                      nand2           1   10.5    45    +121     484 F
g62/B                                                        +0     484
g62/Y                      nand2           1   15.5    46    +120     604 R
g59/A                                                        +0     604
g59/Y                      xor2            1   20.5    58    +146     751 R
stage_y_reg_3/D      <<<   fflopd                              +0     751
stage_y_reg_3/CK           setup                      0      +100     851 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                capture                                     500 R
                           adjustments                         +105    605
--------------------------------------------------------------------------------
Exception    : 'path_adjusts/adj_2' path adjust       5ps
Exception    : 'path_adjusts/adj_1' path adjust     100ps
Timing slack :    -246ps (TIMING VIOLATION)
Start-point  : stage_a_reg_0/CK
End-point    : stage_y_reg_3/D
```

- `multi_cycle`: Overrides the default clock cycle, or the closing and opening edges of the clock. Use the `multi_cycle -help` command to see detailed examples for each of the shifting capture and launch edges for paths.

- `path_group` and `cost_group`: These timing exceptions work together. For example, if you use the following `cost_group` command to define a “group” with an optimization weight of `5` and name the group `C2O`:

  ```
  rc:/> cost_group -name C2O -weight 5
  ```

  Then you can attach different paths to this group. For example, the following commands associate each path that satisfies this exception as a `C2O` group path with a weight of `5`. All paths in a group are optimized together. The group weight is used to optimize a particular set of paths.

  ```
  rc:/> path_group -from [find . -instance *reg*] -to /designs/timing_example/
  ports_out/* -group C2O
  ```

Reporting Path Exceptions

➤ Use the `report timing -lint` command to also report path exceptions. If an exception is not satisfied by any path in the design, then it is reported.

For example, from the RTL shown in Example 4-1, there cannot be a path from `stage_y_reg*` to `stage_a_reg_*` (or to `stage_b_reg_*`). Any exception defined for such a path cannot be satisfied by reporting it as lint (warning), as shown in Example 4-7.

```
rc:/> multi_cycle -capture_shift 2 -from stage_y_reg* -to {stage_a_reg*
stage_b_reg*} -name exception_unsatis
/designs/timing_example/timing/exceptions/multi_cycles/exception_unsatis
rc:/> report timing -lint
```

Example 4-7 Reporting Path Exceptions

```
 Tracing clock networks.
        Levelizing the circuit.
        Applying wireload models.
        Computing net loads.
        Computing delays.
        Computing arrivals and requireds.
============================================================
  Generated by:           RTL Compiler (RC) Version
  Generated on:           Date
  Module:                 timing_example
  Technology library:     tutorial 1.0
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
============================================================
No paths in the design meet the path specifications for the following timing
exceptions. To see the path specifications, use the get_attribute paths attribute
on the exception:
/designs/timing_example/timing/exceptions/multi_cycles/exception_unsati
```

➤ Use the `-num_paths` option to specify the maximum number of paths to report.

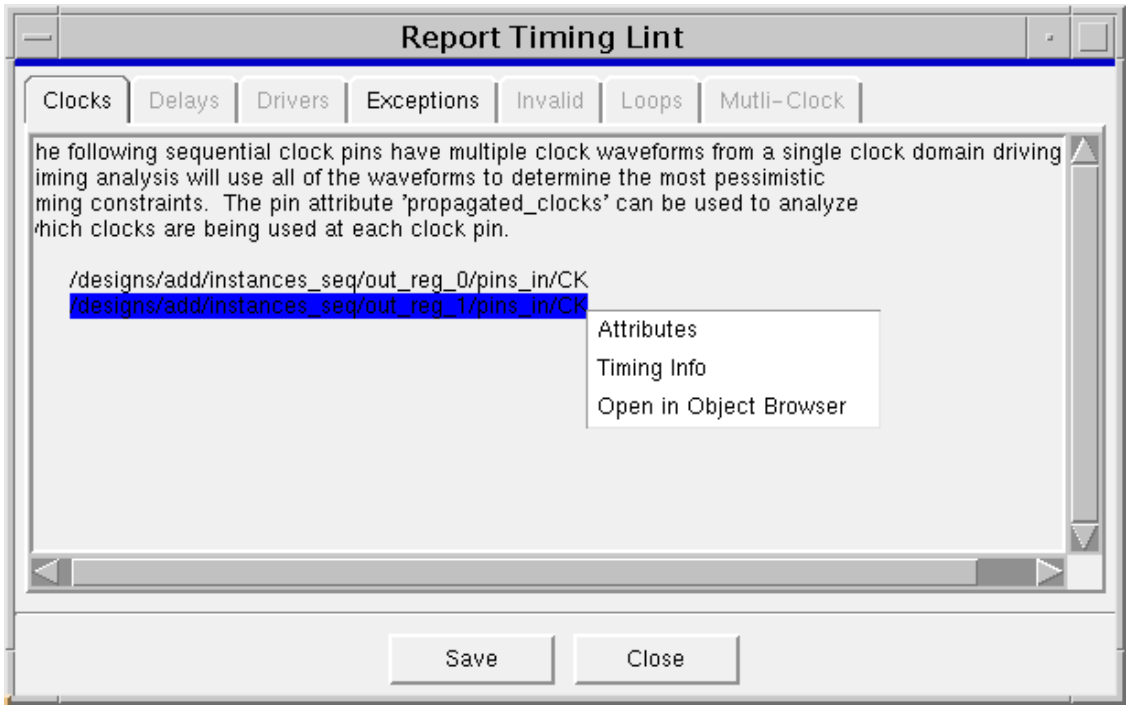
Example 4-8 shows a timing report run with the `-num_paths` option.

Example 4-8 Reporting the Maximum Number of Paths

```
rc:/designs/top> report timing -num_paths 4
============================================================================
  Generated by:           Cadence RTL Compiler (RC) Version
============================================================================
path   1:
    Pin               Type      Fanout  Load  Slew   Delay   Arrival
                                        (fF)  (ps)    (ps)    (ps)
----------------------------------------------------------------------------
----------------------------------------------------------------------------
Timing slack :     543ps
Start-point  : accum[1]
End-point    : aluout_reg_7/D
path   2:
    Pin               Type      Fanout  Load  Slew   Delay   Arrival
                                        (fF)  (ps)    (ps)    (ps)
...
```

➤ You can also use the GUI to obtain a `report timing -lint` report. Go to the menu bar and *Report -> Timing -> Lint Report*.

Example 4-9 GUI: Report timing -lint



Right-click on an object name to view the context-sensitive menu. The *Exceptions* tab also includes a report timing command

Reporting EndPoint Histograms

➤ Use the GUI to visualize endpoint slack histogram. Go to the menu bar and *Report -> Timing -> Endpoint Histogram*.

Figure 4-2 GUI: Endpoint Slack Histogram

Generating an Area Report

The area report lists the total design area, as well as a breakdown of area per level of hierarchy. The area numbers are calculated by counting the number of cells instantiated in the design and multiplying by the cell area specified in the respective technology library.

➤ Use the report area [*design*]... [> *file*] command to report the number of cells mapped against the hierarchical blocks in the current design, the wire-load model for each of the blocks, and the combined cell area in each of the blocks and the top level design.

Example 4-10 shows an area report for the current design.

Example 4-10 Area Report

```
rc:/ report area
=============================================================
  Generated by:           RTL Compiler (RC) version
  Generated on:           Date
  Module:                 filter16opt
  Technology library:     osu05_stdcells
  Operating conditions:   typical (balanced_tree)
  Wireload mode:          enclosed
=============================================================
Instance                  Cells   Cell Area   Net Area     Wireload
-------------------------------------------------------------------
filter16opt                4141     1723122          0       <none>
  csa_tree_add_396_39      1081      475092          0       <none>
  const_mul_284_42          306       71991          0       <none>
  const_mul_287_42          283       70839          0       <none>
  final_adder_add_396_39    245       55935          0       <none>
```

Generating Cell Instance Reports

By default, the instance report provides timing related information on

- Instances
- Library cell instances
- Pins of the instance and all the delay arcs between those pins
- Slew-in, load, slew-out, delay, and slack

➤ Generate timing information for a cell instance in the design using the `report instance` command.

Example 4-11 shows an instance report that includes power information.

Example 4-11 report instance

```
  ...
   Module:                dpldalgn
   Technology library:    tutorial 1.0
   Operating conditions:  typical_case (balanced_tree)
   Wireload mode:         enclosed
=============================================================
          Instance Timing Info
          --------------------
  Instance n0000D3666 of libcell nor2

  Arc  Arc  Slew        Slew
 From  To    In   Load  Out   Delay   Slack
-----------------------------------------

  A r  Y f  46.2  20.7  57.2  136.0  -1022.4
  A f  Y r  46.2  20.7  57.2  136.0  -1022.4
  B r  Y f  16.5  20.7  57.2  134.0   -900.2
  B f  Y r  16.5  20.7  57.2  134.0   -900.2

          Instance Power Info
          -------------------
  Instance n0000D3666 of Libcell nor2

   Leakage    Internal      Net
  Power(nW)   Power(nW)  Power(nW)
---------------------------------

        0.0         0.0     1293.8

              Computed          Computed           Net
Pin  Net    Probability    Toggle Rate(/ns)    Power(nW)
-----------------------------------------------------------
Y   n_176            0.2              0.0050       1293.8
A   n_138            0.7              0.0100       2587.5
B   n_26             0.5              0.0200       5175.0
```

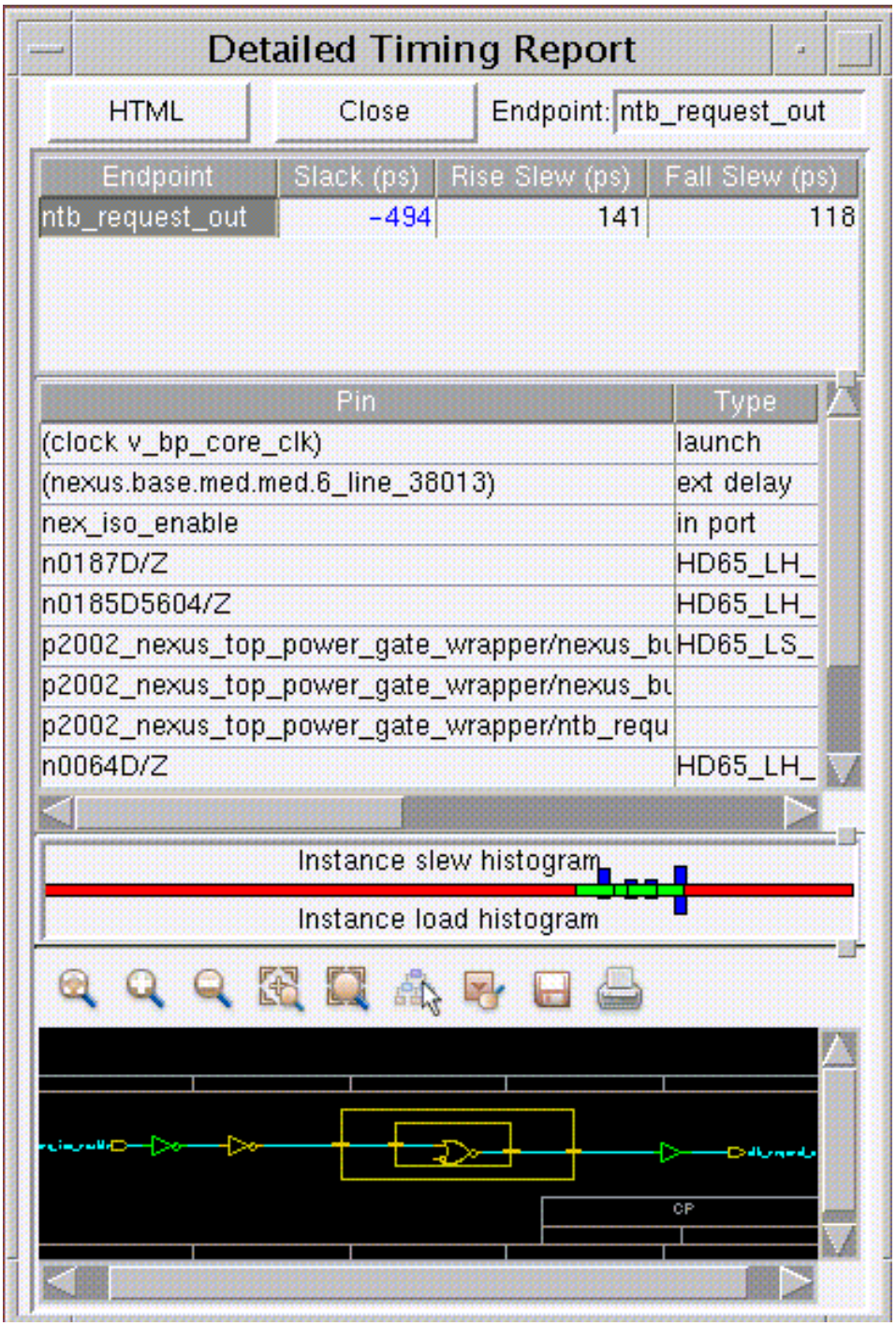
- *Slew-In*: Reports the slew for all input pins.

- *Slew-Out*: Reports the slew for all output pins.
- *Load*: Reports the fanout and interconnect capacitance
- *Delay*: Reports the cell delay, which is determined by the cell's intrinsic delay (internal delay), the load that it is driving, and the input transition (skew).
- *Slack*: Reports the amount of time by which a violation is avoided. A negative slack indicates a timing violation.
- *Arc From - Arc To*: Reports the delay arcs between pins.

In the GUI, you can simultaneously visualize instance slew with propagation delays with the `gui_tb_set_top_pane` command.

```
rc:/> gui_tb_set_top_pane -type slew
```

You will get something like the following:


Detailed Timing Report
HTML
Close
Endpoint: ntb_request_out
Endpoint
Slack (ps)
Rise Slew (ps)
Fall Slew (ps)
ntb_request_out
-494
141
118
Pin
Type
(clock v_bp_core_clk)
launch
(nexus.base.med.med.6_line_38013)
ext delay
nex_iso_enable
in port
n0187D/Z
HD65_LH_
n0185D5604/Z
HD65_LH_
p2002_nexus_top_power_gate_wrapper/nexus_bu
HD65_LS_
p2002_nexus_top_power_gate_wrapper/nexus_bu
p2002_nexus_top_power_gate_wrapper/ntb_requ
n0064D/Z
HD65_LH_
Instance slew histogram
Instance load histogram
CP

Reporting Design Rule Violations (DRVs)

➤ Use the report_design_rules command to report any DRVs that are present in the specified design objects.

Example 4-12 shows a DRV report for the maximum transition violations.

Example 4-12 DRV Report for max_transition

```
rc:/ report design_rules design_name
============================================================
  Generated by:           RTL Compiler (RC) version
  Generated on:           date
  Module:                 design_name
  Technology library:     tutorial 1.0
  Operating conditions:   MAX (balanced_tree)
  Wireload mode:          enclosed
============================================================
Max_transition design rule (violation total = 273)
Pin                         Slew (ps)           Max     Violation
------------------------------------------------------------------
a[16] (Primary Input)             436           400            36
  --> Other violations on net             690
a[17] (Primary Input)             436           400            36
  --> Other violations on net             686
a[0] (Primary Input)              431           400            31
  --> Other violations on net             589
a[2] (Primary Input)              429           400            29
  --> Other violations on net             553
a[14] (Primary Input)             429           400            29
  --> Other violations on net             553
a[1] (Primary Input)              429           400            29
  --> Other violations on net             553
a[18] (Primary Input)             428           400            28
  --> Other violations on net             540
a[13] (Primary Input)             428           400            28
  --> Other violations on net             540
a[7] (Primary Input)              428           400            28
  --> Other violations on net             540
```

Example 4-13 shows a DRV report for the maximum capacitance design rule violations.

Example 4-13 DRV Report for max_capacitance

```
rc:/ report design_rules design_name
============================================================
  Generated by:          RTL Compiler (RC) version
  Generated on:          date
  Module:                design_name
  Technology library:    tutorial 1.0
  Operating conditions:  MAX (balanced_tree)
  Wireload mode:         enclosed
============================================================
Max_capacitance design rule (violation total = 220.9)
Pin                          Load (ff)           Max     Violation
------------------------------------------------------------------
a[16] (Primary Input)           179.5         165.0          14.5
a[17] (Primary Input)           179.4         165.0          14.4
a[0]  (Primary Input)           177.3         165.0          12.3
a[2]  (Primary Input)           176.5         165.0          11.5
a[14] (Primary Input)           176.5         165.0          11.5
a[1]  (Primary Input)           176.5         165.0          11.5
a[10] (Primary Input)           176.2         165.0          11.2
a[18] (Primary Input)           176.2         165.0          11.2
a[7]  (Primary Input)           176.2         165.0          11.2
a[6]  (Primary Input)           176.2         165.0          11.2
a[5]  (Primary Input)           176.2         165.0          11.2
a[4]  (Primary Input)           176.2         165.0          11.2
a[3]  (Primary Input)           176.2         165.0          11.2
a[13] (Primary Input)           176.2         165.0          11.2
a[12] (Primary Input)           176.2         165.0          11.2
a[11] (Primary Input)           176.2         165.0          11.2
a[15] (Primary Input)           176.2         165.0          11.2
a[9]  (Primary Input)           176.0         165.0          11.0
a[8]  (Primary Input)           176.0         165.0          11.0
```

Example 4-14 shows a DRV report for the maximum fanout design rule violations.

Example 4-14 DRV Report for max_fanout

```
rc:/ report design_rules design_name
============================================================
  Generated by:          RTL Compiler (RC) version
  Generated on:          date
  Module:                design_name
  Technology library:    tutorial 1.0
  Operating conditions:  MAX (balanced_tree)
  Wireload mode:         enclosed
============================================================
Max_fanout design rule (violation total = 228.000)
Pin                            Fanout           Max     Violation
------------------------------------------------------------------
a[9] (Primary Input)           20.000         8.000        12.000
a[17](Primary Input)           20.000         8.000        12.000
a[16](Primary Input)           20.000         8.000        12.000
a[15](Primary Input)           20.000         8.000        12.000
a[14](Primary Input)           20.000         8.000        12.000
a[13](Primary Input)           20.000         8.000        12.000
a[12](Primary Input)           20.000         8.000        12.000
a[11](Primary Input)           20.000         8.000        12.000
a[10](Primary Input)           20.000         8.000        12.000
a[18](Primary Input)           20.000         8.000        12.000
a[8] (Primary Input)           20.000         8.000        12.000
a[7] (Primary Input)           20.000         8.000        12.000
a[6] (Primary Input)           20.000         8.000        12.000
a[5] (Primary Input)           20.000         8.000        12.000
a[4] (Primary Input)           20.000         8.000        12.000
a[3] (Primary Input)           20.000         8.000        12.000
a[2] (Primary Input)           20.000         8.000        12.000
a[1] (Primary Input)           20.000         8.000        12.000
a[0] (Primary Input)           20.000         8.000        12.000
```

Generating a Gates Report

Use the report gates command to list all the library cells used in the design, the number of instances, area, and the library name, as shown in Example 4-15. The report gates command does not accept a sub design, such as report gate /design/...

➤ To report the gate count of a sub-block, use the following commands:

```
rc:/> get_attribute cell_area [find / -subdesign <subd name>]
rc:/> get_attribute cell_count [find / -subdesign <subd name>
rc:/> report gates
```

Example 4-15 report gates

```
  Gate  Instances     Area          Library
-------------------------------------------------
AND2X1         10     2880.000    osu05_stdcells
AND2X2        137    39456.000    osu05_stdcells
AOI21X1        60    17280.000    osu05_stdcells
AOI22X1        28    10080.000    osu05_stdcells
BUFX2          74    15984.000    osu05_stdcells
BUFX4           4     1152.000    osu05_stdcells
DFFSR         438   693792.000    osu05_stdcells
FAX1          261   281880.000    osu05_stdcells
HAX1           44    31680.000    osu05_stdcells
INVX1         561    80784.000    osu05_stdcells
INVX2         511    73584.000    osu05_stdcells
INVX4           8     1728.000    osu05_stdcells
INVX8           1      360.000    osu05_stdcells
MUX2X1         93    40176.000    osu05_stdcells
NAND2X1       935   201960.000    osu05_stdcells
NAND3X1        35    11340.000    osu05_stdcells
NOR2X1        221    47736.000    osu05_stdcells
NOR3X1          1      576.000    osu05_stdcells
OAI21X1       618   127926.000    osu05_stdcells
OAI22X1        22     7920.000    osu05_stdcells
OR2X1          11     3168.000    osu05_stdcells
OR2X2          12     3456.000    osu05_stdcells
XNOR2X1        26    13104.000    osu05_stdcells
XOR2X1         30    15120.000    osu05_stdcells
-------------------------------------------------
total        4141  1723122.000
```

Generating a Net Report

➤ Use the report nets command to get information for the top-level nets in the design.

You can specify pin names, nets, instances, maximum and minimum fanout threshold values, nets, and instances.

Control the data printed out using the -minfanout and -maxfanout options for nets that have fanout between these values.

The following command takes pin names and reports the nets associated with them, as shown in Example 4-16:

```
rc:/> report net -pin g22/A bx_reg2/D
```

Example 4-16 report net

```
============================================================
  Generated by:           RTL Compiler (RC) Version
  Generated on:           Date
  Module:                 m1
  Technology library:     slow 1.5
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          segmented
============================================================
     Total   Slew Slew
Net Cap(fF)  Rise Fall Driver(s)   Load(s)
------------------------------------------
n_0    20.3   0.0  0.0  g24/Y    g23/A
                                 g22/A
                                 bx_reg3/CK
ai     12.0   0.0  0.0  ai       g25/A
                                 bx_reg2/D
```

Generating a Summary Report

➤ Use report summary to report the area used by the design, cells mapped for the blocks in the specified design, the wireload model, and the timing slack of the critical path. You can also report if any design rule is violated and the worst violator information.

The report summary command generates a report of the area used by the design, the worst timing endpoint in the design, and the design rule violations summary, as shown in Example 4-17.

Example 4-17 report summary

```
============================================================
  Generated by:           RTL Compiler (RC) version
  Generated on:           date
  Module:                 alu
  Technology library:     tutorial 1.0
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
============================================================
        Timing
        ------
 Slack       Endpoint
-------------------------------------
 -1082ps out1_tmp_reg[9]/D
        Area
        ----
Instance       Cells   Cell Area   Net Area     Wireload
------------------------------------------------------------
gen_test         326         525          0    AL_MEDIUM (S)
 (S) = wireload was automatically selected
        Design Rule Check
        -----------------
Max_transition design rule: no violations.
Max_capacitance design rule (violation total = 19402.5)
Worst violator:
Pin                         Load (ff)           Max     Violation
------------------------------------------------------------------
in0[5] (Primary Input)           96.9           5.0          91.9
Max_fanout design rule (violation total = 16.000)
Worst violator:
Pin                            Fanout           Max     Violation
------------------------------------------------------------------
in0[5] (Primary Input)          8.000         4.000         4.000
```

Generating a QOR Report

➤ Use the report qor command to report the critical path slack, total negative slack (TNS), number of gates on the critical path, and number of violating paths for each cost group. Also report the instance count, number of clock gating elements, total area, cell area, leakage power, runtime, and host name information, as shown in Example 4-18.

Example 4-18 report qor

```
================================================================
  Generated by:           RTL Compiler (RC) version
  Generated on:           Date
  Module:                 filter16opt
  Technology library:     osu05_stdcells
  Operating conditions:   typical (balanced_tree)
  Wireload mode:          enclosed
================================================================
Timing
------
  Cost    Critical       Violating
 Group   Path Slack TNS    Paths
-----------------------------------
default    No paths   0
I2C          6587.6   0          0
C2O          6545.3   0          0
C2C             0.5   0          0

Instance Count
--------------
Leaf Instance Count            4141
Sequential Instance Count       438
Combinational Instance Count   3703
Hierarchical Instance Count       4
Area & Power
------------
Total Area                          1723122.0
Cell Area                           1723122.0
Leakage Power                       541.097 nW
Switching Power                     400202286.966 nW
Runtime                             67 seconds
Hostname                            rcae004.cadence.com
```

Generating a Power Report

➤ Use the report power command, as shown in Example 4-19, to report the power consumed. The information reported depends on your current position in the design hierarchy and on the specified nets or instances.

Example 4-19 report power

```
=============================================================
  Generated by:           RTL Compiler (RC) v05.20-s015_1
  Generated on:           Feb 23 2006  04:25:42 PM
  Technology library:     osu05_stdcells
  Operating conditions:   typical (balanced_tree)
  Wireload mode:          enclosed
=============================================================
                              Leakage     Internal       Net        Switching
       Instance        Cells Power(nW)   Power(nW)     Power(nW)     Power(nW)
-------------------------------------------------------------------------------
filter16opt            4141   541.097 249800383.841 150401903.125 400202286.966
csa_tree_add_396_39    1081   150.959  58769356.269  61963412.500 120732768.769
const_mul_287_42        283    20.499   4761835.405   5740381.250  10502216.655
const_mul_284_42        306    19.943   5458585.064   6765406.250  12223991.315
final_adder_add_396_39  245    16.272  16235385.554  16788100.000  33023485.554
```

Analyzing Latch-Based Designs

For information on using the `latch_borrow` and `latch_max_borrow` attributes, see Specifying Latch Time Borrow Values on page 89.

Latch Time Borrowing

Time borrowing, also known as cycle stealing, takes advantage of the latch transparency to borrow time from the next stage to meet timing constraints.

Static timing analysis of latch-based designs is a two step process.

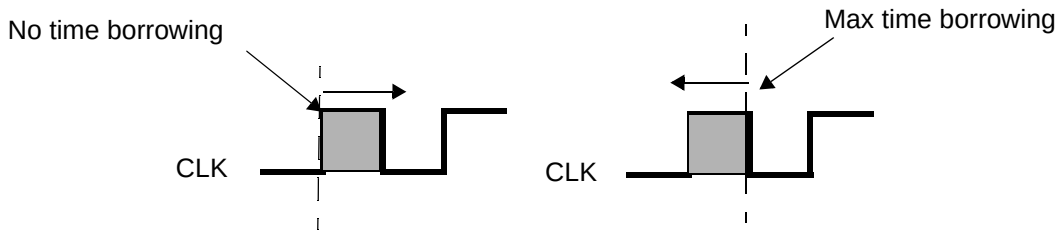
1. Perform timing analysis and assume that the d->q arcs of latches are not active. The q output of every latch switches a short time after the latch opens and is then stable for the rest of the clock cycle.
2. Enable the D to Q arcs and if the arrival times at all of the latch D pins are early enough that they are available before the latch opens again, then the circuit will meet its setup requirements and there is no timing violation.

   However, if the arrival time at one of the D pins is late enough to cause a transition at Q that is later than previously assumed for Q, then time borrowing must be performed and new arrival times must be computed.

Complications occur when the arrival time at one of the latch d pins is late enough that it causes a transition at q that is later than the latest arrival time that was previously assumed for q. Whenever the arrival time at q is caused by a late arrival at d, instead of being caused by the enable ->q event, then time borrowing occurs.

Time is always borrowed from one stage to its previous stage, as shown in Figure 4-3. No time borrowing for a latch means that the logic feeding the D input must be stable before the latch opens. Maximum time borrowing means that the logic feeding the D input has time to become stable until the latch closes.

Figure 4-3 Polarity of Time Borrowing

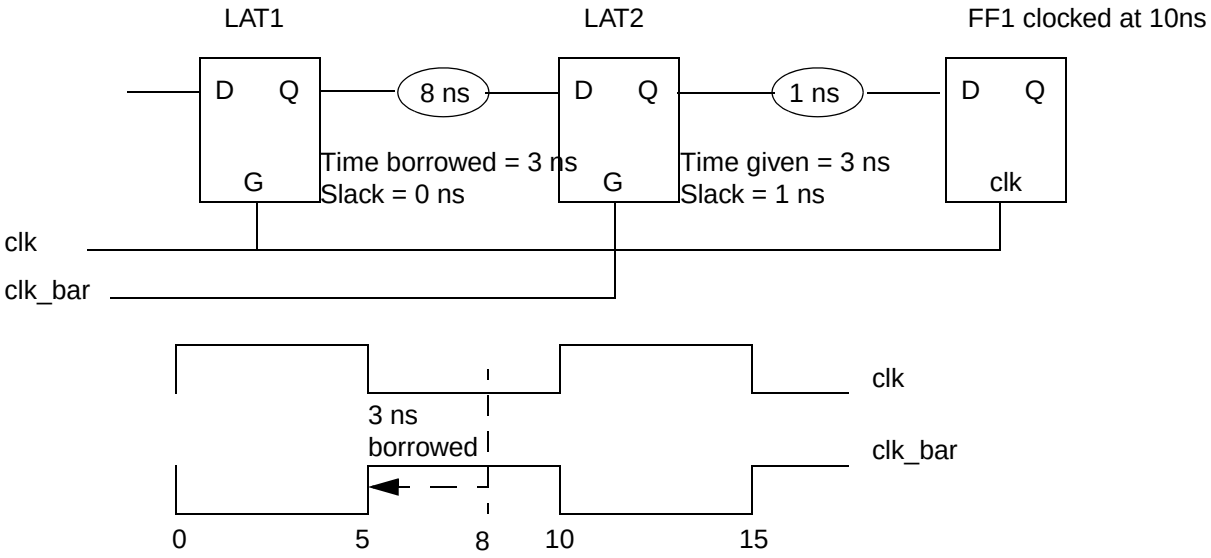


Not all timing analysis engines use the same time borrowing algorithm.

Maximum Time Borrowing

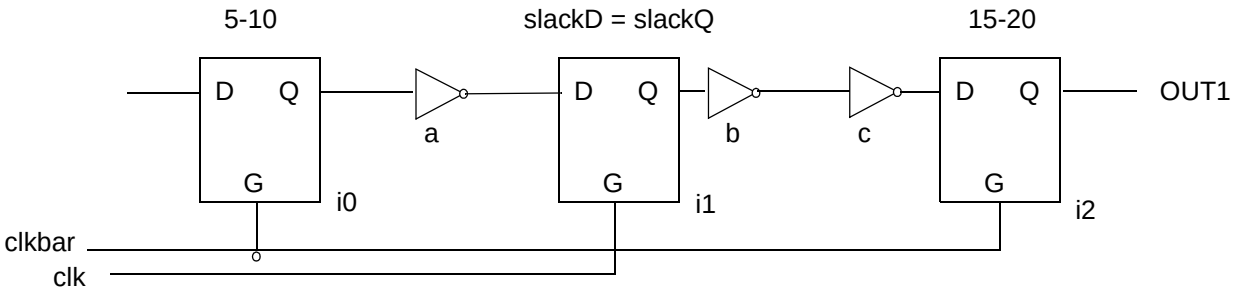
Maximum time borrowing is shown in Figure 4-4. Assume that the setup time is zero.

Figure 4-4 Max Time Borrowing 1



For the `LAT1` to `LAT2` path, the signal arrives at `8` ns while the clock arrives at `5` ns. However, since `LAT2` is still open, this path borrows `3` ns from the next path (`LAT2` to `LAT3`), and thereby meets setup, as shown in Figure 4-5.

Figure 4-5 Max Time Borrowing 2



The amount of slack needed is borrowed, unless that amount causes a late arrival for the next stage. By default, the maximum that can be borrowed is the pulse width-setup, or you can set a limit.

Example 4-20 shows a latch timing report.

Example 4-20 Latch Timing Report

```
============================================================
   Pin            Type         Fanout Load Slew Delay Arrival
                                      (fF) (ps)  (ps)    (ps)
------------------------------------------------------------------
(clock ck)    launch                                        0 R
              latency                              +44      44 R
              uncertainty                          -89     -45 R
latch2/ENA    latchd                                       -45 R
latch2/D      lent                                 +436     392 R
              latch_d_arrival                        +0     392 R
latch2/Q      latchd                1 20.4   16    +148     540 R
latch3/D      latchd                               +0       540
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock ck)    open                                          0 R
              latency                              +44      44 R
              uncertainty                          -89     -45 R
latch3/ENA    borrowed                             +584     540
------------------------------------------------------------------
Timing slack :       0ps
Start-point  : latch2/D
End-point    : latch3/D
```

5

Performing Multi-Mode Timing Analysis

Overview of Multi-Mode Timing Analysis

In today's design flow, designs need to work in different modes. The goal of multi-mode timing analysis is to analyze and optimize different timing modes simultaneously. A mode signifies the functional or operational behavior of a design.

Modes are controlled by a set of constraints that constrain the design and drive timing analysis. A design may have several functional modes, such as test, scan, and normal functional modes. For example, in a Multiple Supply Voltage (MSV) design, a normal functional mode can be further divided into different shut down modes. The timing constraints for these modes can vary and sometimes conflict from one mode to another.

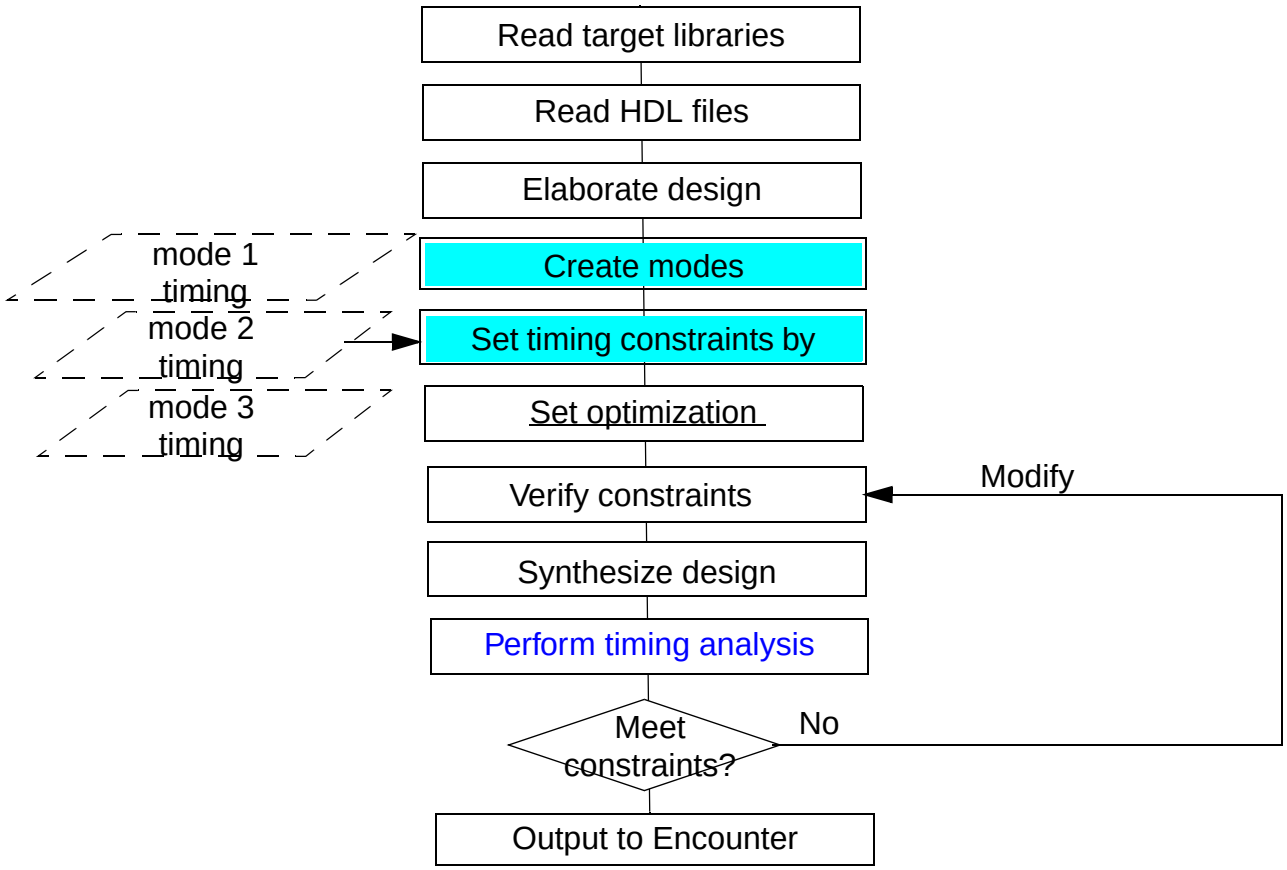
In a traditional synthesis flow, you perform synthesis in each mode and try to close timing by synthesizing all the different timing constraints. This can introduce a critical path in another mode while you are trying to close timing in the current mode. RTL Compiler's multi-mode support reduces the extra design cycle and performs timing analysis and optimization simultaneously based on multiple SDC constraints.

Multi-Mode Timing Analysis and Optimization Design Flow

Figure 5-1 on page 167 shows the multi-mode timing analysis flow. Make sure you create modes after loading and elaborating the design, before reading in an SDC file, and before setting the constraints.

Example 5-1 on page 167 provides a sample script for the multi-mode timing analysis flow.

Figure 5-1 Multi-Mode Timing Analysis and Optimization Flow



Example 5-1 Sample Script for Multi-Mode Timing Analysis and Optimization Flow

```
set_attribute lib_search_path ...
set_attribute hdl_search_path ...
set_attribute library ...
read_hdl <hdl_file_name>
elaborate
create_mode -name {function LBIST MBIST}
read_sdc -mode function core_function.sdc io_function.sdc
read_sdc -mode LBIST core_LBIST.sdc io_LBIST.sdc
read_sdc -mode MBIST core_MBIST.sdc io_MBIST.sdc
synthesize -to_map
report timing
write_encounter design
```

Creating Modes

Before using the `create_mode` command, load the libraries and the design.

➤ After reading and elaborating the design, use the following command to create new modes:

```
rc:/> create_mode [-design design] -name mode_names
```

Note: When there are multiple designs, specify the `-design` option when using the `create_mode` command.

Example 5-2 Creating Multiple Modes Using One Command

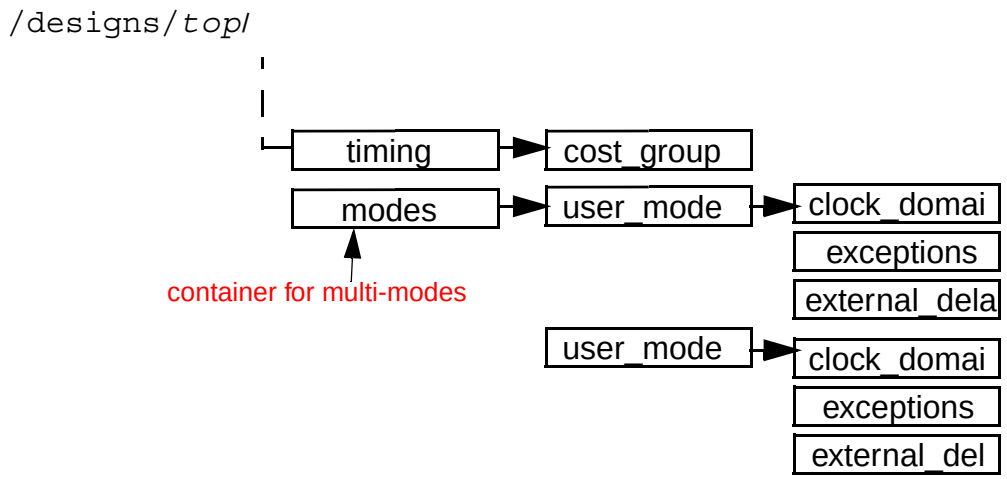
```
rc:/> create_mode -name {mode1 mode2}
```

Example 5-3 Creating Separate Modes Using Multiple Commands

```
rc:/> create_mode -name a
/designs/top/modes/a
rc:/> create_mode -name b
/designs/top/modes/b
rc:/> find /-mode *
/designs/top/modes/a /designs/top/modes/b
```

After creating a new mode, a container called *modes*, as shown in Figure 5-2, displays in the design information hierarchy under the top design: `/designs/top/`, which is used to store all the modes you create. The modes container stored under `/timing`, includes the following three components: `clock_domains`, `exceptions`, and `external_delays`.

Figure 5-2 Multi-Mode Design Information Hierarchy



See the “RTL Compiler Design Information Hierarchy” chapter in the *Using Encounter RTL: Compiler* manual for detailed information.

Note: After creating a mode, if you set a timing constraint without using the `-mode` option, then this will result in an error, as shown in Example Example 5-4 on page 169.

Example 5-4 Error Annotating Constraints After Defining Modes

```
rc:/> read_hdl filename.v
rc:/> elaborate
rc:/> create_mode -name {a b}
rc:/> define_clock -name my_clk -period 100

Error   : Unspecified mode [TUI-226] [define_clock]
        : No mode specified for clock.
        : In multi-mode timing, a mode must be specified using the -mode option.
```

Using Multi-Mode Analysis Timing Commands

RTL Compiler Mode-Specific Timing Commands

➤ Use the `-mode` option with the following commands to create a mode-specific timing constraint. See Setting RTL Compiler Mode-Specific Timing Constraints on page 173.

- `define_clock`
- `external_delay`
- `multi_cycle`
- `path_adjust`
- `path_delay`
- `path_disable`
- `path_group`
- `specify_paths`

SDC Mode-Specific Timing Commands

The following constraints are supported for multi-mode analysis and optimization and will be stored in the design hierarchy under the corresponding mode that you create. See Setting Mode-Specific SDC Timing Constraints on page 174.

- `create_clock`
- `create_generated_clock`
- `group_path`
- `set_false_path`
- `set_input_delay`
- `set_output_delay`
- `set_max_delay`
- `set_max_time_borrow`
- `set_multicycle_path`
- `set_case_analysis`

- `set_disable_timing`

Note: The SDC mode-specific constraints take the `-mode` option when applied with the `dc::` prefix in the RTL Compiler command line. However, the `-mode` option is not required if you read in those constraints using the `read_sdc -mode` *`mode_name`* command.

The following SDC commands are supported for multi-mode if they are applied on a clock object, as shown in Example 5-5 on page 171. If applied on clock pins, then these commands have a global effect, which means that the effects are the same across all modes, as shown in Example 5-6 on page 171.

- `set_clock_uncertainty`
- `set_clock_latency`

Note: These two SDC constraints do not take the `-mode` option when applied with the `dc::` prefix in the RTL Compiler command line.

Example 5-5 Applying set_clock_latency on a Clock Object in Mode a

The following command adds `90ps` on `CLK1` in mode `a` only:

```
rc:/> dc::set_clock_latency -rise 0.09 [dc::get_clocks CLK1]
```

Example 5-6 Global Mode Effect of the set_clock_latency Command

```
rc:/> dc::set_clock_latency -rise 0.05 [find / -pin ff3/CK]
```

See Supported SDC Commands on page 28 for a complete list of supported commands.

SDC Mode-Independent Commands

The following commands are not mode dependent, which means that you cannot set them for specific modes. Using these commands will have the same effect across all modes.

- `set_clock_gating_check`
- `set_clock_transition`
- `set_max_transition`
- `set_drive`
- `set_driving_cell`
- `set_fanout_load`
- `set_ideal_net`
- `set_input_transition`
- `set_load`
- `set_operating_conditions`
- `set_port_fanout_number`
- `set_wire_load_mode`
- `set_wire_load_model`
- `set_wire_load_selection_group`

Setting Timing Constraints in a Multi-Mode Environment

Different Ways of Setting Timing Constraints

You can apply multi-mode timing constraints by:

- Setting RTL Compiler Mode-Specific Timing Constraints
- Setting Mode-Specific SDC Timing Constraints using the `dc::`*`SDC command`* with the `-mode` option.
- Reading SDC Files Using the read_sdc -mode Command
- Setting RTL Compiler Mode-Specific Attributes

After creating a mode, if you set a timing constraint without using the `-mode` option, then this will result in an error, as shown in Example 5-4 on page 169.

Setting RTL Compiler Mode-Specific Timing Constraints

➤ Use the `-mode` option with RTL Compiler Mode-Specific Timing Commands to set timing constraints in different modes as shown in Example 5-7 on page 173.

Example 5-7 Setting Mode Specific RTL Compiler in Different Modes

```
rc:/> create_mode -name {a b}
rc:/> set clka [define_clock -mode a -period 500 clk]
rc:/> set clkb [define_clock -mode b -period 750 clk]
rc:/> external_delay -mode a -clock $clka -input 100 [find / -port in]
rc:/> external_delay -mode a -clock $clka -output 100 [find / -port out]
rc:/> external_delay -mode b -clock $clkb -input 200 [find / -port in]
rc:/> external_delay -mode b -clock $clkb -output 200 [find / -port out]
```

Note: Make sure you read the libraries and load the design before creating modes.

Setting Mode-Specific SDC Timing Constraints

➤ Use the `set_mode` SDC command to set timing constraints in different modes, as shown in Example 5-8 on page 174.

Note: When you use the `set_mode` SDC command, the next SDC commands apply to that mode until you change the mode or unset the mode.

Example 5-8 Setting SDC Timing Constraints in Different Modes

```
rc:/> create_mode -name {a b}
rc:/> dc::set_mode a
rc:/> dc::create_clock -name clka -period 0.5 clk
rc:/> dc::set_input_delay 0.1 -clock [dc::get_clocks {clka}] \
      [dc::get_ports in]
rc:/> dc::set_output_delay 0.1 -clock [dc::get_clocks {clka}] \
      [dc::get_ports out]
rc:/> dc::set_mode b
rc:/> dc::create_clock -name clkb -period 0.75 clk
rc:/> dc::set_input_delay 0.2 -clock [dc::get_clocks {clkb}] \
      [dc::get_ports in]
rc:/> dc::set_output_delay 0.2 -clock [dc::get_clocks {clkb}] \
      [dc::get_ports out]
rc:/> dc::set_mode
```

Note: Make sure you read the libraries and load the design before creating modes.

Reading SDC Files Using the read_sdc -mode Command

Reading Individual SDC Files

➤ After creating all the modes, individually read SDC files using the `read_sdc -mode` command, as shown in Example 5-9 on page 174.

Note: If there are conflicting SDC command definitions that have global effects (effects all modes), then the last definition takes effect.

Example 5-9 Reading One SDC File At a Time

```
rc:/> read_sdc -mode function core_function.sdc
rc:/> read_sdc -mode function io_function.sdc
rc:/> read_sdc -mode LBIST core_LBIST.sdc
```

Reading Multiple SDC Files

Example 5-10 Creating a Mode and Reading Multiple SDC Files at the Same Time

```
rc:/> create_mode -name {function LBIST MBIST}
rc:/> read_sdc -mode function core_function.sdc io_function.sdc
rc:/> read_sdc -mode LBIST core_LBIST.sdc io_LBIST.sdc
rc:/> read_sdc -mode MBIST core_MBIST.sdc io_MBIST.sdc
```

If you read an SDC file before using the `create_mode` command and the SDC file contains SDC Mode-Specific Timing Commands, then the `create_mode` command will fail. If this happens, use the `reset_design` command to remove the *clock_domains*, *exceptions*, and *external_delays* sub-directories under the *timing* directory, then you can successfully create new modes.

Tip

If you set a non-mode specific constraint, then you can still create a mode using the `create_mode` command, as shown in Example 5-11 on page 175.

Example 5-11 Creating Modes After Setting Constraints

```
rc:/> read_hdl filename.v
rc:/> elaborate
rc:/> read_sdc set_driving_cell.sdc
rc:/> create_mode -name {a b}
rc:/> read_sdc -mode a define_clk_for_mode_a.sdc # non-mode specific constraint
rc:/> read_sdc -mode b define_clk_for_mode_b.sdc
```

Setting RTL Compiler Mode-Specific Attributes

Set the following RTL Compiler timing attributes to specify mode-specific values.

- `disabled_arcs_by_mode`
- `latch_borrow_by_mode`
- `latch_max_borrow_by_mode`
- `timing_case_logic_value_by_mode`

Setting Timing Constraints

Specifying Clock Information

➤ Define clocks by mode using the define_clock -mode command. For example:

```
rc:/> define_clock -mode a -name CLK -period 100 [find / -port clk]
```

The SDC equivalent of the define_clock command is create_clock. For example:

```
rc:/> dc::create_clock -mode a -name CLK -period 0.1 [dc::get_ports clk]
```

Specifying External Delays

The external_delay constraint is specified as either an input or an output delay, and is specified relative to a clock edge.

➤ Constrain ports and pins by mode within your design using the external_delay command.

For example, the following command applies input delay to the in port for mode a:

```
rc:/> external_delay -mode a -input 0 -clock \
      [find /designs/top/modes/a -clock CLK] [find / -port in]
```

The SDC equivalents of the external_delay command is the set_input_delay command and the set_output_delay command. For example:

```
rc:/> dc::set_input_delay -mode a 0 -clock [dc::get_clocks {CLK}] \
      [dc::get_ports in]
```

Specifying False Paths

➤ Use the path_disable command to specify a false path. For example, the following command disables timing paths starting from port in:

```
rc:/> path_disable -mode a -from [find / -port in]
```

The following is the SDC equivalent command:

```
rc:/> dc::set_false_path -mode a -from [dc::get_ports in]
```

Specifying Multi-Cycle Paths

➤ Specify multi-cycle paths using the multi_cycle command. For example:

```
rc:/> multi_cycle -mode a 2 -setup -from [find / -port in] \
      -to [find / -port out]
```

The SDC equivalent command is:

```
rc:/> dc::set_multicycle_path -mode a 2 -setup -from [get_ports in] \
      -to [dc::get_ports out]
```

Specifying Path Delays

➤ Specify a path delay exception using the path_delay command. For example:

```
rc:/> path_delay -mode a -delay 5000 -from [find / -port in]
```

The SDC equivalent command is:

```
rc:/> dc::set_max_delay -mode a 5 -from [dc::get_ports in]
```

Specifying Mode-Specific Logic Values

➤ Force the timing engine to assume specified logic values for particular timing modes using the following attribute:

```
rc:/> set_attribute timing_case_logic_value_by_mode
```

For example, the following command sets the Q pin to 1 for mode a:

```
rc:/> set_attribute timing_case_logic_value_by_mode {{a 1}} [find / -pin F1/Q]
```

Use this attribute for pins or ports.

➤ You can see if the value of a pin was computed to have a constant logic value using the following attribute:

```
rc:/> get_attribute timing_case_computed_value_by_mode [find / -pin F1/Q]
{/designs/top/modes/a 1}
```

The SDC equivalent command is:

```
rc:/> dc::set_case_analysis -mode a 1 [dc::get_pins F1/Q]
```

Disabling Mode-Specific Timing Arcs

➤ Specify libarc objects to be disabled in each timing mode using the following attribute:

```
rc:/> set_attribute disabled_arcs_by_mode {{mode [libarcs]...} \
      {{mode [libarcs]...} ... object_name
```

For example:

```
rc:/> set_attribute disabled_arcs_by_mode [list [list a [find [find / -libpin
NAND2X1/Y] -libarc A_Y_*]]] [find . -inst nand_inst]
```

The SDC equivalent command is:

```
rc:/> dc::set_disable_timing -mode a -from [dc::get_pins nand_inst/A] \
     -to [get_pins nand_inst/Y] nand_inst
```

➤ Check and see if any of the timing arcs for instances are disabled due to case analysis for each timing mode using the following attribute:

```
rc:/> get_attribute timing_case_disabled_arcs_by_mode instance_name
```

For example, the following command disables the `A` to `Y` arc of the `nand_inst` instance:

```
rc:/> get_attribute disabled_arcs_by_mode nand_inst
{/designs/top/modes/a /libraries/typical/libcells/NAND2X1/Y/inarcs/A_Y_n60}
```

If different modes disagree on whether a timing arc should be disabled, then delay calculation and combinational loop breaking will treat the arc as if it is not disabled and one set of delay calculation values will be used across all modes.

However, arrival and required time propagation will be blocked for a particular mode if a timing arc is disabled in that mode. This means that mode specific disabling of arcs, either directly or by using the SDC `set_case_analysis` command, will behave similar to the SDC `set_false_path` command.

Specifying Latch Time Borrowing

➤ Use the `latch_borrow_by_mode` attribute for top designs and instances to specify latch time borrow values for different modes. For example:

```
rc:/> set_attribute latch_borrow_by_mode [list [list a 20]] /designs/top/
Setting attribute of design 'top': 'latch_borrow_by_mode' = {/designs/top/
modes/a 20.0}
```

Specifying Max Time Borrowing

➤ Specify the maximum time that can be borrowed from the next cycle to meet timing constraints using the `latch_max_borrow_by_mode` attribute. For example:

```
rc:/> set_attribute latch_max_borrow_by_mode [list [list a 10]] \
      [find / -instance inst_1]
{/designs/top/modes/a 10.0}
```

The SDC equivalent command is:

```
rc:/> dc::set_max_time_borrow -mode a 10000 inst_1
```

Getting Mode-Specific Object Information

Getting External Delay Values

➤ Return a list of external delays for each mode on a pin or a port using the following attribute:

```
rc:/> get_attribute external_delays_by_mode [find / -port in1]
{/designs/top/modes/b /designs/top/modes/b/external_delays/in_del_1} \
{/designs/top/modes/a /designs/top/modes/a/external_delays/in_del_1}
rc:/> get_attribute delay /designs/top/modes/b/external_delays/in_del_1
20.0 20.0 20.0 20.0
```

The `delay` attribute returns the minimum and maximum rise and fall delay values of the external_delay constraint.

Note: RTL Compiler does not use the minimum values, but storing the minimum values allows RTL Compiler to write out the SDC constraints correctly.

Getting Slack Values

➤ Return a list of slack values for each timing mode for a design, pin, port, or cost group using the following attribute:

```
rc:/> get_attribute slack_by_mode /designs/top/
{/designs/top/modes/b -166.2} {/designs/top/modes/a -106.2}
```

➤ Use the `slack` attribute to return the worst slack value across all modes.

For example, the following command returns the `-166.2` slack value since it is the worst slack between modes `a` and `b`:

```
rc:/> get_attribute slack /designs/top/
-166.2
```

Getting Clock Information

➤ Return a Tcl list of clock information for each mode on a pin or a port that has propagated to the specified pin using the following attribute:

```
rc:/> get_attribute propagated_clocks_by_mode [find / -port ck]
{/designs/top/modes/b \
{{clock /designs/top/modes/b/clock_domains/domain_1/clockb phase +......}}} \
{/designs/top/modes/a \
{{clock /designs/top/modes/a/clock_domains/domain_1 clock phase + ......}}}
```

Getting Computed Disabled Arc Information

➤ Return a Tcl list of computed disabled arc information for each mode for an instance using the following attribute:

```
rc:/> get_attribute timing_case_disabled_arcs_by_mode \
      [find / -instance inst_1]
{/designs/top/modes/a /libraries/typical/libcells/NAND2X1/Y/inarcs/A_Y_n60}
```

Getting Disabled Arc Information

➤ Return a Tcl list of disabled arc information for each mode for an instance using the following attribute:

```
rc:/> get_attribute disabled_arcs_by_mode [find / -instance inst_1]
{/designs/top/modes/a /libraries/typical/libcells/NAND2X1/Y/inarcs/A_Y_n60}
```

Getting Timing Case Computed Value Information

➤ Return a Tcl list of timing case computed values for each mode on a pin or a port using the following attribute:

```
rc:/> get_attribute timing_case_computed_value_by_mode [find / -pin F1/Q]
{/designs/top/modes/a 1}
```

Getting Timing Case Logic Value Information

➤ Return a Tcl list of timing case logic value information for each mode on a pin or a port using the following attribute:

```
rc:/> get_attribute timing_case_logic_value_by_mode [find / -pin F1/Q]
{/designs/top/modes/a 1}
```

Getting the Maximum Latch Borrow Value Information

➤ Return a Tcl list of latch max borrow value information for each mode for a design, an instance, and a pin using the following attribute:

```
rc:/> get_attribute latch_max_borrow_by_mode [find / -pin i0/ENA]
{/designs/top/modes/b 20000.0}
```

Getting the Latch Borrow Value Information

➤ Return a Tcl list of latch borrow value information for each mode for a design and an instance using the following attribute:

```
rc:/> get_attribute latch_borrow_by_mode [find / -pin i0/ENA]
{/designs/top/modes/b 20000.0}
```

Analyzing and Reporting Multi-Mode Information

- Use the `report clocks [-mode mode_name]` command to generate a report on the clocks of the current design. If the `-mode mode_name` option is specified, then only the clocks defined in that mode will be reported.
- Use the `report summary` command to report the area used by the design, the cells mapped for the blocks in the specified design, the wireload model, and the timing slack of the critical path. It also reports if any design rule is violated and the worst violator information. The slack shown in the report summary will be the worst slack across all modes.
- Use the `report timing` command to generate a timing report for the current design.

  If you do not use the `-mode mode_name` option, then the `report timing` command will report the worst slack for each mode. If you use the `-mode mode_name` option, then the `report timing` command will show the worst slack in that particular mode.

Writing out the Multi-Mode Environment

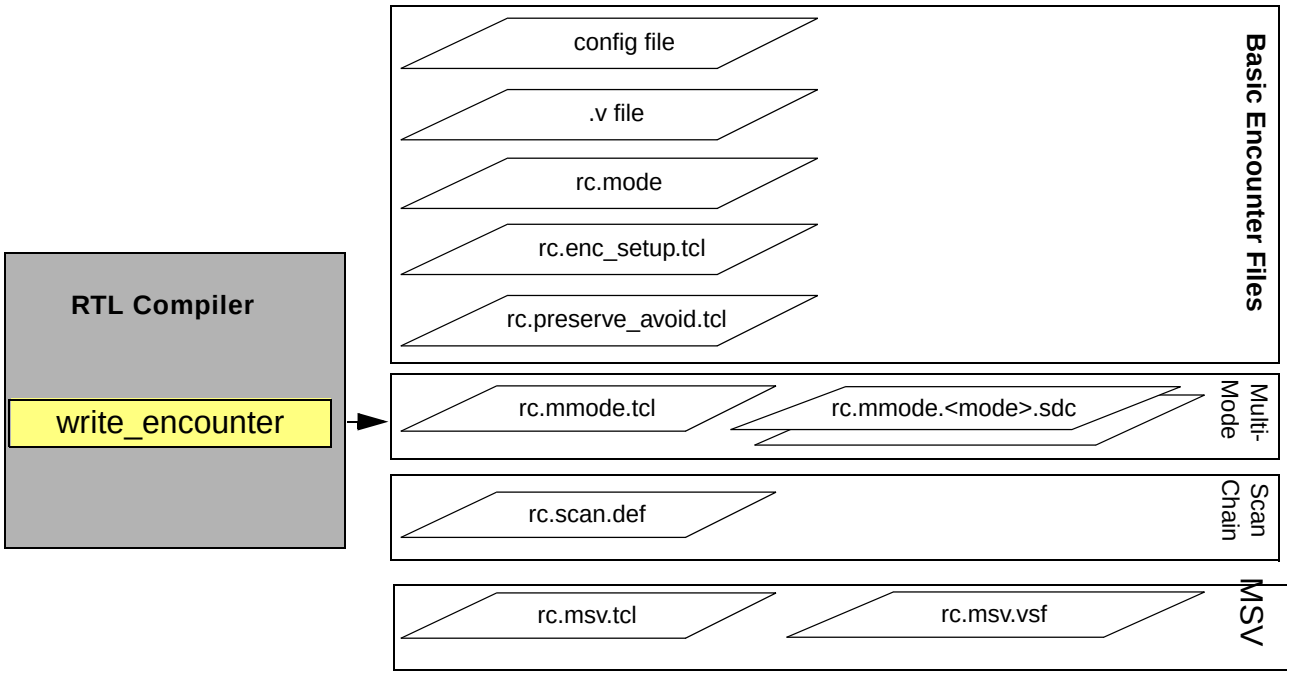
- Use the `derive_environment` command to create a new design from the specified instance and create timing constraints for this design based on the timing constraints that the instance had in the original design. If you are deriving an environment in a multi-mode environment, then all the mode information will be derived into the new environment.
- Use the `write_encounter` *`design`* command to write Encounter input files to a single directory, as shown in Figure 5-3.
- Use the `write_script` command to generate a script that represents the timing and design rule constraints of the design. The resulting script can subsequently be used to examine the current design constraints, or it can be read back into RTL Compiler to perform analysis or optimization at a later time.

Outputting to Encounter

➤ Write Encounter input files to a single directory using the `write_encounter` *`design`* command.

Figure 5-3 shows the output `write_encounter` command files. All the output files will be stored under the `rc_enc_des` directory and the output files will have the "`rc.`" prefix.

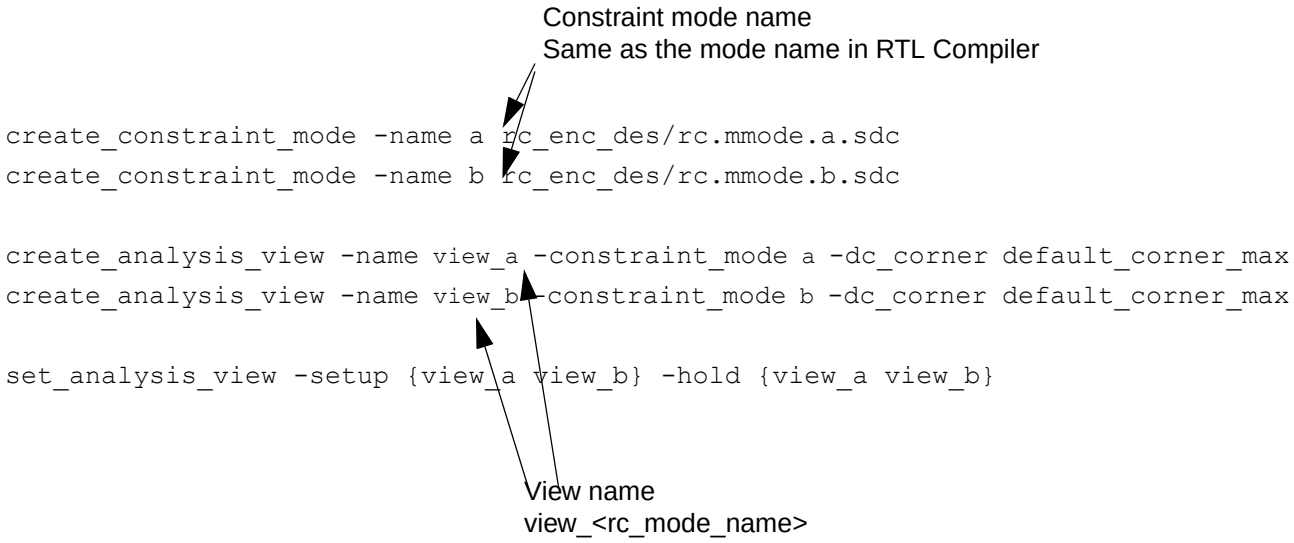
Figure 5-3 write_encounter Command Output Files



If your design has scan chains, then a scan DEF file, such as `rc.scan.def`, as shown in Figure 5-3, will be created. If your design is a MSV design, then a MSV setup file and a Shifter table file, such as `rc.msv.tcl`, `rc.msv.vsf` will be created. If your design is in multi-mode environment, then a FE view file, such as `rc.mmode.tcl`, along with SDC files (one for each mode) will be created.

Example 5-12 on page 183 shows the multi-mode view (`rc.mmode.tcl`) file.

Example 5-12 Multi-Mode View File (rc.mmode.tcl)



- Use the `report timing [-view {listofViewNames}]` command to report timing in Encounter.

Writing Out Multi-Mode SDC Constraints

If you used the `write_encounter` command, then each mode will have a corresponding SDC file written out. For more information, see the `write_encounter` command.

- Write out one SDC file for one mode using the `-mode` option with the `write_sdc` command. For example:

  ```
  rc:/> write_sdc -mode mode1 mode1.sdc
  ```

Make sure you specify the `-mode` option with the `write_sdc` command or the `write_sdc` command will fail.

Note: RTL Compiler does not write out merged SDC information for all the modes.

Performing Power Analysis in a Multi-Mode Environment

If there are different clock periods defined on the same clock pin or clock port between different modes, then RTL Compiler uses the last one defined as the toggle count and performs power analysis based on it.

In Example 5-13 on page 184, since a 2.5Mhz (400 ps) clock for mode b is the last one defined, then the RTL Compiler low power engine uses 5/ns as the toggle rate on the clk clock net.

Example 5-13 Using the Last Defined Clock Period as the Toggle Count for Power Analysis

```
rc:/> create_mode -name [list a b]
rc:/> set clock [define_clock -name clock -mode a -period 200 [find / -clock clk]
rc:/> set clock [define_clock -name clock -mode b -period 400 [find / -clock clk]
rc:/> get_attribute lp_computed_toggle_rate [find / -net clk]
5
```

In Example 5-14 on page 184, since a 5Mhz (200ps) clock for mode a is defined last, then the RTL Compiler low power engine uses10/ns as the toggle rate on the clk clock net.

Example 5-14 Using the Last Defined Clock Period as the Toggle Count for Power Analysis

```
rc:/> create_mode -name [list a b]
rc:/> set clock [define_clock -name clock -mode b -period 400 [find / -clock clk]
rc:/> set clock [define_clock -name clock -mode a -period 200 [find / -clock clk]
rc:/> get_attribute lp_computed_toggle_rate [find / -net clk]
10
```

For more information on low power analysis, see *Low Power in Encounter RTL Compiler*.

Additional Information

Using the Same Names for Timing Objects in Different Modes

You can use the same names for `clock_domains`, `exceptions`, and `external_delays` timing objects as long as they are in different modes.

In the following example, in one mode you define a clock called *`CLOCK`* and in another mode you define a different clock with the name *`CLOCK`*, as shown in Example 5-15 on page 185.

Example 5-15 Creating Two Clocks With the Same Name in Different Modes

```
rc:/> create_mode -name {a b}
rc:/> define_clock -mode a -name clk -period 100 [find / -port clk]
/designs/my_design/modes/a/clock_domains/domain_1/clk
rc:/> define_clock -mode b -name clk -period 20 [find / -port clk]
/designs/my_design/modes/b/clock_domains/domain_1/clk
```

When you want to apply external delay and exceptions, make sure you specify which clock you are referring to. For example, the following command applies input delay to the `in` pin for mode `a` while referring to clock `CLK` in mode `a`:

```
rc:/> external_delay -mode a -input 0 -clock CLK [find / -port ports_in/in]
```

This also applies to the SDC equivalent command:

```
rc:/> dc::set_input_delay 0 -mode a -clock [dc::get_clocks {CLK}] \
     [dc::get_ports in]
```

Using the following command to apply input delay will fail because there are two clocks with the same `CLK` name:

```
rc:/> external_delay -mode a -input 0 -clock \
      [find /designs/top/modes/a -clock CLK] [find / -port ports_in/in]
```

The following SDC command will also fail:

```
rc:/> dc::set_input_delay 0 -mode a -clock {CLK} [dc::get_ports in]
```

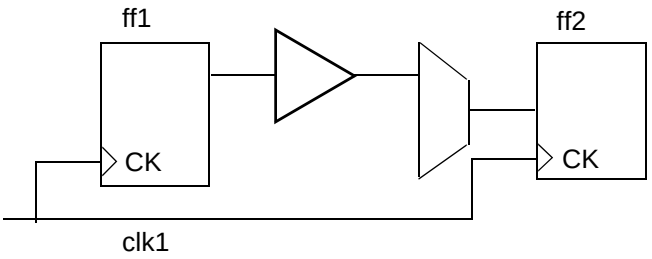
Unconstrained Timing Paths

Applying the following SDC commands in the middle of a timing path can segment (snip) that timing path across all modes and can leave the original timing paths unconstrained in some modes.

- `set_max_delay`
- `set_input_delay`
- `set_output_delay`

As shown in Figure 5-4 and Example 5-16 on page 188, both the `sdc1` mode and the `sdc2` mode show a timing path from `ff1/CK` to `ff2/CK` before applying the `set_max_delay` constraint.

Figure 5-4 Timing Report from ff1/CK to ff2/CK in 'sdc2' Mode




```
Pin          Type             Fanout Load Slew Delay Arrival
                                     (fF) (ps)  (ps)    (ps)
-----------------------------------------------------------
(clock CLK1)   launch                                 0 R
ff1/CK                                 0              0 R
ff1/Q          DFFHQX1      1  1.4    19   +52       52 F
buf_i/A                                      +0       52
buf_i/Y        BUFX2        1  1.5    13   +29       81 F
mux_i_2/B                                    +0       81
mux_i_2/Y      MX2X1        1  1.6    21   +40      122 F
ff2/D          DFFHQX1                       +0      122
ff2/CK         setup                    0    +26     148 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock CLK1)   capture                             2000 R
-----------------------------------------------------------
Timing slack : 1852ps
Start-point  : ff1/CK
End-point    : ff2/D
Mode         : sdc2
```

The timing report shown in Example 5-16 on page 188 shows a positive slack of `9.852 ps` from `ff1/CK` to `ff2/D` in the `sdc1` mode.

Example 5-16 Timing Report from ff1/CK to ff2/CK in 'sdc1' Mode

```
 Pin             Type           Fanout Load Slew Delay Arrival
                                       (fF) (ps)  (ps)    (ps)
-------------------------------------------------------------
(clock CLK1)    launch                                0 R
ff1/CK                                 0              0 R
ff1/Q           DFFHQX1      1  1.4    19   +52       52 F
buf_i/A                                      +0       52
buf_i/Y         BUFX2        1  1.5    13   +29       81 F
mux_i_2/B                                    +0       81
mux_i_2/Y       MX2X1        1  1.6    21   +40      122 F
ff2/D           DFFHQX1                      +0      122
ff2/CK          setup                  0     +26      148 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock CLK1)    capture                             10000 R
-------------------------------------------------------------
Timing slack : 9852ps
Start-point  : ff1/CK
End-point    : ff2/D
Mode         : sdc1
```

If you set the `set_max_delay` constraint in the `sdc2` mode from `ff1/CK` to `mux_i_2/B`, as shown in the following example:

```
rc:/designs/top> dc::set_max_delay 5 -mode sdc2 -from ff1/CK -to mux_i_2/B
```

Then the following warning message warns that a `max_delay` is set in the middle of a timing path.

```
Warning : At least one to-point is not a timing endpoint. [TIM-317]
        : Provided to_point is '/designs/top/instances_comb/mux_i_2/pins_in/B'.
        : If the to-point does not become a valid timing endpoint later on (perhaps
as a result of a set_max_delay constraint or set_disable_timing), then the
exception will not be applied to this to-point.
```

Despite the warning, the timing path between `ff1/CK` and `ff2/CK` is snipped for both modes, as shown in the following examples:

```
rc:/designs/top> report timing -from ff1/CK -to ff2/D -mode sdc1
No paths found.
rc:/designs/top> report timing -from ff1/CK -to ff2/D -mode sdc2
No paths found
```

Example 5-17 Timing Report for ‘in2’ Pin

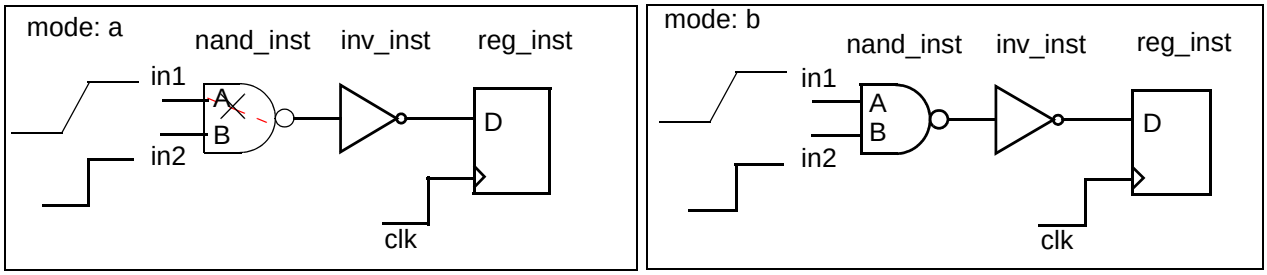
```
rc:designs/top> report timing -mode a -from in2 -to reg_inst/D

    Pin              Type     Fanout Load  Slew   Delay  Arrival
                                     (fF)  (ps)    (ps)    (ps)
-------------------------------------------------------------------
(clock clock)      launch                                    0 R
(in_del_1)         ext delay                    +0           0 F
in2           <<<  in port         1  7.0    0    +0           0 F
nand_inst/a2                                    +0           0
nand_inst/zn       nand_libcell    1  6.8 7559  +274         274 R
inv_inst/i                                      +0         274
inv_inst/z         inv_libcell     1  7.2  303  +929        1203 R
reg_inst/D    <<<  reg_libcell                  +0        1204
reg_inst/cp        setup                    0  +302        1505 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock)      capture                                 100 R
-------------------------------------------------------------------
Timing slack : -1405ps (TIMING VIOLATION)
Start-point  : in2
End-point    : reg_inst/D
Mode         : a
```

Figure 5-5 shows the `set_case_analysis` command applied on the input `in1` pin using the following command:

```
rc:/> dc::set_case_analysis 1 -mode a [dc::get_pins nand_inst/A]
```

The reported slew for `nand_inst/zn` is still the same after the timing path from the `in1` to the `nand_inst` instance is disabled.

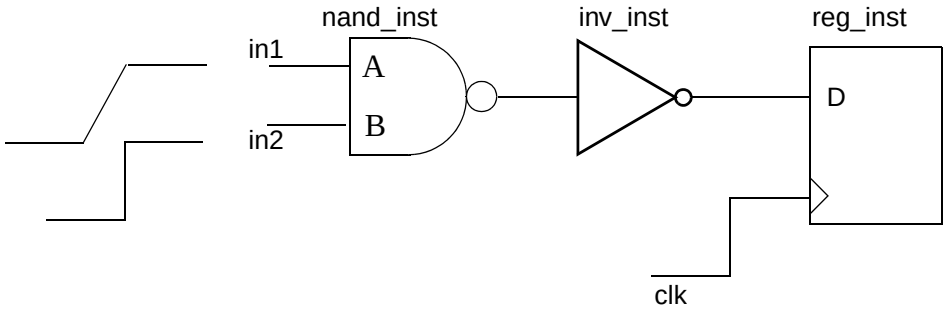
Figure 5-5 Applying the SDC set_case_analysis on Input ‘in1’ in Mode ‘a’




```
Pin                  Type      Fanout Load Slew    Delay      Arrival
                                      (fF) (ps)    (ps)       (ps)
-------------------------------------------------------------------
(clock clock)        launch                                     0 R
(in_del_1)           ext delay                    +0            0 F
in2            <<<   in port          1  7.0    0    +0            0 F
nand_inst/a2                                      +0            0
nand_inst/zn         nand_libcell      1  6.8 7559   +274      274 R
inv_inst/i                                        +0          274
inv_inst/z           inv_libcell       1  7.2  303   +929     1203 R
reg_inst/D     <<<   reg_libcell                   +0          1204
reg_inst/cp          setup                      0  +302     1505 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock)        capture                                  100 R
-------------------------------------------------------------------
Timing slack : -1405ps (TIMING VIOLATION)
Start-point  : in2
End-point    : reg_inst/D
Mode         : a
```

Compared to the same scenario in single mode, as shown in Figure 5-6, reporting the `in2` input pin in single mode timing analysis shows the same **7559** ps slew on the `nand_inst/zn` pin as that shown in Example 5-17 on page 189.

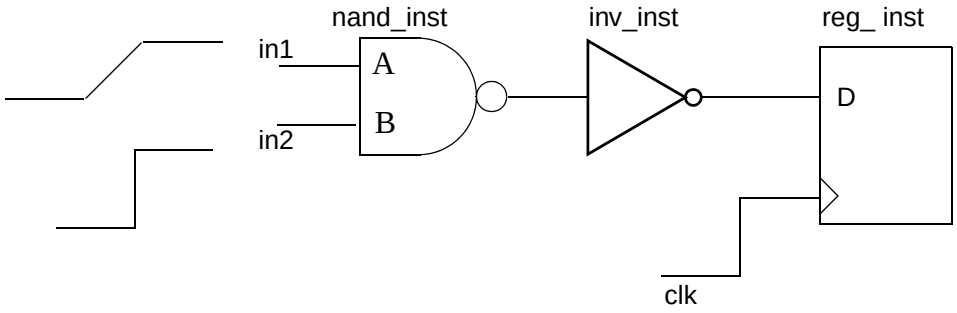
Figure 5-6 Timing Report Before the set_case_analysis Is Applied in Single Mode




```
rc:/designs/top> report timing -from in2 -to reg_inst/D

Pin              Type           Fanout Load Slew Delay Arrival
                                       (fF) (ps)  (ps)   (ps)
-----------------------------------------------------------------
(clock clock)      launch                                  0 R
(in_del_1)         ext delay                    +0         0 F
in2           <<<  in port         1  7.0    0    +0         0 F
nand_inst/a2                                    +0         0
nand_inst/zn       nand_libcell    1  6.8 7559  +274      274 R
inv_inst/i                                      +0      274
inv_inst/z         inv_libcell     1  7.2  303  +929     1203 R
reg_inst/D    <<<  reg_libcell                  +0     1204
reg_inst/cp        setup                     0  +302     1505 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock)      capture                                 100 R
-----------------------------------------------------------------
Timing slack : -1405ps (TIMING VIOLATION)
Start-point  : in2
End-point    : reg_inst/D
```

After the `set_case_analysis` command is applied on the input `in1` pin as shown in Figure 5-7, the slew on the `nand_inst/zn` pin is much smaller: **`155`** ps instead of **`7559`** ps.

Figure 5-7 Timing Report for 'in2' Pin in Single Mode After the Arc has been Disabled




```
rc:/designs/top> dc::set_case_analysis 1 nand_inst/A
rc:/designs/top> report timing -from in2 -to reg_inst/D

Pin              Type               Fanout Load Slew Delay Arrival
                                           (fF) (ps)  (ps)    (ps)
-------------------------------------------------------------------
(clock clock)      launch                                     0 R
(in_del_1)         ext delay                    +0            0 R
in2          <<<   in port          1  7.0    0   +0          0 R
nand_inst/a2                                    +0            0
nand_inst/zn       nand_libcell    1  6.8  155  +347         347 F
inv_inst/i                                      +0          347
inv_inst/z         inv_libcell     1  7.2  106  +205         552 F
reg_inst/D   <<<   reg_libcell                  +0          552
reg_inst/cp        setup                     0  +276         828 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock)      capture                                  100 R
-------------------------------------------------------------------
Timing slack : -728ps (TIMING VIOLATION)
Start-point  : in2
End-point    : reg_inst/D
```

6

Troubleshooting Timing Analysis Issues

Factors that Impact a Timing Report

There are various factors that effect the timing report, such as the wire-load model, library variations, load, and so on. These factors can considerably change the design's timing and slack depending on the library variables.

For example, a library could have a timing characteristic such that it has almost a constant to linear timing curve for lower loads and fanouts that increase exponentially. This effects the slack of the design and is reflected in the report timing path. Example 6-1 shows a timing report with a path to an `y[0]` external port with no external load.

Example 6-1 Path to an External Port with no External Load

```
rc:/> report timing -to y[0]
================================================================================
  Generated by:          RTL Compiler (RC) Version
  Generated on:          Date
  Module:                timing_example
  Technology library:    cb35os141d_typ 7.1/3.3b
  Operating conditions:  NCCOM (balanced_tree)
  Wireload mode:         segmented
================================================================================
    Pin                       Type      Fanout  Load  Slew   Delay   Arrival
                                                (fF)  (ps)    (ps)    (ps)
--------------------------------------------------------------------------------
(clock clk)                   launch                                     0 R
stage_y_reg_0/cp                                          0               0 R
stage_y_reg_0/q               dfnrq1         1   21.0   114    +512     512 F
timing_example/y[0]     <<<   out port                           +0     512 F
(ou_del_1)                    ext delay                          +0     512 F
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                   capture                                  500 R
--------------------------------------------------------------------------------
Timing slack : -12ps (TIMING VIOLATION)
Start-point  : stage_y_reg_0/cp
End-point    : y[0]
```

Example 6-2 shows the same path with `1000fF` of external annotated `pin_cap`.

Example 6-2 Path -to 1000fF External Annotated pin_cap

```
rc:/> set_attr external_pin_cap 1000 y[0]
  Setting attribute of port y[0]: 'external_pin_cap' = 1000.0
rc:/> report timing -to y[0]
        Tracing clock networks.
        Levelizing the circuit.
        Applying wireload models.
        Computing net loads.
        Computing delays.
        Computing arrivals and requireds.
============================================================================
  Generated by:          RTL Compiler (RC) Version
  Generated on:          Date
  Module:                timing_example
  Technology library:    cb35os141d_typ 7.1/3.3b
  Operating conditions:  NCCOM (balanced_tree)
  Wireload mode:         segmented
============================================================================
    Pin                      Type      Fanout   Load  Slew   Delay   Arrival
                                                (fF)  (ps)    (ps)    (ps)
----------------------------------------------------------------------------
(clock clk)                  launch                                     0 R
stage_y_reg_0/cp                                          0              0 R
stage_y_reg_0/q              dfnrq1        1  1021.0  3741   +2051    2051 R
timing_example/y[0]     <<<  out port                          +0    2051 R
(ou_del_1)                   ext delay                         +0    2051 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                  capture                                  500 R
----------------------------------------------------------------------------
Timing slack :   -1551ps (TIMING VIOLATION)
Start-point  : stage_y_reg_0/cp
End-point    : y[0]
```

Comparing Delay Values Between Two Timing Reports

Whenever delay values from two timing reports are different but you suspect they should be the same, first check to see if the delay values are associated with different rise or fall edges. These values may be gate delays, slews, external delays, latch borrow values, clock skew values, slacks, and so on. RTL Compiler reports a `R` or `F` in the timing path report to indicate whether values are associated with a rising or falling logic transition. Comparing a rise value from one report to a fall value from another report is a common mistake.

Tips for Reporting Timing Issues

Finding Timing Problems

How do I find out if my design has one of the following problems?

- Suspicious clocking, such as unclocked registers, multiple-clocked registers, or strange clock frequencies
- Suspicious exceptions such as exceptions that are not satisfied by any paths, are never used, or conflict with others
- Suspicious logic such as non-tristate parallel drivers or combinational loops

➤ Use the `report timing -lint` command.

Handling Timing Exceptions

How do I get a handle to a timing exception so that I can refer to it later?

➤ Use the following command:

```
rc:/> set my_multi_cycle [multi_cycle -capture 2 -to clk]
```

This creates a variable called *my_multi_cycle* that creates a timing exception through the multi_cycle command.

How do I see the timing exceptions in which a given object (pin, port, instance, or clock) participates?

➤ Use the exceptions attribute. For example:

```
rc:/> get_attribute exceptions $object
```

Alternatively, for more detail, use the ls -attribute command with the exceptions attribute. For example:

```
rc:/> ls -a [get_attribute exceptions $object]
```

How can I tell if one exception has a higher priority than another?

Use the following Tcl process:

```
proc compare_excepts {a b} {
    set a_prio [get_attribute user_priority $a]
    set b_prio [get_attribute user_priority $b]

    if {$a_prio == $b_prio} {
      set a_prio [get_attribute priority $a]
      set b_prio [get_attribute priority $b]
    }
    return [expr {$a_prio - $b_prio}]
  }
```

If a timing path satisfies two conflicting timing exceptions, then how do I tell RTL Compiler to use one and not the other?

➤ Set a higher priority on the desired exception using the user_priority attribute. For example:

```
rc:/> set_attribute user_priority 5 $my_multi_cycle
rc:/> set_attribute user_priority 4 $other_exception
```

How do I eliminate a single exception without affecting other exceptions?

➤ Use the rm command. For example:

```
rc:/> rm $my_multi_cycle
```

Reporting and Adjusting Path Exceptions

How can I report paths that are captured by the falling edge of the clock waveform, but not the rise?

➤ Use the `report timing` command with the specify_paths command. For example:

```
rc:/> report timing -paths [specify_paths -to_fall_clock clock1]
```

How do I report paths that specifically use one of my exceptions?

➤ Use the `report timing -exceptions` command. For example:

```
rc:/> report timing -exceptions $my_multi_cycle
```

How do I report paths that end at a certain pin and use one of my exceptions?

➤ Specify the pin and exception names with the `-to` and `-exception` options of the `report timing` command. For example:

```
rc:/>report timing -to $pin -exceptions $my_multi_cycle
```

How do I report paths that satisfy, but not necessarily use, one of my exceptions?

➤ Use the `report timing -paths` command with the get_attribute command. For example:

```
rc:/> report timing -paths [eval [get_attribute paths $my_multi_cycle]]
```

How can I tighten constraints by a particular value on all paths through a certain adder?

➤ Use the path_adjust command. The following example tightens the constraints by 100 picoseconds on all paths through the adder on the output pin:

```
rc:/> path_adjust -delay -100 -through adder/pins_out/*
```

How do I make all paths through a certain pin use default timing constraints?

Use the following Tcl script:

```
# find the highest user-priority value in use
  set priority 0
  foreach exception [find / -exception *] {
    set this_prio [get_attribute user_priority $exception]
    if {$this_prio > $priority} {
      set priority $this_prio
    }
  }
  # create a single-cycle path exception
  set except [multi_cycle -capture 1 -through $pin]
  # make the new exception highest priority
incr priority
  set_attribute user_priority $priority $except
```

Reporting and Adjusting Multi-Cycle Paths

How do I report the multi-cycle path in my design that has the least slack?

➤ Use the `report timing -exception` command with the `find -exception` command. For example:

```
rc:/> report timing -exceptions [find / -exception multi_cycles/*]
```

How do I find out which timing exception caused a path to be multi-cycled?

The exception name is shown in the path output of the `report timing` command.

How do I find the endpoints in my design that have multi-cycle paths ending at them?

➤ Use the `report timing` command with the `-endpoints` and `-exceptions` options with the `find -exception` command. For example:

```
rc:/> report timing -endpoints -exceptions [find / -exception multi_cycles/*]
```

How do I change an exception from being two cycles to three cycles without affecting other exceptions?

➤ Use the multi_cycle, basename and get_attribute commands to specify a particular exception. For example: report timing -slack_limit 0 -endpoint -num_p *some very high number*:

```
rc:/> multi_cycle -capture 3 -name [basename $my_multi_cycle] \
    -paths [get_attribute paths $my_multi_cycle]
```

How can I choose specific edges to use for timing between a two nanosecond clock and a three nanosecond clock?

➤ Use the multi_cycle command with the -launch, -capture, and -from options. For example:

```
rc:/> multi_cycle -launch -1 -capture 2 -from clock2 -to clock3
```

How can I specify an exception that should apply to the falling edge of the clock waveform, but not the rising edge?

➤ Use the multi_cycle command with the specify_paths command. For example:

```
rc:/> multi_cycle -paths [specify_paths -to_fall_clock clock1]
```

Reporting Timing Constraint Violations

How do I get a report that lists every violating endpoint and the slack that is similar to the SDC report_constraints -all_violators command?

➤ Use the report timing command with the following options for timing violations:

```
rc:/>report timing -slack_limit 0 -endpoints -num_paths <some very high number>
```

Use the report design_rules command to report any design rule violations.

Handling Combinational Feedback Loops

How do I handle combinational feedback loops?

RTL Compiler automatically analyzes the elaborated design for combinational feedback loops during timing analysis; for example, when using the `report timing` or the `synthesize` command. Upon detecting such loops, and before performing timing analysis, RTL Compiler selects a buffer from the technology library to serve as a loop breaker instance. RTL Compiler then instantiates this cell along the feedback loop and disables the timing arc inside the cell. These cells follow the `cdn_loop_breaker<`*`number`*`>` nomenclature, and therefore, they are easily identifiable in the netlist using the `find` command. For example:

```
rc:/> find /designs -instance cdn_loop_breaker*
```

Tip

Timing constraints like `set_case_analysis` and `disable_timing` can potentially avoid combinational feedback loops by making the timer ignore certain paths which could have loops. Therefore, it is important to apply timing constraints before loop breaking to minimize unnecessary loop_breaker buffers.

Removing cdn_loop_breaker Instances

RTL Compiler inserts the `cdn_loop_breaker` instances to break combinational feedback loops. To remove these instances from the netlist, use the following command:

`remove_cdn_loop_breaker`

Analyzing a Design Using the Timing Report

The following sections describe how to use the information in the timing report to troubleshoot your design when it does not meet timing:

It assumes that you:

- Synthesized the design down to gates
- Checked that the timing report for your design shows a negative slack
- Tried multiple incremental compiles with the `synthesize -to_gen -effort high` command
- Optimized all violating timing paths by setting the `tns_opto` attribute to `true`

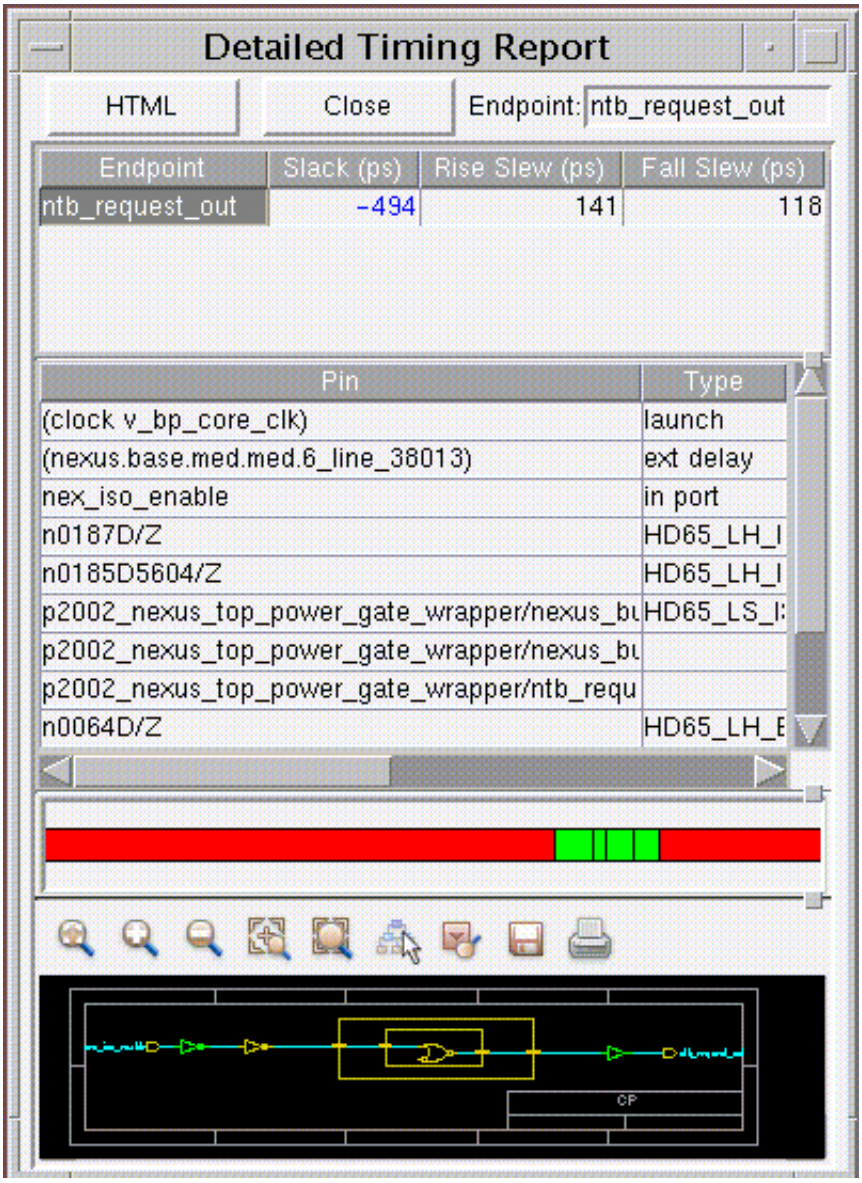
Analyzing the Critical Path

By default, the timing report shows the critical path first. The critical path is the timing path in the design with the greatest amount of negative slack (margin). It is not always the longest path in the design if you are counting the number of instances. However, the critical path is the most difficult path that RTL Compiler has found while trying to meet its timing goals, which are defined by the constraints.

What makes a critical path difficult can be caused by many different factors, some accidental, some intentional, some by chance, and some due to missing data. For this reason, you need to make sure that all inputs to RTL Compiler are correct and accounted for. It is not enough to run a timing report, see a negative slack, and simply report the results. You must examine the timing report to understand why the design is not meeting timing.

Use the `gui_cp_find_violators` command to analyze the critical paths in the GUI. The following example illustrates finding critical path violators by type.

```
rc:/> gui_cp_find_violators -color green -type slew -threshold 125
```



Analyzing the Negative Slack

Is the amount of time between the capture clock edge and the launch clock edge reasonable?

- How much time does a signal really need to go through all the logic in the critical path?

After you account for the capture clock edge time, minus any external input delay, minus the setup of the destination point, minus any uncertainty in the capture clock, and minus any external output delay, how much time is left for the path? It is a common mistake to overconstrain a path, such that you can never meet timing. This is especially true when you have paths that go from input ports to output ports or when paths cross clock domains. Make sure the timing constraints for the path are correct.

- Does the amount of time fit the complexity of the path?

  The amount of functionality in a path has a lot to do with the success of a synthesis run. Look at the amount of time allotted by RTL Compiler for the data to get from the beginning of the path to the end of the path and how much functionality it has to go through. Paths that have too much logic and not a “fair” amount of time are good candidates to make into multicycle paths or false paths, or, if single cycle behavior is expected to be maintained, then you may have to add a register stage in the middle of the path to break it up into a manageable length.

Analyzing Asynchronous and Synchronous Clock Domains

Does the path cross between two different clock domains?

Are the clock domains asynchronous or synchronous to each other?

- By default, RTL Compiler assumes that all clocks in a design are synchronous to each other. This allows RTL Compiler to compute the smallest, non-negative interval between the capture clock edges and the launch clock edges. By default, this smallest interval is used by RTL Compiler during timing analysis.

➤ If you want to use a larger interval, then specify a `multicycle_path` constraint between the clocks.

- If the clock domains are asynchronous to each other, then they should be “false-pathed.”

➤ To establish a false path between clocks, use the `-domain` option with the `define_clock` constraint to assign each clock to a different clock domain.

Using the `-domain` option automatically establishes a false path between its clock domain and all other domains.

Important

RTL Compiler should never be allowed to assign a minimum interval relationship between the clocks if they are asynchronous.

Finding Large Cell Delays

Are there any instances in the path with unexpectedly large cell delays?

- If all the cells have large delays, then there could be a problem with the wire-load model, the operating conditions, or the technology library, all of which affect every cell delay.

  Check your log file to see if there were any warnings when reading in the library, or when applying the constraints to the design.

- If only one or two cells have unexpectedly large delays, then you need to figure out why.

  Look at the *fanout*, *load*, and *slew* columns in the timing report to see if any of those factors correlate with the large delays.

  - Are there any nets with a large number of fanouts in the path?
  - Is there a high fanout that is under driven? How does the fanout versus the drive strength of the cells compare? It should be that a large fanout = large driver and a small fanout = small driver. Consider the total capacitive load, as well as the fanout, when determining the proper drive strength. The total capacitive load is listed in the timing report under the *load* column.
  - Are there any nets with large capacitances in the path?
  - The *load* column is a summation of the wire-load and the pin load on the net. If the load is high, then check to see if it correlates with the fanout. For example, if as the fanout increases, the load value increases, then that would point toward the wire-load model. If the load is independent of the fanout, then it could be a bad `set_load` constraint or a mis-characterized value in the `.lib` file.
  - Are there any large transition times in the *slew* column?

    The *slew* column reflects how fast a signal is transitioning and is influenced by the capacitive load and the drive strength of the cell. It is used by the delay models to look up the delay for a given cell. Large transition times can be caused by:

    - A bad `input_transition` constraint if the large slew is at the beginning of the path
    - A bad cell description, which would be found in the `.lib` file
    - A large capacitance due to the wire-load model
    - A bad input capacitance of a pin on the next cell

- Sometimes cells are not characterized correctly. Hard macros and custom IPs, such as RAM cells do not always go through the same inspection process as technology libraries.

Often the `.lib` file provided was not characterized under the same operating conditions as the rest of the design. Check the `.lib` file if the delay is unexpectedly large.

Finding Paths Crossing Hierarchical Boundaries

Look at the *pin* column in the timing report, which lists every instance and pin name in the critical path. You can tell when a hierarchical crossing occurs by looking at the instance names. A hierarchical boundary uses indentation to represent the hierarchy and does not list any values in the other columns of the timing report.

- Are there too many hierarchical boundary crossings in the critical path?
  - Too much hierarchy can hinder optimization. The boundaries have to be maintained, which prevents sharing and moving logic together in order to reduce it. Consider ungrouping sections of the design that have too much hierarchy.
  - The other problem with too much hierarchy has to do with the wire-load models used for each level of hierarchy.

➤ Use the `report area` command to get a listing of every wire-load model used for each level of hierarchy in the design to see if the wire-load model selection is good enough, or if it requires some manual intervention, such as using the `force_wireload` command.

- Are there a lot of arithmetic components in the critical path?

  Do you see a lot of adders and other complex arithmetic objects in the critical path? Is the micro architecture really the best for this technology (speed)? Are the arithmetic components the fastest available?

➤ Use the `report datapath` command to see what type of implementations were used.

You may have to override the speed grade of the implementations manually if RTL Compiler is not giving the best results.

- Are there any really big arithmetic components in the critical path?

  Are you using “timing driven” Carry Save Arithmetic (CSA) optimization?

➤ Set CSA optimization using the `synthesize -to_gen -effort high` command.

Using a `medium` effort with the `synthesize -to_gen` command uses CSA, but it is not “timing driven.”

- Does the critical path have a multiplexor followed by an arithmetic resource?

  Speculation reverses the effects of resource sharing between multiplexors and datapath resources, such as adders if the critical path comes in through the select lines (the control lines) of the mux. Use the `synthesize -to_gen -effort high` command to

enable speculation. Use this command if the critical path has both a mux followed by an arithmetic resource.

Analyzing the Critical Path Type: I2O, I2C, C2C, C2O

Are the paths inputs to outputs (I20), or inputs to registers(I2C), or registers-to-registers (C2C), or registers to outputs (C2O)?

- Register-to-register paths should always meet timing, assuming the design was architected correctly, because register-to-register paths only have the launch clock and the capture clock to define their timing requirements. Assuming the clocks are defined correctly, if there is a problem meeting timing and all optimization tricks have been exhausted, then you will have to re-architect the critical path, which means rewriting the RTL.
- Input to register or register to output paths use the most constraints to properly identify the context of the boundary conditions and quite often there are mistakes in the constraint values, such as too high of an input capacitance, a larger than necessary input external delay, a slow transition time, a large output loading, a high external fanout, or a larger than necessary external delay on outputs, and so on. Review the constraints set on the I/O ports and make sure they are reasonable.
- Input to output paths should never exist at the chip level because that would represent a signal that is never registered within the design. However, sometimes they do exist by accident or by design and the constraints create a situation that can never meet timing. Check the timing constraints to make sure they are not overly pessimistic.

Analyzing Ideal Nets in the Path

Ideal nets are designated with an "`i`" next to the pin name in the *pin* column of the timing report. RTL Compiler recognizes clock nets and asynchronous set and reset nets in a design and treats them as "ideal," which means that RTL Compilerdoes not perform any optimization and buffering on the nets. Thus, it is easy for an ideal net to have a high fanout and a high capacitance, but show a zero transition, and consequently, not be considered a critical path. However, when you use the clock or reset nets mixed in with the data signals, the ideal nets suddenly become critical paths. Any logic on an ideal net will not be properly optimized and that can cause all sorts of problems. The best design practice is to isolate clocks and asynchronous pins from the data pins.

Displaying Net Names in a Timing Report

To display the net names in the *instance* or *type* columns in a timing report, copy the current version of the `::report::timing::report_pin_name` and modify it to return the pin name and the connected net name. Other net information, such as fanout and load are reported by default.

Reporting timing_model Instances in a Gate Report

To list an instance that is a `timing_model` when using the `report gates` command during the post-synthesis stage, use the following script.

```
proc find_timing_module {} {
    set count 0
    set list_instance [find /des*/* -instance *]
    foreach index $list_instance {
        set model [get_attr timing_model $index]
        if {$model == true} {
            puts $index
            incr count
        }
    }
    puts "Total number of instances with attribute timing_model: $count"
}
```

Finding all the SDC Commands Used with the dc:: Prefix

To list all the SDC commands that you can use with the `dc::` prefix, use the following procedure:

```
foreach cmd [lsort [info commands dc::*]] { puts $cmd }
```

A

SDC Commands

Introduction

Currently this appendix describes the new commands added in SDC 1.7 and some other frequently used SDC commands.

SDC commands can be specified

- in an SDC file and read in with the `read_sdc` command
- at the RTL Compiler prompt by adding the "`dc::`" prefix before the SDC command.

In the following sections, the title reflects how to use the command interactively in the tool, while the syntax shows the command name as used in the SDC constraints file.

dc::all_clocks

```
all_clocks
```

Returns all clocks in the design when you do not perform multi-mode timing analysis.

When performing multi-mode timing analysis, the command returns all clocks that apply to the current mode.

- When you use this constraint in an SDC file, the constraint will be applied to the mode specified with the `-mode` option of the `read_sdc` command that reads in the file.
- When you want to use this constraint at the RTL Compiler prompt, make sure that the mode to which this constraint needs to be applied was previously specified using the dc::set_mode command.

dc::all_inputs

```
all_inputs
     [ -clock clock_list [-edge_triggered | -level_sensitive]
     | -edge_triggered
     | -level_sensitive
     | -no_clocks]
```

Returns all the input ports in the design. Using the command options you can filter the ports.

Options and Arguments

`-clock` *`clock_list`*	Returns all input ports that have an input delay relative to the specified clock(s).
`-edge_triggered`	Returns all input ports that have a non level-sensitive input delay with respect to the specified clocks. If the `-clock` option is omitted, it returns all input ports that have an edge-triggered input delay with respect to any clock in the design.
`-level_sensitive`	Returns all input ports that have a level-sensitive input delay with respect to the specified clocks. If the `-clock` option is omitted, it returns all input ports that have a level-sensitive input delay with respect to any clock in the design.
`-no_clocks`	Returns all input ports that do not have a clock definition.

dc::all_instances

```
all_instancess
     {design | libcell | subdesign}
```

Returns a list of instances available in the design, or all instances of the specified libcell or subdesign.

Options and Arguments

design	Returns the instances of the specified design. You cannot use this command at the top level of the design. You can use it for hierarchical modules below the top design.
libcell	Returns the instances of the specified library cell.
subdesign	Returns the instances of the specified subdesign.

Examples

- The following command returns the instances of design `m3`:

```
rc:/> ls subdesigns
/designs/m1/subdesigns:
./          m2/         m2myclk/    m3/         m3_0/       m3_0_0/     m4/
m5/         m5_bbox/    m6/
rc:/> dc::all_instances [dc::get_design m3]
/designs/m1/instances_hier/m2/instances_hier/m3
```

- The following command returns the instances of libcell `CLKINVX2`

```
rc:/> dc::all_instances [dc::get_lib_cell CLKINVX2]
/designs/m1/instances_comb/g127 /designs/m1/instances_comb/g132 /designs/m1/
instances_hier/m2/instances_comb/g150 /designs/m1/instances_hier/m2/
instances_hier/m3/instances_hier/m4/instances_hier/m5/instances_comb/g58


rc:/>
```

- The following command returns the instances of subdesign `m4`.

```
rc:/> dc::all_instances /designs/m1/subdesigns/m4
/designs/m1/instances_hier/m2/instances_hier/m3/instances_hier/m4
```

dc::all_outputs

```
all_outputs
     [-clock clock_list]
     [-edge_triggered | -level_sensitive]
```

Returns all the output ports in the design.

Options and Arguments

`-clock` *`clock_list`*	Not supported by RTL Compiler.
`-edge_triggered`	Not supported by RTL Compiler.
`-level_sensitive`	Not supported by RTL Compiler.

dc::all_registers

```
all_registers
     [-cells]
     [-clock clock...]
     [-fall_clock clock...] [-rise_clock clock...]
     [-clock_domains clock_domain...]
     [-data_pins] [-clock_pins] [-slave_clock_pins]
     [-async_pins] [-output_pins]
     [-edge_triggered] [-level_sensitive]
     [-master_slave] [-inverted_output]
     [-exclude instance...]
     [-no_hierarchy]
```

Returns all the sequential elements in the design. Using the command options you can filter the elements that are returned.

Note: This command was added in SDC 1.7.

Options and Arguments

`-async_pins` Not supported by RTL Compiler.

`-cells` Not supported by RTL Compiler.

`-clock` *clock_list*

Returns only those sequential elements clocked by the specified clocks.

`-clock_domains` *clock_domain_list*

Returns only those sequential elements that belong to the specified clock domains.

`-clock_pins` Returns the clock pins of all sequential elements in the design.

`-data_pins` Returns the data pins of all sequential elements in the design.

`-edge_triggered` Returns all sequential elements of type flop in the design.

`-exclude` *instance_list*

Excludes the specified instances from the list of sequential elements to be returned.

`-fall_clock` *clock_list*

Returns only those sequential elements clocked by the falling edge of the specified clocks.

`-inverted_output`	Returns all sequential elements in the design that have an inverted output.
`-level_sensitive`	Returns all sequential elements of type latch in the design.
`-master_slave`	Returns all sequential elements in the design that are master-slave flops.
`-no_hierarchy`	Returns only those sequential elements at the top-level of the design.
`-output_pins`	Returns the output pins of all sequential elements in the design.
`-rise_clock` *`clock_list`*	Returns only those sequential elements clocked by the rising edge of the specified clocks.
`-slave_clock_pins`	Returns the clock pins of the slave flops in the design.

dc::create_clock

```
create_clock
     [-add]
     [-name clock] [-domain clock_domain]
     -period float
     [-waveform float]
     [-apply_inverted {port|pin}]
     [port|pin]
```

Creates a clock object and defines its waveform. If you do not specify any sources, but you specify the `-name` option, a virtual clock is created.

Options and Arguments

`-add` — Allows to specify multiple clocks on the same source for simultaneous analysis with different clock waveforms.

If you omit this option and a clock was already defined on a pin or port, this definition would overwrite the previous one.

`-apply_inverted {port | pin}`

Specifies inverted sources of the clock.

Note: This is non-SDC option.

`-domain clock_domain`

Specifies the clock domain to which the clock belongs.

Default: `domain_1`

Note: This is non-SDC option.

`-name clock` — Specifies the name of the clock.

If you omit this option, the clock gets the same name as the first clock source (inverted or non-inverted). If no sources were specified for this clock, the clock name defaults to `create_clock_n`, where `n` is 1,2, and so on.

`-period float` — Specifies the length of the clock period.

`port | pin` — Specifies the non-inverted sources of the clock.

`-waveform` *`float...`* Specifies the rise and fall edge times of the clock waveform over one clock period. The first value corresponds to the first rising transition after time zero. The numbers should represent one full clock period. If you omit this option, a default waveform is assumed: the leading edge occurs at 0 and the trailing edge occurs at the midpoint of the period, such that a symmetric clock is generated.

Example

The following command creates clock clk1 on port clkA with a period of 5 time units. The clock has a rising edge at 0 time units and a falling edge at 2.5 time units.

```
dc::create_clock [dc::get_ports clkA]  -name clk1  -period 5  -waveform {0 2.5}
```

dc::create_generated_clock

```
create_generated_clock
     [-add] [-combinational]
     [-name clock] [-domain domain]
     -source [-master_clock clock]
     [-divide_by integer
     |-multiply_by integer [-duty_cycle float]
     |-edges integer [integer] [integer]
       [-edge_shift float [float] [float]]]
     [-apply_inverted {port|pin}...]
     {port|pin}...
```

Creates a new clock signal from the clock waveform of a given pin in the design.

Options and Arguments

`-add` Allows to specify multiple generated clocks on the same source when multiple clocks must fan into the source pin.

If you omit this option and a clock was already defined on a pin or port, this definition would overwrite the previous one.

`-apply_inverted {`*`port`*`|`*`pin`*`}`

Specifies the pin or port on which to apply the inverted clock waveform.

Note: This is non-SDC option.

`-combinational` Not supported by RTL Compiler.

Note: RTL Compiler writes out the option when you specify the `write_sdc` command.

`-divide_by` *`integer`* Determines the frequency of the new clock by dividing the frequency of the source clock by this factor.

If the frequency is divided by a certain factor, the clock period is multiplied by that same factor.

If the factor is a power of 2 (2,4,8... except 1), the rising edges of the master clock are used to determine the edges of the generated clock, otherwise the edges are scaled from the master clock edges.

`-domain` *domain*
: Specifies the clock domain to which the generated clock belongs.

  Default: `domain_1`

  Note: This is non-SDC option.

`-duty_cycle` *float*
: Specifies the duty cycle (high pulse width) as a percentage of the clock period. Specify a number between 0 and 100.

`-edge_shift` *float* [*float*] [*float*]
: Specifies how much each edge specified with the `-edges` option should be shifted.

`-edges` *integer* [*integer*] [*integer*]
: Selects a list of edges from the source clock that form the edges of the derived clock. Currently, three edge numbers are allowed.

`-invert`
: Inverts the resulting waveform of the generated clock.

`-master_clock` *clock*
: Derives the generated (target) clock from the specified clock.

  Use this option if multiple signals arrive at the source pin and you want to indicate a particular clock from which the target clock must be generated.

`-multiply_by` *integer*
: Determines the frequency of the new clock by multiplying the frequency of the source clock with this factor.

  If the frequency is multiplied with a certain factor, the clock period is divided by that same factor.

`-name` *clock*
: Specifies the name of the generated clock.

port | *pin*
: Specifies the pins and ports to which the generated clock must be applied

`-source`
: Specifies the name of the pin from which the clock must be derived.

Examples

- The following command derives target clock `gen_clk1` from the clock at pin `CK` on instance `inst1` and divides the frequency by 2. The clock i9s applied to pin `Q` of instance `inst1`.

```
dc::create_generated_clock -name gen_clk1 -source [dc::get_pins inst1/CK] \
-divide_by 2 [dc::get_pins inst1/Q]
```

- The following command derives target clock `gen_clk1` from the clock at pin `CK` on instance `inst1` by specifying edges 1, 5 and 7. This implies a `divide_by 3` and a duty cycle of 66.66%.

```
dc::create_generated_clock name gen_clk1 source [dc::get_pins inst1/CK]
-edges {1 5 7} [dc::get_pins inst1/Q]
```

dc::current_design

```
current_design
     [design]
```

Changes the current directory in the design hierarchy to the specified design and returns the path to the design.

If no design is specified, the command returns the top-level design.

Options and Arguments

design	Specifies the name of the top-level design.

Examples

The following command changes the current directory in the design hierarchy to the specified design `top`.

```
rc:/> dc::current_design top
/designs/top
rc:/designs/top>
```

Assume the top design is called `test`. The following command returns the path to the top-level design.

```
rc:/> dc::current_design
/designs/test
```

dc::current_instance

```
current_instance
     [instance]
```

Changes the current directory in the design hierarchy to the specified instance and returns the path to the instance.

If no instance is specified, the command returns the path to the top-level design.

Options and Arguments

instance	Specifies the name of the instance.

Example

The following command changes the current directory in the design hierarchy to the specified instance .

```
rc:/> dc::current_instance instA
/designs/top/instances_seq/instA
rc:/designs/top/instances_seq/instA>
```

The following command returns the path to the top-level design.

```
rc:/designs/top/instances_seq> dc::current_instance
/designs/top
```

dc::filter_collection

```
filter_connection
     {port | pin | instance}... filter_expression
```

Returns a collection of objects that match the specified filter expression.

Options and Arguments

filter_expression	Corresponds to a filter expression supported for the specified objects.
instance	Specifies a list of instances or specifies a command that returns a list of instances.
pin	Specifies a list of pins or specifies a command that returns a list of pins.
port	Specifies a list of ports or specifies a command that returns a list of ports.

Example

The following command returns all instances whose libcell matches `inv1`.

```
rc:/> dc::filter_collection [dc::get_cells *] "ref_name==inv1"
/designs/top_most/instances_comb/inst2 /designs/top_most/instances_comb/inst5
```

dc::get_cell

```
{get_cell | get_cells} [-filter string]
     [ -hierarchical | -of_objects list
     | pattern [-hsc string] [-regexp [-nocase]] ]
```

Returns the full path to a list of instances. Using the command options you can filter the elements that are returned.

Note: This command was added in SDC 1.5.

Options and Arguments

`-filter` *string*

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "area{==|!=|<=|>=|<|>}`*float*`"`

`area` checks the area of the instance.

`-filter "{base_name|name}{==|=~|!=|!~}`*name*`"`

`base_name` or `name` checks the base name of the instance.

`-filter "{full_name|hierarchical_name}{==|=~|!=|!~}`*vname*`"`

`full_name` or `hierarchical_name` checks the vdir name of the instance.

`-filter "is_combinational{==|!=}{true|false}"`

`is_combinational` checks if the instance is combinational.

`-filter "is_dont_touch{==|!=}{true|false}"`

`is_dont_touch` checks if the instance is preserved.

`-filter "is_hierarchical{==|!=}{true|false}"`

`is_hierarchical` checks if the instance is hierarchical.

`-filter "is_sequential{==|!=}{true|false}"`

`is_sequential` checks if the instance is sequential.

`-filter "object_type{==|=~|!=|!~}instance"`

`object_type` checks if the object is an instance.

	`-filter "pin_count{==|!=|<=|>=|<|>}`*`integer`*`"`
	`pin_count` checks the pin count of the instance.
	`-filter "ref_lib_cell_name{==|=~|!=|!~}`*`libcell`*`"`
	`ref_lib_cell_name` checks the libcell name of the instance.
	`-filter "ref_name{==|=~|!=|!~}`*`libcell`*`"`
	`ref_name` checks the libcell name of the instance.
	Note: Any filter option can be specified as `option` or `@option` in the expression.
	You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.
	Note: This is a non-SDC option.

`-hierarchical`	Selects all instances in the hierarchy below the current level that match the specified pattern.
`-hsc` *`string`*	Specifies the hierarchical delimiter for patterns. For example, if the hierarchical delimiter is @, the `sub/I1@I2` pattern matches instance `I2` within the hierarchy `sub/I1`.
`-nocase`	Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.
`-of_objects` *`list`*	Specifies to return instances with the specified pins or connected to the specified nets.
`pattern`	Specifies a pattern of cell names or regular expressions. *Default*: *
`-regexp`	Specifies that the pattern is a regular expression.

Examples

- The following command returns those instances which have been mapped to the libcell `flopped`.

```
rc:/> dc::get_cells -filter "ref_name==fflopd"

/designs/top/instances_seq/inst1 /designs/top/instances_seq/instf
```

- The following command returns those instances that are named `inst1`.

```
rc:/> dc::get_cells -filter "full_name==inst1"

/designs/top/instances_seq/inst1
```

- The following command returns those instances connected to the net `clk`.

```
rc:/> dc::get_cells -of_objects /designs/top/nets/clk
/designs/top/instances_seq/inst3
```

- The following command returns all instances that match the regular expression pattern ^inst1.*$

```
rc:/> dc::get_cells -regexp ^inst1.*$
/designs/top/instances_seq/inst1 /designs/top/instances_comb/inst1_0
/designs/top/instances_comb/inst1_1
```

- The following command uses a complex expression to return the path to
  - the list of all instances that either have libcell name inv1 or instance name inst_mux,
  - the list of instances that either have libcell `MX2X1` or `fflopd`

```
rc:/> dc::get_cells -filter "((ref_name ==inv1 || full_name==inst_mux) &&
(ref_name==MX2X1)) || (ref_name==fflopd))
```

dc::get_clock

```
{get_clock | get_clocks}
     [-filter string]
     [-hsc string] [-regexp] [-nocase] [pattern]
```

Returns the full path to a list of timing clocks that match the pattern.

Options and Arguments

`-filter string`

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "{base_name|name}{==|=~|!=|!~}name"`

`base_name` or `name` checks the base name of the clock.

`-filter "full_name{==|=~|!=|!~}vname"`

`full_name` checks the vdir name of the clock

`-filter "has_detailed_parasitics{==|!=}{true|false}"`

`has_detailed_parasitics` checks if the `phys_capacitance` attribute is set on the clock.

`-filter "object_type{==|=~|!=|!~}clock"`

`object_type` checks if the object is a clock.

`-filter "period{==|!=|<=|>=|<|>}period"`

`period` checks the period of the clock.

`-filter "sources{==|=~|!=|!~}clock_sources"`

`sources` checks the inverted and non-inverted sources of the clock

Note: Any filter option can be specified as `option` or `@option` in the expression.

You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.

Note: This is a non-SDC option.

`-hsc string`

Not supported by RTL Compiler.

`-nocase`

Not supported by RTL Compiler.

pattern	Specifies a pattern of clock names. *Default*: *
-regexpr	Not supported by RTL Compiler.

Examples

- The following command returns the path to clock `test_clock`.

```
rc:/> dc::get_clocks test_clock
/designs/top_most/timing/clock_domains/domain_1/test_clock
```

- The following command returns the path to all clocks starting with `test`:

```
rc:/> dc::get_clocks test*
/designs/top_most/timing/clock_domains/domain_1/test_clock
```

- The following command returns the path to a clock whose name is `c1`.

```
rc:/> dc::get_clocks -filter "full_name==c1"
/designs/top/timing/clock_domains/domain_1/c1
```

dc::get_design

```
{get_design | get_designs}
     pattern
```

Returns the full path to any design or subdesign matching the pattern.

Options and Arguments

`-filter` *string*

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "{base_name|name}{==|=~|!=|!~}`*name*"

`base_name` or `name` checks the name of the design.

`-filter "is_dont_touch{==|!=}{true|false}"`

`is_dont_touch` checks if the design is preserved.

`-filter "max_capacitance{==|!=|<=|>=|<|>}`*float*"

`max_capacitance` checks the `max_capacitance` attribute set on the design

`-filter "max_fanout{==|!=|<=|>=|<|>}`*integer*"

`max_fanout` checks the `max_fanout` attribute set on the design

`-filter "max_transition{==|!=|<=|>=|<|>}`*float*"

`max_transition` checks the `max_transition` attribute set on the design

`-filter "object_type{==|=~|!=|!~}design"`

`object_type` checks if the object is a design.

Note: Any filter option can be specified as `option` or `@option` in the expression.

You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.

Note: This is a non-SDC option.

pattern

Specifies a pattern of design or subdesign names.

Note: The command only looks for subdesigns if no design with the specified pattern is found.

Examples

- The following command returns the path to design `top_most`.

```
rc:/> dc::get_designs top_most*
/designs/top_most
```

- The following command returns the path to subdesign `top_most_1`:

```
rc:/> dc::get_designs top_most_*
/designs/top_most/subdesigns/top_most_1
```

dc::get_generated_clocks

```
get_generated_clocks
     [-quiet] [-regexp [-nocase]]
     [-filter string] [pattern]
```

Returns the full path to all the generated clocks that match the pattern. Using the command options you can filter the clocks that are returned.

Options and Arguments

`-filter` *string*

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "{base_name|name}{==|=~|!=|!~}`*name*"

`base_name` or `name` checks the base name of the generated clock.

`-filter "full_name{==|=~|!=|!~}`*vname*"

`full_name` checks the vdir name of the generated clock

`-filter "object_type{==|=~|!=|!~}clock"`

`object_type` checks if the object is a clock.

`-filter "period{==|!=|<=|>=|<|>}`*period*"

`period` checks the period of the generated clock.

`-filter "sources{==|=~|!=|!~}`*clock_sources*"

`sources` checks the inverted and non-inverted sources of the generated clock

Note: Any filter option can be specified as `option` or `@option` in the expression.

You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.

`-nocase`

Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.

pattern

Specifies the pattern for generated clock names to be matched.

If no pattern is specified, all generated clocks are considered.

-quiet	Suppresses all the generated messages.
-regexp	Specifies that the specified pattern is a regular expression.

Examples

Assume you defined the following clocks:

```
define_clock -name clk -period 10000 -domain domain_1 \
    [find /des*/*/ports_in -port Clk]
define_clock -name clk1 -period 20000 -domain domain_2 \
[find /des*/*/ports_in -port Clk1]
```

Also assume you read in the following SDC file:

```
create_generated_clock -name gen_clk -source [find /des*/*/ports_in -port Clk]
-divide_by 2 [find /des*/*/*seq/f6/ -pin CK]
create_generated_clock -name gen_clk1 -source [find /des*/*/ports_in -port Clk1]
-divide_by 2 [find /des*/*/*seq/f12/ -pin CK]
create_generated_clock -name gen_clk2 -source [find /des*/*/ports_in -port Clk1]
-divide_by 2 [find /des*/*/*seq/f13/ -pin CK]
```

- The following command returns the full path to all generated clocks:

  ```
  rc:/> dc::get_generated_clocks
  /designs/test/timing/clock_domains/domain_1/gen_clk /designs/test/timing/
  clock_domains/domain_1/gen_clk1 /designs/test/timing/clock_domains/domain_1/
  gen_clk2
  ```

- The following command returns the path to the generated clock with period 20000.

  ```
  rc:/> dc::get_generated_clocks -filter "period==20000.0"
  /designs/test/timing/clock_domains/domain_1/gen_clk
  ```

- The following command returns the path to the generated clocks whose name is not `gen_clk1`.

  ```
  rc:/> dc::get_generated_clocks -filter "name!=gen_clk1"
  /designs/test/timing/clock_domains/domain_1/gen_clk /designs/test/timing/
  clock_domains/domain_1/gen_clk2
  ```

- The following command returns the path to the generated clocks whose name matches `gen_clk1` or `gen_clk2`.

  ```
  rc:/> dc::get_generated_clocks -regexp -nocase GEN_CLK1|GEN_CLK2
  /designs/test/timing/clock_domains/domain_1/gen_clk1 /designs/test/timing/
  clock_domains/domain_1/gen_clk2
  ```

Related Information

Related command: dc::create_generated_clock on page 222

dc::get_lib

```
{get_lib | get_libs}
     [-filter string]
     [-regexp] [-nocase] pattern
```

Returns the full path to any library that matches the pattern. Using the command options you can filter the elements that are returned.

Options and Arguments

`-filter` *string*	Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:
	`-filter "{base_name|name}{==|=~|!=|!~}`*name*`"`
	`base_name` or `name` checks the base name of the library.
	`-filter "object_type{==|=~|!=|!~}library"`
	`object_type` checks if the object is a library.
	Note: Any filter option can be specified as `option` or `@option` in the expression.
	You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.
	Note: This is a non-SDC option.

`-nocase`	Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.
pattern	Specifies a pattern of library names or regular expressions.
`-regexp`	Specifies that the pattern is a regular expression.

Examples

- The following command just specifies a pattern to find the path to the `tutorial` library.

```
rc:/> dc::get_libs tut*
/libraries/tutorial
```

- The following command specifies a pattern with regular expression to find the path to the `tutorial` library.

```
rc:/> dc::get_libs -regexp tut.*
/libraries/tutorial
```

- The following command specifies a pattern that is case insensitive to find the path to the `tutorial` library.

```
rc:/> dc::get_libs -regexp -nocase TUT.*
/libraries/tutorial
```

dc::get_lib_cell

```
{get_lib_cell | get_lib_cells}
     [-filter string]
     [-hsc string] [-regexp] [-nocase] [pattern]
```

Returns the full path to any library cell that matches the pattern. Using the command options you can filter the elements that are returned.

Options and Arguments

`-filter` *string*

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "area{==|!=|<=|>=|<|>}`*float*`"`

`area` checks the area of the libcell

`-filter "{base_name|name}{==|=~|!=|!~}`*name*`"`

`base_name` or `name` checks the base name of the libcell.

`-filter "is_combinational{==|!=}{true|false}"`

`is_combinational` checks if the libcell is combinational

`-filter "is_dont_touch{==|!=}{true|false}"`

`is_dont_touch` checks if the libcell is preserved.

`-filter "is_dont_use{==|!=}{true|false}"`

`is_dont_use` checks if the libcell must be avoided.

`-filter "is_sequential{==|!=}{true|false}"`

`is_sequential` checks if the libcell is sequential.

`-filter "is_tristate{==|!=}{true|false}"`

`is_tristate` checks if the libcell is tristate.

`-filter "object_type{==|=~|!=|!~}libcell"`

`object_type` checks if the object is a libcell.

Note: Any filter option can be specified as `option` or `@option` in the expression.

	You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions. **Note:** This is a non-SDC option.
`-hsc` *`string`*	Specifies the hierarchical delimiter for patterns.
`-nocase`	Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.
`pattern`	Specifies a pattern of libcell names or regular expressions.
`-regexp`	Specifies that the pattern is a regular expression.

Examples

- The following command returns all libcells whose name start with `nan`..

```
rc:/> dc::get_lib_cells nan*
/libraries/tutorial/libcells/nand2
```

- The following command returns all libcells whose name start with `nan` and that are part of the library whose name starts with `tut`.

```
rc:/> dc::get_lib_cells tut*/nan*
/libraries/tutorial/libcells/nand2
```

- The following command returns all libcells whose name start with `XOR3XL`.

```
rc:/> dc::get_lib_cells XOR3XL*
/libraries/library_domains/set1/typical/libcells/XOR3XL
/libraries/library_domains/set2/typical/libcells/XOR3XL
```

  In this case, two library sets were read in which each contained the `typical` library with their version of the `XOR3XL` cell.

- The following command returns all libcells whose name start with `XOR3XL` and that belong to the typical library.

```
rc:/> dc::get_lib_cells typical@XOR3XL -hsc @
/libraries/library_domains/set1/typical/libcells/XOR3XL
/libraries/library_domains/set2/typical/libcells/XOR3XL
```

dc::get_lib_pin

```
{get_lib_pin | get_lib_pins}
     [-filter string]
     [-hsc string] [-regexp] [-nocase] [pattern]
```

Returns the full path to any libcell pin that matches the pattern. Using the command options you can filter the elements that are returned.

Options and Arguments

`-filter string`

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "{base_name|name}{==|=~|!=|!~}name"`

`base_name` or `name` checks the base name of the libcell pin.

`-filter "capacitance{==|!=|<=|>=|<|>}float"`

`capacitance` checks the average capacitance (rise+fall/2) of the libcell pin

`-filter "direction=={in|out|inout}"`

`direction` checks the direction of the libcell pin.

`-filter "is_async_pin{==|!=}{true|false}"`

`is_async_pin` checks if the libcell pin is an asynchronous pin

`-filter "is_clear_pin{==|!=}{true|false}"`

`is_clear_pin` checks if the libcell pin is a clear pin

`-filter "is_clock_pin{==|!=}{true|false}"`

`is_clock_pin` checks if the libcell pin is a clock pin

`-filter "max_capacitance{==|!=|<=|>=|<|>}float"`

`max_capacitance` checks the `max_capacitance` attribute set on the libcell pin

`-filter "max_fanout{==|!=|<=|>=|<|>}integer"`

`max_fanout` checks the `max_fanout` attribute set on the libcell pin

	`-filter "max_transition{==|!=|<=|>=|<|>}`*`float`*`"`
	`max_transition` checks the `max_transition` attribute set on the libcell pin
	`-filter "object_type{==|=~|!=|!~}libpin"`
	`object_type` checks if the object is a libpin.
	Note: Any filter option can be specified as `option` or `@option` in the expression.
	You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.
	Note: This is a non-SDC option.

`-hsc` *`string`*	Specifies the hierarchical delimiter for patterns.
`-nocase`	Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.
`pattern`	Specifies a pattern of libpin names or regular expressions.
`-regexp`	Specifies that the pattern is a regular expression.

Examples

In the following examples, two library sets were read in. Each library set contained the `typical` library with its version of the `XOR3XL` cell. The examples show different ways to get the same information.

- The following command returns all libpins whose name start with `A` and that belong to a libcell whose name starts with `XOR3XL`.

```
rc:/> dc::get_lib_pins XOR3XL*/A*
/libraries/library_domains/set1/typical/libcells/XOR3XL/A
/libraries/library_domains/set2/typical/libcells/XOR3XL/A
```

- The following command uses the hierarchy delimiter to request to return all libpins whose name start with A and that belong to a libcell whose name starts with `XOR3XL`.

```
rc:/> dc::get_lib_pins XOR3XL*@A* -hsc @
/libraries/library_domains/set1/typical/libcells/XOR3XL/A
/libraries/library_domains/set2/typical/libcells/XOR3XL/A
```

- The following command returns all libpins whose name start with `A` and that belong to a libcell whose name start with `xor3xl` but indicates that the pattern is not case sensitive.

```
rc:/> get_lib_pins -regexp -nocase xor3xl/A.*
/libraries/library_domains/set1/typical/libcells/XOR3XL/A
/libraries/library_domains/set2/typical/libcells/XOR3XL/A
```

dc::get_lib_timing_arcs

```
get_lib_timing_arcs
     [-to libpin...] [-from libpin...]
     [-of_objects libcell...] [-filter string]
```

Returns all timing arcs that match the specified pattern. Using the command options you can filter the timing arcs that are returned.

Options and Arguments

`-filter string`

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "{base_name|name}{==|=~|!=|!~}name"`

`base_name` or `name` checks the name of the timing arc.

`-filter "from_lib_pin{==|=~|!=|!~}pin_name"`

`from_lib_pin` checks the name of the from pin of the arc.

`-filter "object_type{==|=~|!=|!~}libarc"`

`object_type` checks if the object is a timing arc.

`-filter "sdf_cond{==|=~|!=|!~}sdf_cond"`

`sdf_cond` checks the SDF condition of the arc.

`-filter "timing_type{==|=~|!=|!~}timing_type"`

`timing_type` checks the timing type (positive_unate, negative_unate, non_unate) of the arc.

`-filter "when{==|=~|!=|!~}when_cond"`

`when` checks the when condition of the arc.

Note: Any filter option can be specified as `option` or `@option` in the expression.

You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.

Note: This is a non-SDC option.

`-from libpins`

Returns all outgoing timing arcs for the specified library cell pin(s).

`-of_objects` *cells*

Returns all timing arcs for the specified library cells.

`-to` *libpins* Returns all incoming timing arcs for the specified library cell pin(s).

Examples

- The following command returns the timing arcs to pin `Y` of cells `nand2` and `nor2`.

```
rc:/> dc::get_lib_timing_arcs -to "nand2/Y nor2/Y"
/libraries/tutorial/libcells/nand2/Y/inarcs/A_Y_n60 /libraries/tutorial/
libcells/nand2/Y/inarcs/B_Y_n60 /libraries/tutorial/libcells/nor2/Y/inarcs/
A_Y_n60 /libraries/tutorial/libcells/nor2/Y/inarcs/B_Y_n60
```

- The following command retruns the timing arcs between pins `A` and `Y` of library cell `nand2`.

```
rc:/> dc::get_lib_timing_arcs -to nand2/Y -from nand2/A
/libraries/tutorial/libcells/nand2/Y/inarcs/A_Y_n60
```

- The following command returns all timing arcs for libcell `fflopd_ckn`.

```
dc::get_lib_timing_arcs -of fflopd_ckn
/libraries/tutorial/libcells/fflopd_ckn/D/inarcs/CKN_D_S50 /libraries/
tutorial/libcells/fflopd_ckn/D/inarcs/CKN_D_H50 /libraries/tutorial/libcells/
fflopd_ckn/Q/inarcs/CKN_Q_c50
```

- The following command filters based on the value of the `timing_type`.

```
rc:/> dc::get_lib_timing_arcs -from nand2/A -to nand2/Y \
-filter "timing_type==negative_unate"
/libraries/tutorial/libcells/nand2/Y/inarcs/A_Y_n60
```

dc::get_net

```
{get_net | get_nets}
     [-filter string]
     [-hierarchical] [-hsc string] [-regexp]
     [-nocase] [-of_objects string | pattern]
```

Returns the full path to any net that matches the pattern. Using the command options you can filter the elements that are returned.

Options and Arguments

`-filter` *string*

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "area{==|!=|<=|>=|<|>}`*float*`"`

`area` checks the enclosed area of the net

`-filter "{base_name|name}{==|=~|!=|!~}`*name*`"`

`base_name` or `name` checks the base name of the net.

`-filter "full_name{==|=~|!=|!~}`*vname*`"`

`full_name` checks the vdir name of the net

`-filter "is_dont_touch{==|!=}{true|false}"`

`is_dont_touch` checks if the net is preserved.

`-filter "object_type{==|=~|!=|!~}net"`

`object_type` checks if the object is a net.

Note: Any filter option can be specified as `option` or `@option` in the expression.

You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.

Note: This is a non-SDC option.

`-hierarchical`

Selects all the nets in the hierarchy below the current level that match the specified pattern.

`-hsc` *string*

Specifies the hierarchical delimiter for patterns.

`-nocase`

Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.

`-of_objects` *`string`*	Specifies to return the nets that are connected to the specified cells, pins, and ports.
`pattern`	Specifies a pattern of net names or regular expressions. *Default*: *
`-regexp`	Specifies that the pattern is a regular expression.

Examples

- The following command returns net `in` that is in the `inst9` hierarchy.

```
rc:/> dc::get_nets -hsc @ inst9@in
/designs/top_most/instances_hier/inst9/nets/in
```

- The following command returns all nets whose name start with `c`.

```
rc:/> dc::get_nets c* -filter "object_type==net"
/designs/aco1/nets/ck {/designs/aco1/nets/c[0]} {/designs/aco1/nets/c[1]}
{/designs/aco1/nets/c[2]} {/designs/aco1/nets/c[3]} {/designs/aco1/nets/c[4]}
{/designs/aco1/nets/c[5]}
```

dc::get_object_name

```
get_object_name
     string
```

Returns the specified argument.

Note: This command was added for compatibility while reading in SDC files with non-SDC constraints. Since RTL Compiler does not have a concept of collections, this command just returns the specified argument.

Options and Arguments

string	Specifies the argument to be returned.

Example

```
rc:/> dc::get_object_name /designs/top/instances_hier/abc
/designs/top/instances_hier/abc
```

dc::get_path_groups

```
get_path_groups
     [-quiet] [-regexp [-nocase]]
     [-filter string] [pattern]
```

Returns all the path groups that match the pattern. Using the command options you can filter the path groups that are returned.

Options and Arguments

`-filter` *string*	Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:
	`-filter "{base_name|name}{==|=~|!=|!~}`*name*`"`
	`base_name` or `name` checks the base name of the path group.
	`-filter "object_type{==|=~|!=|!~}path_group"`
	`object_type` checks if the object is an path_group.
	Note: Any filter option can be specified as `option` or `@option` in the expression.
	You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.
	Note: This is a non-SDC option.

`-nocase`	Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.
pattern	Specifies the pattern for path group names to be matched. If no pattern is specified all path groups are considered.
`-quiet`	Suppresses all the generated messages.
`-regexp`	Specifies that the specified pattern is a regular expression.

Examples

- Assuming you had three generated clocks, the tool will implicitly associate path groups with these clocks. The following command returns the full path to the path groups associated with the generated clocks:

```
rc:/designs/test/> dc::get_path_groups
/designs/test/timing/exceptions/path_groups/gen_clk /designs/test/timing/
exceptions/path_groups/gen_clk1 /designs/test/timing/exceptions/path_groups/
gen_clk2
```

- The following command returns the path to a path group whose pattern matches `gen_clk1` or `gen_clk2`.

```
rc:/designs/test/> dc::get_path_groups -regexp -nocase gen_clk1|gen_clk2
/designs/test/timing/exceptions/path_groups/gen_clk1 /designs/test/timing/
exceptions/path_groups/gen_clk2
```

- The following command returns the path to a path group whose name matches `gen_clk`.

```
rc:/designs/test/> dc::get_path_groups -filter "name==gen_clk"
/designs/test/timing/exceptions/path_groups/gen_clk
```

Related Information

Related commands: dc::create_generated_clock on page 222

path_group

dc::get_pin

```
{get_pin | get_pins}
     [-filter string] [-hierarchical] [-hsc string]
     [-regexp] [-nocase] {-of_objects string | pattern}
```

Returns the full path to any pin that matches the pattern. Using the command options you can filter the elements that are returned.

Options and Arguments

`-filter` *string*

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "{base_name|name}{==|=~|!=|!~}`*name*`"`

`base_name` or `name` checks the base name of the pin.

`-filter "{direction|pin_direction}=={in|out|inout}"`

`direction` or **`pin_direction`** checks the direction of the port.

`-filter "{full_name|hierarchical_name}{==|=~|!=|!~}`*vname*`"`

`full_name` or `hierarchical_name` checks the vdir name of the pin

`-filter "is_hierarchical{==|!=}{true|false}"`

`is_hierarchical` checks if the pin is hierarchical.

`-filter "fanin{==|=~|!=|!~}`*integer*`"`

`fanin` checks the fanin of the pin

`-filter "fanout{==|=~|!=|!~}`*integer*`"`

`fanout` checks the fanout of the pin

`-filter "is_pin{==|!=}{true|false}"`

`is_pin` checks if the object is a pin.

`-filter "is_port{==|!=}{true|false}"`

`is_port` checks if the object is a port.

`-filter "object_type{==|=~|!=|!~}pin"`

`object_type` checks if the object is a pin.

	Note: Any filter option can be specified as `option` or `@option` in the expression. You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions. **Note:** This is a non-SDC option.
`-hierarchical`	Selects all pins in the hierarchy below the current level that match the specified pattern.
`-hsc` *`string`*	Specifies the hierarchical delimiter for patterns.
`-nocase`	Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.
`-of_objects`	Specifies to return pins that are part of the specified instances or to which the specified nets connect.
`pattern`	Specifies a pattern of cell names or regular expressions. *Default*: *
`-regexp`	Specifies that the pattern is a regular expression.

Examples

- The following command returns the full path to all the pins of instance `g6`.

```
rc:/designs/rct> dc::get_pins -hsc @ g6@*
/designs/rct/instances_comb/g6/pins_in/A /designs/rct/instances_comb/g6/
pins_in/B /designs/rct/instances_comb/g6/pins_out/Y
```

- The following command returns the full path to pin `Y` that is part of instance `inst1`, which in turn belongs to the hierarchy of instance `inst9`.

```
rc:/> dc::get_pins -hsc @ inst9@inst1@Y
/designs/top_most/instances_hier/inst9/instances_comb/inst1/pins_out/Y
```

- The following command returns the full path to the output pin of instance `g6`.

```
rc:/designs/rct> dc::get_pins -hsc @ g6@* -filter direction==out
/designs/rct/instances_comb/g6/pins_out/Y
```

- The following command returns the full path to all output pins of the instances whose name start with `g`.

```
rc:/designs/rct> dc::get_pins -filter direction==out -of_objects g*
/designs/rct/instances_comb/g3/pins_out/Y /designs/rct/instances_comb/g5/
pins_out/Y /designs/rct/instances_comb/g4/pins_out/Y /designs/rct/
instances_comb/g6/pins_out/Y
```

- The following command returns the full path to all output pins to which nets `n_1` and `n_3` connect.

```
rc:/designs/rct> dc::get_pins  -filter direction==out -of_objects {n_1 n_3}
/designs/rct/instances_comb/g6/pins_out/Y /designs/rct/instances_comb/
cdn_loop_breaker/pins_out/Y
```

- The following command checks if the pin inst1/A specified in the filter expression exist and then returns the path to the pin:

```
rc:/ dc::get_pins -filter "full_name==inst1/A"
/designs/top/instances_comb/inst1/A
```

- The following commands return nothing:

```
rc:/> dc::get_pins -filter "is_port==true" *
rc:/> dc::get_ports -filter "is_pin==false" *
```

- The following command returns all pins connectd by net `ck`.

```
rc:/> dc::get_pins -filter "object_type==pin" -of_object /designs/aco1/nets/ck
{/designs/aco1/instances_seq/c_reg[3]/pins_in/clk} {/designs/aco1/
instances_seq/c_reg[0]/pins_in/clk} {/designs/aco1/instances_seq/c_reg[4]/
pins_in/clk} {/designs/aco1/instances_seq/c_reg[1]/pins_in/clk} {/designs/
aco1/instances_seq/c_reg[5]/pins_in/clk} {/designs/aco1/instances_seq/
c_reg[2]/pins_in/clk}
```

dc::get_port

```
{get_port | get_ports}
     [-filter string] [-regexp] [-nocase]
     [-of_objects string | pattern]
```

Returns the full path to any port that matches the pattern. Using the command options you can filter the elements that are returned.

Options and Arguments

`-filter` *string*

Filters the result based on the specified expression evaluating to `true`. The following filter expressions are supported:

`-filter "{base_name|name}{==|=~|!=|!~}`*name*`"`

`base_name` or `name` checks the base name of the port.

`-filter "{direction|port_direction}=={in|out|inout}"`

`direction` or **`port_direction`** checks the direction of the port.

`-filter "full_name{==|=~|!=|!~}`*vname_of_port*`"`

`full_name` checks the vdir name of the port

`-filter "is_pin{==|!=}{true|false}"`

`is_pin` checks if the object is a pin.

`-filter "is_port{==|!=}{true|false}"`

`is_port` checks if the object is a port.

`-filter "max_capacitance{==|!=|<=|>=|<|>}`*float*`"`

`max_capacitance` checks the maximum capacitance value set on the port

`-filter "max_fanout{==|!=|<=|>=|<|>}`*integer*`"`

`max_fanout` checks the `max_fanout` attribute set on the port

`-filter "max_transition{==|!=|<=|>=|<|>}`*float*`"`

`max_transition` checks the `max_transition` attribute set on the port

	`-filter "object_type{==|=~|!=|!~}port"` `object_type` checks if the object is a port. **Note:** Any filter option can be specified as `option` or `@option` in the expression. You can also combine two or more expressions using the following operators: &&, ||. You can use parentheses () to enclose simple filter expressions.
`-nocase`	Indicates that the pattern is not case sensitive. This option can only be used with the `-regexp` option.
`-of_objects`	Specifies to return ports connected to the specified nets.
`pattern`	Specifies a pattern of port names or regular expressions. *Default*: *
`-regexp`	Specifies that the pattern is a regular expression.

Examples

- The following command returns the full path to all input ports starting with letter `d`.

```
rc:/designs/rct> dc::get_ports -filter direction==in d*
{/designs/rct/ports_in/din[15]} {/designs/rct/ports_in/din[14]} {/designs/
rct/ports_in/din[13]} {/designs/rct/ports_in/din[12]} {/designs/rct/ports_in/
din[11]} {/designs/rct/ports_in/din[10]} {/designs/rct/ports_in/din[9]} {/
designs/rct/ports_in/din[8]} {/designs/rct/ports_in/din[7]} {/designs/rct/
ports_in/din[6]} {/designs/rct/ports_in/din[5]} {/designs/rct/ports_in/
din[4]} {/designs/rct/ports_in/din[3]} {/designs/rct/ports_in/din[2]} {/
designs/rct/ports_in/din[1]} {/designs/rct/ports_in/din[0]} /designs/rct/
ports_in/done
```

- The following command returns the full path to all ports connected to nets whose name start with `rd_addr`.

```
rc:/designs/rct> dc::get_ports -of_objects rd_addr*
{/designs/rct/ports_in/rd_addr[0]} {/designs/rct/ports_in/rd_addr[9]} {/
designs/rct/ports_in/rd_addr[11]} {/designs/rct/ports_in/rd_addr[2]} {/
designs/rct/ports_in/rd_addr[13]} {/designs/rct/ports_in/rd_addr[4]} {/
designs/rct/ports_in/rd_addr[15]} {/designs/rct/ports_in/rd_addr[6]} {/
designs/rct/ports_in/rd_addr[8]} {/designs/rct/ports_in/rd_addr[1]} {/
designs/rct/ports_in/rd_addr[10]} {/designs/rct/ports_in/rd_addr[12]} {/
designs/rct/ports_in/rd_addr[3]} {/designs/rct/ports_in/rd_addr[14]} {/
designs/rct/ports_in/rd_addr[5]} {/designs/rct/ports_in/rd_addr[7]}
```

- The following commands return nothing:

```
rc:/> dc::get_ports -filter "is_pin==true" *
rc:/> dc::get_pins -filter "is_port==false" *
```

dc::getenv

```
getenv
     env_variable
```

Returns the contents of the specified environment variable.

Options and Arguments

env_variable	Specifies the environment variable for which you want to display the contents.

Example

The following command displays the content of the CDN_SYNTH_ROOT variable.

```
rc:/> dc::getenv CDN_SYNTH_ROOT
/net/your_install_dir/tools.lnx86
```

dc::group_path

```
group_path
     {{-from {port|pin|clock|instance}...
      |-rise_from {port|pin|clock|instance}...
      |-fall_from {port|pin|clock|instance}...}
     |{-to {port|pin|clock|instance}...
      |-rise_to {port|pin|clock|instance}...
      |-fall_to {port|pin|clock|instance}...}
     |{-through {port|pin|instance|net}...
      |-rise_through {port|pin|instance|net}...
      |-fall_through {port|pin|instance|net}...}...
     [-rise] [-fall]
     [-setup] [-hold]
     [-exception_name name]
     {-name cost_group | -default}
     [-weight float]
     [-critical_range float]
```

Assigns paths that meet the path selection criteria to a cost group. Paths can be selected using the `-from`, `-fall_from`, `-rise_from`, `-through`, `-fall_through`, `-rise_through`, `-to`, `-fall_to`, or `-rise_to` options. You must provide at least one of these options.

Options and Arguments

`-critical_range` Not supported by RTL Compiler

`-default` Associates the path group with the default cost group

`-exception_name` *`name`*

Specifies the name of the timing exception this path group should be associated with.

`-fall` Limits the path selection to those paths who have a falling edge at the end points.

`-fall_from {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a falling edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-fall_through {`*`port | pin | instance | net`*`}`

Selects all paths whose signals have a falling edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-fall_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a falling edge at the specified end points.The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-from {`*`port | pin | clock | instance`*`}`

Selects all paths that start from the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-hold` Not supported by RTL Compiler.

Note: Although the tool does not perform any hold analysis, RTL Compiler does write out this path group when the `write_sdc` command is given.

`-name` *`cost_group`* Specifies the name of the cost group to which this path group belongs.

The cost group should have been previously defined with a `define_cost_group` command.

`-rise` Limits the path selection to those paths who have a rising edge at the end points.

`-rise_from {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-rise_through {`*`port | pin | instance | net`*`}`

Selects all paths whose signals have a rising edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-rise_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge on the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-setup` Indicates that the path groups apply for setup analysis only.

`-through {`*`port | pin | instance | net`*`}`

Selects all paths that traverse through the specified points. The points to traverse can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-to {`*`port | pin | clock | instance`*`}`

Selects all paths that end at the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-weight` *`float`* Sets a weight for the cost group with which this path group is associated.

Example

- The following example assigns the paths from all inputs to all outputs to the cost group called `I2O`.

```
dc::group_path -from /designs/*/ports_in/* -to /designs/*/ports_out/* \
-name I2O
```

- The following two commands are equivalent. They both define path groups for cost group C2C that have a rising edges at the end points.

```
dc::group_path -rise -to inst1/pin1 -name C2C
dc::group_path -rise_to inst1/pin1 -name C2C
```

Related Information

Related command: define_cost_group

path_group

specify_paths

dc::remove_clock_gating_check

```
remove_clock_gating_check
     [-setup float]
     [-rise] [-fall]
     [-high] [-low]
     [instance | pin ]...
```

Removes any clock gating check value defined using the `set_clock_gating_check` command or inferred automatically by RC.

Note: To write out this constraint in the output SDC file, you must set:

```
set write_remove_cgc 1
```

Options and Arguments

`-fall`	Removes any clock gating check specified for a falling clock transition.
`-high`	Removes any clock gating check for which the non-controlling value of the clock is high.
instance | *pin*	Removes the clock gating checks from the specified instances or pins.
`-low`	Removes any clock gating check for which the non-controlling value of the clock is low.
`-rise`	Removes any clock gating check specified for a rising clock transition.
`-setup` *float*	Removes any clock gating check that has the specified setup value.

dc::remove_clock_latency

```
remove_clock_latency
     [-source]
     [-clock clock_list]
     {clock | port | pin}...
```

Removes the general and clock-specific latencies from the specified objects.

Note: Although supported by the `read_sdc` command, it is *not* an official SDC command.

Options and Arguments

`-clock` *`clock_list`*	Removes only the clock-specific latencies from the specified pins and ports with respect to the specified clocks. General latencies are not removed.
{*`clock`*|*`pin`*|*`port`*}	Specifies the objects (clocks, pins and ports) from which the latencies should be removed. **Note:** No clocks can be specified in the object list if the `-clock` option is specified.
`-source`	Removes the source latencies from the specified clocks, pins or ports. If this option is omitted, the network latencies are removed.

Example

The following command removes all the network latencies defined on pin `pin1` that refer to clock `clk1`.

```
dc::remove_clock_latency -clock clk1 pin1
```

Related Information

Related command: dc::set_clock_latency on page 275

dc::remove_disable_clock_gating_check

```
remove_disable_clock_gating_check
     {pin | instance}...
```

Restores the clock gating checks inferred on the specified pins or the enable pins of the specified instances that were present before applying the `set_disable_clock_gating_check` command.

Note: This command is supported only on mapped instances and pins of mapped instances. This command is not supported on the clock pins of an instance.

Note: To write this constraint in the output SDC file, you must set the following variable:

```
set write_disable_and_rmdisable_cgc 1
```

Options and Arguments

instance	Restores the clock gating check value(s) inferred at every single enable pin of the specified instance.
pin	Restores the clock gating check value(s) on the specified pins.

dc::remove_generated_clock

```
remove_generated_clock
     {-all | clock_list}
```

Removes all the generated clock objects in the design.

Options and Arguments

`-all`	Removes all generated clock objects.
`clock_list`	Removes the specified list of generated clocks. You can use wildcards to specify a list of clocks.

Example

- The folowing command removes the generated clock `gen_clk`.

  ```
  rc:/> dc::remove_generated_clock [find / -clock gen_clk]
  ```

- The following command removes all generated clocks whose name start with abc.

  ```
  dc::remove_generated_clock [find / -clock abc*]
  ```

dc::remove_ideal_net

`remove_ideal_net` *`net_list`*

Causes the specified nets to be treated as non-ideal and allows the timing and optimization engines to optimize these nets. The command sets the `ideal_driver` attribute on all the leaf-level drivers of the specified nets to `false`.

Options and Arguments

`net_list`	Specifies the nets which must be treated as non-ideal.

Related Information

Related command:	dc::set_ideal_net on page 303
Sets this attribute:	ideal_driver

dc::remove_ideal_network

`remove_ideal_network {pin | port | net}...`

Sets the `ideal_network` and `ideal_driver` attributes on all the leaf-level drivers of the specified net to `false`.

Options and Arguments

`{pin | port | net}` Specifies the object to which the command applies.

Related Information

Related command: dc::set_ideal_network on page 304

Sets these attributes: ideal_driver

ideal_network

dc::remove_input_delay

```
remove_input_delay
     [-clock clock] port
```

Removes the input delay on the specified port related to either the specified clock or all clocks.

Note: If an input delay is associated with several ports, it will not be removed.

Options and Arguments

`-clock clock`	Specifies the input delay relative to the specified clock. If this option is omitted, the input delays relative to all clocks on this port will be removed.
port	Removes the input delay from the specified port.

Example

```
rc:/designs/rct> dc::remove_input_delay [find / -port ena]
```

dc::remove_output_delay

```
remove_output_delay
     [-clock clock] port
```

Removes the output delay on the specified port related to either the specified clock or all clocks.

Note: If an output delay is associated with several ports, it will not be removed.

Options and Arguments

`-clock` *clock*	Specifies the output delay relative to the specified clock. If this option is omitted, the input delays relative to all clocks on this port will be removed.
port	Removes the output delay from the specified port.

dc::set_case_analysis

```
set_case_analysis
     integer {port | pin}
```

Sets the timing_case_logic_value attribute of the specified pin or port to the specified logic value when you do not perform multi-mode timing analysis.

When performing multi-mode timing analysis, this command *updates* the timing_case_logic_value_by_mode attribute of the specified pin or port with the specified logic value for the current mode. If the attribute value already contained a value for the specified mode, it will overwrite the existing value. If the attribute value did not yet contain a value for the specified mode, it will add the mode name and the corresponding logic value.

- When you use this constraint in an SDC file, the constraint will be applied to the mode specified with the -mode option of the read_sdc command that reads in the file.
- When you want to use this constraint at the RTL Compiler prompt, make sure that the mode to which this constraint needs to be applied was previously specified using the dc::set_mode command.

Options and Arguments

integer	Specifies the logic value for the specified pin or port.
{*port* | *pin*}	Specifies the pin or port for to which the logic value applies.

dc::set_clock_gating_check

```
set_clock_gating_check
     [-setup float] [-hold float]
     [-rise | -fall]
     [-high] [-low]
     [design | subdesign | clock | instance | pin ]...
```

Performs a setup check with respect to the edge of the clock signal that changes the state of the clock pin from controlling to non-controlling.

For AND and NAND gates, the setup check is performed with respect to the rising edge of the clock input. For OR and NOR gates, the setup check is performed with respect to the falling edge of the clock input.

If several clock gating checks are specified, the clock gating check specified on a pin has a higher priority then the clock gating check specified on an instance, followed by the clock gating check specified on a clock, a subdesign, the design.

Note: This command is not supported on the clock pins of an instance.

Options and Arguments

`[-fall | -rise]`	Applies the clock gating check to the falling or the rising transition. If neither option is specified, the clock gating check is applied to both the rising and falling transitions.
`[-high | -low]`	Specifies the non-controlling value of the clock. Timing analysis performs the check on the non-controlling value of the clock. By default, the software determines the non-controlling value of the clock using information from the logic of the gate. For OR gates the non-controlling value is low. For complex gates, such as XOR and MUX, the clock is never controlling and it does not perform checks unless you specify either the `-high` or `-low` option. The specified value takes precedence over the default values. This option can only be used with pins or instances.

`-hold` *`float`*

Specifies the hold value.

Note: Because the tool does not perform any minimum analysis, RTL Compiler just stores this value and writes it out when the `write_sdc` command is given.

`design` | *`subdesign`* | *`clock`* | *`instance`* | *`pin`*

Specifies the design, subdesign, clock, instance, or pin for which you want to set the clock gating check.

If no object is specified, the clock gating check is applied to the entire design.

`-setup` *`float`*

Specifies the setup value.

Default: `0.0`

dc::set_clock_groups

```
set_clock_groups
     -name string
     [-logically_exclusive] [-physically_exclusive]
     [-asynchronous]
     [-allow_paths]
     [-group clock_list]...
```

Specifies the groups of clocks between which timing analysis will not be done (does a `set_false_path` between the groups of clocks) and the type of crosstalk analysis that will be performed between the clock groups.

Although RTL Compiler does not perform crosstalk analysis, it stores the options related to crosstalk analysis (`-allow_paths, -asynchronous, -logically_exclusive,` and `-logically_exclusive`) and writes them out when the `write_sdc` command is given to preserve the original intent.

Note: This command was added in SDC 1.7.

Options and Arguments

`-allow_paths` Not supported by RTL Compiler.

Note: RTL Compiler writes out the option when you specify the `write_sdc` command.

`-asynchronous` Not supported by RTL Compiler.

Note: RTL Compiler writes out the option when you specify the `write_sdc` command.

`-logically_exclusive`

Not supported by RTL Compiler.

Note: RTL Compiler writes out the option when you specify the `write_sdc` command.

`-physically_exclusive`

Not supported by RTL Compiler.

Note: RTL Compiler writes out the option when you specify the `write_sdc` command.

`-name` *`string`* Specifies the name of the clock group.

`-group` *`clock_list`* Specifies a group of clocks.

Example

The following command sets a false path between clock1 and clock2, and clock1 and clock3

```
set_clock_groups -name cg_group1 -logically_exclusive -group {clock1} \
-group {clock2 clock3}
```

When executing the `write_sdc` command, the `set_clock_groups` command in the input SDC file is written out verbatim (including the `-logically_exclusive` option which is not supported by RTL Compiler). The false path exception created by the `set_clock_groups` command is not written out.

dc::set_clock_latency

```
set_clock_latency
     [-min ] [-max]
     [-rise ] [-fall]
     [-early] [-late]
     [-source] [-clock clock_list]
     latency {clock | port | pin}...
```

Specifies a clock source or network, early or late latency value of one or more clocks that propagate to the specified pin or port.

When you specify this command, it sets a pin, port or clock latency-related attribute.

Options and Arguments

{*clock* |*pin* | *port*}

Specifies the clocks, pins and ports to which the specified latency value applies.

`-clock` *clock_list*
: Specifies the lists of clocks to which the latency value applies.

  If no clocks are specified, the latency value applies to all clocks that propagate to the specified pin or port.

`-early`
: Indicates that the specified latency value is for an early path.

  For setup analysis, the capture path is the early path.

  For hold analysis, the launch path is the early path.

`-fall`
: Indicates that the specified latency value applies to a fall edge.

latency
: Specifies the latency value. This is a floating number.

`-late`
: Indicates that the specified latency value is for a late path.

  For setup analysis, the launch path is the late path.

  For hold analysis, the capture path is the late path.

`-min`
: Indicates that the specified latency value applies for hold analysis and is set on an *early* attribute.

`-max`
: Indicates that the specified latency value applies for setup analysis and is set on a *late* attribute.

`-rise`
: Indicates that the specified latency value applies to a rise edge.

`-source`	Indicates that the specified latency value is a source latency. If this option is not specified, the specified latency value is considered to be a network latency.

Examples

In the following examples, the latency value specified in the SDC commands is specified in ns, while the time unit in the RTL Compiler tool is in ps.

- The following commands specify the launch latency for hold analysis of the rise edges of all clocks propagating to port `p1`.

```
dc::set_clock_latency -early -min -rise 0.55 [find / -port p1]
set_attribute clock_network_early_latency \
{no_value no_value 550 no_value} [find / -port p1]
```

- The following commands specify the capture latency for hold analysis of the rise edge of all clocks propagating to port `p1`.

```
dc::set_clock_latency -late -min -rise 0.59 [find / -port p1]
set_attribute clock_network_early_latency \
{590 no_value no_value no_value} [find / -port p1]
```

- The following commands specify the capture latency for setup analysis and launch latency for hold analysis for the rise edges of the clocks propagating to port `p1`.

```
dc::set_clock_latency -early -rise 0.65 [find / -port p1]


set_attr clock_network_early_latency {no_value no_value 650 no_value} \
[find / -port p1]
set_attr clock_network_late_latency {650 no_value no_value no_value} \
[find / -port p1]
```

- The following commands specify the capture latency for hold analysis and launch latency for setup analysis for the rise edges of the clocks propagating to port `p1`.

```
dc::set_clock_latency -late -rise 0.74 [find / -port p1]


set_attr clock_network_early_latency {740 no_value no_value no_value} \
[find / -port p1]
set_attr clock_network_late_latency {no_value no_value 740 no_value} \
[find / -port p1]
```

Related Information

Related command:	dc::remove_clock_latency on page 263
Related attributes:	clock_network_early_latency
	clock_network_late_latency
	clock_source_early_latency
	clock_source_early_latency
	network_early_latency_by_clock
	network_late_latency_by_clock
	source_early_latency_by_clock
	source_late_latency_by_clock

dc::set_clock_sense

```
set_clock_sense
     [-clocks clock_list]
     [-positive | -negative | -stop_propagation]
     [-pulse string]
     {pin|port}...
```

Sets the clock sense on the specified pins and ports.

Note: This command was added in SDC 1.7.

Options and Arguments

`-clocks` *clock_list*

Specifies the clocks for which the clock sense must be set.

If no clock is specified, the command looks at all clocks defined on each of the pins and ports and changes the clock sense as specified.

`-negative` Specifies that the specified pins and ports are inverted sources of the clock.

If the pin or port is a non inverting source of the clock, the clock sense on the port is changed and it becomes an inverting source of the clock.

{pin|port} Specifies the name of the pins and ports for which the clock sense must be set.

`-positive` Specifies that the specified pins and ports are sources of the clock.

If the pin or port is an inverting source of the clock, the clock sense on the port is changed and it becomes a non-inverting source of the clock.

`-pulse` *string* Not supported by RTL Compiler.

`-stop_propagation` Stops the propagation of the specified clocks from the specified pins.

Related Information

Sets these attributes:

clock_sense_negative

clock_sense_positive

clock_sense_stop_propagation

propagated_clocks

dc::set_clock_skew

```
set_clock_skew
     [-minus_uncertainty] [-plus_uncertainty]
     float {clock | port}...
```

Sets the skew on the specified clocks and ports.

Skew is internally modeled as clock uncertainty.

If neither `-minus_uncertainty` or `-plus_uncertainty` are specified, the skew is applied for both setup and hold checks. However, only the setup value is used for timing analysis and optimization, while the hold value is stored and written out with the `write_sdc` command.

Options and Arguments

`{clock | port}`	Specifies the clocks and ports to which the specified skew value applies.
`float`	Specifies the skew value.
`-minus_uncertainty`	Shifts the clock edge to the left to make the clock arrive earlier. Used to check setup delays. Equivalent to: `dc::set_clock_uncertainty -setup` *`uncertainty`* `[`*`clock`*`|`*`port`*`]...`
`-plus_uncertainty`	Not supported by RTL Compiler. Equivalent to: `dc::set_clock_uncertainty` *`uncertainty`* `[clock|port]...`

Example

The following command sets the skew for clock `clk1` to 0.5 time units.

```
rc:/> dc::set_clock_skew 0.5 [find / -clock clk1]
```

Related Information

Related command:	dc::set_clock_uncertainty on page 283
Sets these attributes:	clock_hold_uncertainty
	clock_setup_uncertainty

dc::set_clock_transition

```
set_clock_transition
     {-min | -max} {-rise | -fall} float clock_list
```

Specifies the clock transition (slew) on the specified clocks.

You can specify four slew values: the minimum rise, minimum fall, maximum rise, and maximum fall slew.

The minimum values are not used for timing analysis, but they will be written out when you excute the `write_sdc` command.

Options and Arguments

clock_list	Specifies the clock(s) to which the clock slew applies.
-fall	Indicates that the specified value is a falling transition value.
float	Specifies the slew value for the specified clock(s).
-max	Indicates that the specified value is a maximum slew value.
-min	Indicates that the specified value is a minimum slew value.
-rise	Indicates that the specified value is a rising transition value.

dc::set_clock_uncertainty

```
set_clock_uncertainty
     [-fall | -rise]
     [-from_edge string] [-to_edge string]
     [-from clock_list | -fall_from clock_list |
       -rise_from clock_list]
     [-to clock_list | -fall_to clock_list |
       -rise_to clock_list]
     [-hold | -setup]
     [-clock clock_list] uncertainty
     [clock|port|instance|pin]...
```

Specifies the uncertainty on the clock network. You specify either a simple or an inter-clock uncertainty.

- Simple uncertainties are defined directly on a clock, port, pin, or instance.These uncertainty values are stored in attributes.
- The inter-clock uncertainties (defined using options such as `-from`, `-to`, `-rise_from`, `-rise_to`) are modeled as `path_adjust` exceptions. These uncertainties take precedence over the simple uncertainty values.

Options and Arguments

{*clock* | *pin* | *port* | *instance*}

Specifies the clocks, pins, ports and instances to which the specified uncertainty value applies. In case of instances, the uncertainty is applied on the input pins of the instance.

Note: These arguments only apply to simple uncertainties.

`-clock` *clock_list*

Specifies the clock(s) to which the uncertainty value applies.

If this option is not specified, it applies to all clocks propagating to that pin or port.

Note: This option only applies to simple uncertainties.

`-fall` | `-rise`

Specifies to apply the uncertainty to the falling or rising edge of the capture clock pin.

If neither option is specified, the uncertainty value applies to both edges.

Note: These options only apply to inter-clock uncertainties.

`-from` *`clock_list`* | `-fall_from` *`clock_list`* | `-rise_from` *`clock_list`*

Applies the uncertainty value to the specified list of launching clocks.

Note: These options only apply to inter-clock uncertainties.

`-from_edge {rise | fall | both}`

Specifies the edge type of the launch clock.

Note: This option only applies to inter-clock uncertainties and cannot be specified with either `-fall_from` or `-rise_from`.

`-hold | -setup` Specifies that the clock uncertainty applies to hold or setup checks.

If neither option is specified, the uncertainty value applies to both checks.

Note: These options apply to both types of uncertainties.

`-to` *`clock_list`* | `-fall_to` *`clock_list`* | `-rise_to` *`clock_list`*

Applies the uncertainty value to the specified list of capture clocks.

Note: These options only apply to inter-clock uncertainties.

`-to_edge {rise | fall | both}`

Specifies the edge type of the capture clock.

Note: This option only applies to inter-clock uncertainties and cannot be specified with either `-fall_to` or `-rise_to`.

`uncertainty` Specifies the uncertainty value for the clock(s). Use a floating number.

Examples

- The following command sets a (simple) setup uncertainty of 0.4 sdc units for all paths ending at `reg1` and captured by the rising transition of all clocks propagating to `reg1/CK`.

```
dc::set_clock_uncertainty -setup -rise 0.4 [find / -pin reg1/CK]
```

- The following command sets a (simple) setup uncertainty of 0.4 sdc units for all paths ending at `reg1` and captured by the rising transition of `clk1`.

```
dc::set_clock_uncertainty -setup -rise -clock clk1 0.4 [find / -pin reg1/CK]
```

- The following command sets a (inter clock) setup uncertainty of 0.5 sdc units for all paths launched by `clk2` and captured by `clk1`.

```
dc::set_clock_uncertainty -setup -from clk2 -to clk1 0.5
```

Related Information

Affects this command:	path_adjust
Related command:	clock_uncertainty
Sets these attributes:	clock_hold_uncertainty
	clock_setup_uncertainty
	hold_uncertainty_by_clock
	setup_uncertainty_by_clock

dc::set_data_check

```
set_data_check
     [-from pin | -rise_from pin | -fall_from pin]
     [{-to | -rise_to | -fall_to} pin]
     [-setup | -hold]
     [-clock clock_object] value
```

Performs a data to data check, that is, a non-sequential check used to constrain one data signal with respect to another data signal.

For setup checks the tool computes the maximum arrival time on the constrained pin (specified with the `-to|-rise_to|-fall_to` option) and checks it against the minimum arrival time on the related pin (specified with the `-from|-rise_from|-fall_from` option).

The arrival times on the related pin can depend on the clock that triggers the data event on this pin.

Current Limitations

1. RTL Compiler does not support data-to-data checks on top-level ports. RTL Compiler will not infer a check and this will also not be captured as part of `write_sdc`.
2. RTL Compiler does not support data-to-data checks speci• ed on pins belonging to different instances. RTL Compiler will not infer such a data check.

Options and Arguments

`-clock` *clock*
: Specifies the clock that triggers the data event on the related (-from) pin.

  If this option is not specified, the data check is clock independent.

`-from` *pin* | `-fall_from` *pin* | `-rise_from` *pin*
: Specifies the related pin, that is, the pin with respect to which the constraint is set.

  If you specify the -from option, the data check is performed against the falling and rising edge of related pin.

`-hold`
: Not supported by RTL Compiler.

`-setup`
: Applies the data check for setup analysis.

`-to` *pin* | `-fall_to` *pin* | `-rise_to` *pin*

Specifies the constrained pin, that is, the pin to which the constraint applies.

If you specify the `-to` option, the data check is performed on the falling and rising edge of specified pin.

value

Specifies the data check value in the time units specified with the `set_time_unit` command.. Use a floating number.

Example

Consider the following module:

```
module top (in1,in2,c1,c2,c3,out);
    input in1,in2,c1,c2,c3;
    output out;
    fflopd inst1 (.D(in1),.CK(c1),.Q(w1));
    fflopd inst2 (.D(in2),.CK(c2),.Q(w2b));
    buf1 b1 (.A(w2b),.Y(w2b1));
    buf1 b2 (.A(w2b1),.Y(w2b2));
    buf1 b3 (.A(w2b2),.Y(w2b3));
    buf1 b4 (.A(w2b3),.Y(w2b4));
    buf1 b5 (.A(w2b4),.Y(w2b5));
    buf1 b6 (.A(w2b5),.Y(w2b6));
    buf1 b7 (.A(w2b6),.Y(w2b7));
    nand2 inst3 (.A(w1),.B(w2b7),.Y(w3));
    fflopd inst4 (.D(w3),.CK(c3),.Q(out));
endmodule
```

The following command specifies a setup check between pins `B` and `A` of instance `inst3` and verifies that the arrival time on `A` is less than the specified value (0.9 time units):

```
dc::set_data_check -setup -from inst3/B -to inst3/A 0.9
```

Related Information

Related command: dc::set_time_unit on page 345

dc::set_disable_clock_gating_check

```
set_disable_clock_gating_check
     {pin | instance} ...
```

Disables the clock gating check inferred on the specified pins or the enable pins of the specified instance.

Note: This command is supported only on mapped instances and pins of mapped instances. This command is not supported on the clock pins of an instance.

Note: To write this constraint in the output SDC file, you must set the following variable:

```
set write_disable_and_rmdisable_cgc 1
```

Options and Arguments

instance	Disables all clock gating checks inferred at every single enable pin of the specified instance. To restore the clock gating check on an instance, use: `dc::remove_disable_clock_gating_check instance`
pin	Specifies the pin on which to apply the command. If the pin is the enable pin of a clock gate, all clock gating checks inferred at the enable pin are disabled and the timing path is not broken at this pin. To restore the clock gating check on a pin, use: `dc::remove_disable_clock_gating_check pin`

dc::set_dont_touch

```
set_dont_touch
     {design | subdesign | instance | net | libcell} ...
```

Preserves the specified object(s). Sets the `preserve` attribute to `true` on the specified object.

This command does not work on unmapped or unresolved objects.

For net objects, the `preserve` attribute is only set on the specified net, and not on all the segments of the physical net (if the net traverses many hierarchies). To preserve the entire *hierarchical* net, set the `sdc_set_dont_touch_physical_net` variable to 1.

Options and Arguments

{*design* | *subdesign* | *instance* | *net* | *libcell*}

Specifies the object(s) that you want to preserve.

dc::set_dont_touch_network

```
set_dont_touch_network
     [-no_propagate]
     {clock | port | pin}...
```

Preserves the nets and combination logic fanning out from a particular port or pin. If a clock object is specified, the nets and combinational logic fanning out from the clock sources are preserved.

Options and Arguments

{*clock* | *port* | *pin*}
: Specifies the objects to be preserved.

-no_propagate
: Preserves all the nets driven by the specified pin up to the first leaf-cell-input (combinational or sequential).

  If this option is omitted, preserving stops at the first sequential cell in the network.

Example

The following command preserves all nets and combinational logic fanning out of port `in`, till the first sequential cell is encountered in each fanout path.

```
dc::set_dont_touch_network [find / -port in]
```

Related Information

Sets this attribute: preserve

dc::set_dont_use

```
set_dont_use
     libcell_list [true | TRUE | false | FALSE]
```

Indicates whether the specified library cell(s) should be avoided by the technology mapper. Sets the `avoid` attribute on the specified cells to the specified value.

Options and Arguments

`libcell_list` Specifies the library cell(s) that must be marked for the technology mapper.

{`true` | `TRUE` | `false` | `FALSE`}

Specifies whether the library cell(s) should be avoided.

- `true` or `TRUE` indicates that the cell(s) should be avoided.
- `false` or `FALSE` indicates that the cell can be used by the mapper.

Default: {`true` | `TRUE`}

Examples

- The following two SDC commands result in the `avoid` attribute being set to `true` on the `inv1` cell.

  ```
  dc::set_dont_use inv1
  dc::set_dont_use inv1 true
  ```

- The following SDC command results in the `avoid` attribute being set to `false` on the `inv1` cell.

  ```
  dc::set_dont_use inv1 false
  ```

dc::set_drive

```
set_drive
     [-rise] [-fall]
     [-min] [-max]
     resistance port_list
```

Specifies the resistance of the external driver in kilohms for a minimum rise, minimum fall, maximum rise, and maximum fall transition. The resolution is 1/1000. RTL Compiler ignores (does not use) the minimum values but the they can be passed to downstream tools through `write_sdc`.

As a result, the tool sets the `external_resistance` attribute on the specified ports and removes the external driver (sets the `external_driver` attribute to NULL) on the specified ports.

Options and Arguments

`-fall`	Indicates that the specified resistance value applies to a fall transition.
`-min`	Indicates that the specified resistance value applies for hold analysis.
`-max`	Indicates that the specified resistance value applies for setup analysis.
`port_list`	Specifies the lists of ports to which the resistance value applies.
`resistance`	Specifies the external resistance value. This is a floating number.
`-rise`	Indicates that the specified resistance value applies to a rise transition.

Examples

- The following command sets the resistance for the minimum and maximum rise and fall transitions of the external driver of port `in` to 0.560.

```
rc:/> dc::set_drive 0.56 [find / -port in]
rc:/> get_attr external_resistance [find / -port in]
0.560 0.560 0.560 0.560
```

- The following commands set the resistance for the minimum rise and maximum rise transition of the external driver of port fS_opb[1], respectively to 0.560 and 0.280.

```
rc:/designs/cpu_top/> dc::set_drive 0.56 [find / -port S_opb[1]] -min -ris
rc:/designs/cpu_top/> get_attr external_resistance [find / -port S_opb[1]]
0.560 no_value no_value no_value
rc:/designs/cpu_top/> dc::set_drive 0.28 [find / -port S_opb[1]] -max  -ris
rc:/designs/cpu_top/> get_attr external_resistance [find / -port S_opb[1]]
0.560 no_value 0.280 no_value
```

Related Information

Related attributes: external_resistance

external_driver

dc::set_driving_cell

```
set_driving_cell
     {-cell | -lib_cell} cell [-library library]
     [-from_pin pin] [-pin pin]
     [-input_transition_rise float]
     [-input_transition_fall float]
     [-rise] [-fall] [-max | -min]
     [-multiply_by integer] [-no_design_rule]
     [-dont_scale]
     [-clock clock_list] [-clock_fall] port_list
```

Specifies the libcell that drives the specified ports.

Options and Arguments

`-cell` *cell*	Specifies the name of the libcell used to drive the port.
`-clock`	Not supported by RTL Compiler.
`-clock_fall`	Not supported by RTL Compiler.
`-dont_scale`	Not supported by RTL Compiler.
`-fall`	Indicates that the information applies only to a falling transition on the ports. By default, if both the `-fall` and `-rise` options are omitted, the information applies to the rising and falling transitions on the ports.
float	Specifies the transition time for the specified port(s).
`-from_pin` *pin*	Specifies an input pin on the specified cell. The command uses the drive of the timing arc from this pin to the pin specified with the `-pin` option. If this option is omitted, the transition on `-pin` is assumed to be 0.
`-input_transition_fall` *float*	Specifies the input falling transition time associated with the `-from_pin` option. *Default*: `0`

`-input_transition_rise` *`float`*

Specifies the input rising transition time associated with the `-from_pin` option.

Default: `0`

`-lib_cell` *`cell`*

Specifies the name of the libcell used to drive the port.

`-library` *`library`*

Specifies the name of the library to which the libcell belongs.

If this option is omitted, the command searches in all available libraries.

`-max`

Indicates that the driver cell information applies to setup analysis.

By default, if both the `-max` and `-min` options are omitted, the information applies to setup analysis.

`-min`

Not supported by RTL Compiler.

`-multiply_by` *`integer`*

Specifies the total number of external drivers. The number of additional external drivers is the specified number minus 1 and this value will be set on the `external_non_tristate_drivers` port attribute.

Default: `1`

`-no_design_rule`

Indicates to ignore the design rule violations seen by the external driver pin.

`-pin` *`pin`*

Specifies the output pin of the libcell which will be used to drive the port.

If this option is omitted, the command chooses an output pin at random.

`port_list`

Specifies the port(s) to which the external driver applies.

`-rise`

Indicates that the information applies only to a rising transition on the ports.

By default, if both the `-fall` and `-rise` options are omitted, the information applies to the rising and falling transitions on the ports.

Example

The following command specifies `nand2` as driving cell for port `a/foo[3]`. The from_pin `A` of the `nand2` libcell has a rising input transition time of `0.1`.

```
dc::set_driving_cell -lib_cell nand2 -from_pin A -input_transition_rise 0.10
[dc::get_ports {a/foo[3]}]
```

Related Information

Related attribute: external_non_tristate_drivers

dc::set_equal

```
set_equal
     port port
```

Declares the two specified ports to be logically equivalent.

Note: The command does not work on output ports and ports with multidriven nets.

Example

The following command declares ports `in` and `in1` as logically equivalent.

```
rc:/> dc::set_equal [find / -port in] [find / -port in1]
```

dc::set_false_path

```
set_false_path
     {{-from {port|pin|clock|instance}...
      |-rise_from {port|pin|clock|instance}...
      |-fall_from {port|pin|clock|instance}...}
     |{-to {port|pin|clock|instance}...
      |-rise_to {port|pin|clock|instance}...
      |-fall_to {port|pin|clock|instance}...}
     |{-through {port|pin|instance|net}...
      |-rise_through {port|pin|instance|net}...
      |-fall_through {port|pin|instance|net}...}...
     [-rise] [-fall]
     [-setup] [-hold]
     [-exception_name name]
```

Applies a false path constraint to all selected paths.

Note: This command is similar to the native `path_disable` command.

Options and Arguments

`-exception_name` *`name`*

Specifies the name of the timing exception you want to create.

Default: `dis_`*`n`*

`-fall`

Limits the path selection to those paths who have a falling edge at the end points.

`-fall_from {`*`port`* | *`pin`* | *`clock`* | *`instance`*`}`

Selects all paths whose signals have a falling edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-fall_through {`*`port`* | *`pin`* | *`instance`* | *`net`*`}`

Selects all paths whose signals have a falling edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-fall_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a falling edge at the specified end points.The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-from {`*`port | pin | clock | instance`*`}`

Selects all paths that start from the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-hold` Not supported by RTL Compiler.

Note: Although the tool does not perform any hold analysis, RTL Compiler does write out this constraint when the `write_sdc` command is given.

`-rise` Limits the path selection to those paths who have a rising edge at the end points.

`-rise_from {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-rise_through {`*`port | pin | instance | net`*`}`

Selects all paths whose signals have a rising edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-rise_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge on the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-setup` Applies the false path constraint for setup analysis only.

`-through {`*`port`* | *`pin`* | *`instance`* | *`net`*`}`

Selects all paths that traverse through the specified points. The points to traverse can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-to {`*`port`* | *`pin`* | *`clock`* | *`instance`*`}`

Selects all paths that end at the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

Examples

- The following command applies a false path (disables the path for timing analysis) to all paths between `inst1/CK` and inst2/D.

  ```
  dc::set_false_path  -from inst1/CK  -to inst2/D
  ```

- The following command applies a false path (disables the path for timing analysis) through all paths that go through `inst1/A.`

  ```
  dc::set_false_path through inst1/A
  ```

dc::set_fanout_load

`set_fanout_load` *`float port_list`*

Specifies the load that the port(s) see outside the design.

This command sets the `external_fanout_load` attribute on the specified output and inout ports.

Options and Arguments

`float`	Specifies the external load.
`port_list`	Specifies the ports to which the external load applies.

Example

The following command specifies an external load of 45.6 for port `out`.

```
rc:/> dc::set_fanout_load 45.6 [find / -port out]
rc:/> get_attr external_fanout_load [find / -port out]
45.600
```

Related Information

Sets this attribute: external_fanout_load

dc::set_hierarchy_separator

```
set_hierarchy_separator
     character
```

Specifies the hierarchy delimiter character used in the SDC file.

Options and Arguments

character	Specifies the hierarchy delimiter character. *Default*: /

Example

In the following example the hierarchy delimiter is set to #. The next command returns the full path to the instances that match the specified pattern inst10#inst1. Only one instance is returned in this case.

```
dc::set_hierarchy_separator #
dc::get_cells inst10#inst1
/designs/top/instances_hier/inst10/instances_comb/inst1
```

dc::set_ideal_net

`set_ideal_net` *`net_list`*

Causes the specified nets to be treated as ideal and prevents the timing and optimization engines from optimizing these nets. The command sets the `ideal_driver` attribute on the leaf-level drivers of the specified nets to `true`.

Options and Arguments

`net_list`	Specifies the nets which must be treated as ideal nets.

Example

```
dc::set_ideal_net opa_5[1] opa_5[2]
```

Related Information

Related command:	dc::remove_ideal_net on page 266
Sets this attribute:	ideal_driver

dc::set_ideal_network

```
set_ideal_network {pin | port | net}... [-no_propagate]
```

Sets the `ideal_network` attribute on all leaf-level drivers of the specified object to `true`.

Options and Arguments

`{pin | port | net}`

Specifies the object to which the command applies.

`-no_propagate`

Only sets the `ideal_driver` attribute on the leaf-level drivers of the specified object to `true`. This way only the electrically connected nets are idealized.

Related Information

Related command: dc::remove_ideal_network on page 267

Sets this attribute: ideal_driver

ideal_network

dc::set_input_delay

```
set_input_delay
     [-clock clock]
     [-clock_fall] [-clock_rise]
     [-level_sensitive]
     [-network_latency_included]
     [-source_latency_included]
     [-rise] [-fall] [-max] [-min]
     [-add_delay] [-name name]
     delay {port|pin|port_bus}...
```

Constrains input and inout ports, port busses, and pins (that are valid startpoints) within the design relative to a clock edge. Calls the native `external_delay` command.

If you omit both the `-min` and `-max` options, the delay is assumed to apply for both setup and hold analysis.

If you omit both the `-rise` and `-fall` options, the delay applies for both the rising and falling edges of the input signal.

Note: This command sets the `external_delays` attribute on the specified pins or ports.

Options and Arguments

`-add_delay`	Specifies to add the specified input delay information for the specified pins or ports. Use this option to capture the delay information if multiple paths lead to an input pin or port that are relative to different clocks or clock edges. If you omit this option, the specified delay information overwrites all existing input delays for the specified pins or ports.
`-clock` *`clock`*	Specifies the reference clock. The input delay is defined relative to this clock. If no clocks are specified, the input delay is relative to all clocks.
`-clock_fall`	Specifies to use the falling edge of the reference clock as reference edge. *Default:* `-clock_rise`

`-clock_rise` Specifies to use the rising edge of the reference clock as reference edge.

Default: `-clock_rise`

`delay` Specifies the input delay value. This is a floating number.

`-fall` Specifies that the delay is measured with respect to the falling edge of the input signal.

`-level_sensitive` Specifies that the constraint comes from a level-sensitive latch.

By default, the constraint comes from an edge-triggered flip-flop.

`-max` Indicates that the specified delay applies for setup analysis.

`-min` Indicates that the specified delay applies for hold analysis. Hold analysis is not supported by RTL Compiler, but the value is written out with `write_sdc`.

`-name` *`name`* Associates a name with the specified timing constraint.

`-network_latency_included`

Specifies that the input delay includes the clock network latency values. This implies that the tool does not need to consider the clock network latency values when optiming this path.

Sets the `clock_network_latency_included` attribute to `true`.

{*`port`* | *`pin`* | *`port_bus`*}

Specifies the input and inout pins, ports, and port buses to which the specified input delay value applies.

`-rise` Specifies that the delay is measured with respect to the rising edge of the input signal.

`-source_latency_included`

Specifies that the input delay includes the clock source latency values. This implies that the tool does not need to consider the clock source latency values when optiming this path.

Sets the `clock_source_latency_included` attribute to `true`.

Example

The following commands set an input delay on input port `en` with respect to clocks `clk1` and `clk2`. The example shows the effect of using the `-add_delay` option. When the second `set_input_delay` is specified without the `-add_delay` option, the first input delay is removed. However, when the third `set_input_delay` is specified with the `-add_delay` option, the delay information is stored.

```
rc:/> dc::set_input_delay -clock clk1 2 en
rc:/> get_attr external_delays [find / -port en]
/designs/test2/timing/external_delays/in_del
rc:/> dc::set_input_delay -clock clk2 5 en
   Removing external delay 'in_del'.
Info    : Removed object. [TUI-58]
       : Removed external_delay '/designs/test2/timing/external_delays/in_del'.
rc:/> get_attr external_delays [find / -port en]
/designs/test2/timing/external_delays/in_del_1_1
rc:/> dc::set_input_delay -add -clock clk1 2 en
rc:/> get_attr external_delays [find / -port en]
/designs/test2/timing/external_delays/in_del_1_1 /designs/test2/timing/
external_delays/in_del
```

Related Information

Sets these attributes: clock_network_latency_included

clock_source_latency_included

external_delays

dc::set_input_transition

```
set_input_transition
     {-min | -max} {-rise | -fall}
     [-clock_fall] [-clock clock_list]
     float port_list
```

Sets a specific transition time (slew) on the specified input and inout ports. You can specify the minimum rise, minimum fall, maximum rise, or maximum fall slew.

Sets the `fixed_slew` attribute and removes the value from the the `external_driver` attribute on the specified ports.

The minimum values are not used for timing analysis, but they will be written out when you excute the `write_sdc` command.

Options and Arguments

`-clock`	Not supported by RTL Compiler.
`-clock_fall`	Not supported by RTL Compiler.
`-fall`	Indicates that the specified value is a falling transition value.
`float`	Specifies the transition time for the specified port(s).
`-max`	Indicates that the specified value is a maximum transition value.
`-min`	Indicates that the specified value is a minimum transition value.
`port_list`	Specifies the port(s) to which the transition time applies.
`-rise`	Indicates that the specified value is a rising transition value.

Example

The following command sets the minimum rise slew for input port `in` to `0.6`.

```
rc:/> dc::set_input_transition -rise -min 0.6 /designs/top/ports_in/in
rc:/> get_attr fixed_slew /designs/top/ports_in/in
600.0 no_value no_value no_value
```

Related Information

Sets these attributes: external_driver

fixed_slew

dc::set_lib_pin

```
{set_lib_pin | set_lib_pins}
     [-max_capacitance float]
     [-max_fanout integer]
     [-max_transition float]
     libpin_list
```

Sets the `max_fanout`, `max_transition` and `max_capacitance` attributes on the specified libpin.

Options and Arguments

`-max_capacitance` *`float`*

Specifies the maximum capacitive load that can be driven by the specified libcell pins.

`-max_fanout` *`integer`*

Specifies the maximum fanout that can be driven by the specified libcell pins.

`-max_transition` *`float`*

Specifies the maximum transition time on the specified libcell pins

`libpin_list` Specifies the libcell pins to which the specified constraints apply.

Example

The following command specifies that the maximum fanout that can be driven by pin `Y` of libcell `nand2` is 10.

```
rc:/> dc::set_lib_pin -max_fanout 10 nand2/Y
```

Related Information

Sets these attributes: max_capacitance

max_fanout

max_transition

dc::set_load

```
set_load
     { [-pin_load]
       [-subtract_pin_load]
       [-wire_load] port...
     | net...
     | port_bus ...}
     [-min] [-max] capacitance
```

Sets the capacitance to the specified value on the ports and nets in the design.

If you omit both the `-min` and `-max` options, the capacitance applies for both setup and hold analysis.

Options and Arguments

capacitance	Specifies the capacitance. This is a floating number.
`-max`	Indicates that the capacitance applies for setup analysis.
`-min`	Indicates that the capacitance applies for hold analysis. Hold analysis is not supported by RTL Compiler, but the value is written out with `write_sdc`.
`-pin_load`	Specifies the load of pins connected to this port. Sets the `external_pin_cap` attribute on the port. This option applies only to ports.
{*port* | *net* | *port_bus*}	Specifies the port, net, or port_bus objects to which the specified capacitance applies.
`-subtract_pin_load`	Subtracts the capacitance of the pins connected to the net of the port from the specified capacitance value before it applies it to the ports. This option applies only to ports.
`-wire_load`	Specifies the load of the wire connected to this port. Sets the `external_wire_cap` attribute on the port. This option applies only to ports.

Related Information

Sets these attributes:	external_pin_cap
	external_wire_cap

dc::set_load_unit

```
set_load_unit
     {-picofarads | -femtofarads } [float]
```

Specifies the load unit used inside the SDC file or as part of the `dc::` command.

Note: Although supported by the `read_sdc` command, it is *not* an official SDC command.

Options and Arguments

`float`	Indicates the number that any load value in subsequent SDC commands should be multiplied with.
`-femtofarads`	Indicates that the load values are specified in femto farads.
`-picofarads`	Indicates that the load values are specified in pico farads.

Examples

- The following command indicates that each load unit in the SDC file equals 5 femtofarads.

  ```
  dc::set_load_unit -femtofarads 5
  ```

dc::set_logic_dc

```
set_logic_dc
     {pin | port} ...
```

If the specified object is an output port or an input pin, connects the object to constant 0.

If the specified object is an input port or an output pin, connects the loads of these objects to constant 0.

Note: This command does not work on pins or ports with multi-driver nets.

Options and Arguments

pin	Specifies the name of the pin(s) to which the command applies.
port	Specifies the name of the port(s) to which the command applies.

dc::set_logic_one

```
set_logic_one
     {pin | port} ...
```

If the specified object is an output port or an input pin, connects the object to constant 1.

If the specified object is an input port or an output pin, connects the loads of these objects to constant 1.

Note: This command does not work on pins or ports with multi-driver nets.

Options and Arguments

pin	Specifies the name of the pin(s) to which the command applies.
port	Specifies the name of the port(s) to which the command applies.

dc::set_logic_zero

```
set_logic_zero
     {pin | port} ...
```

If the specified object is an output port or an input pin, connects the object to constant 0.

If the specified object is an input port or an output pin, connects the loads of these objects to constant 0.

Note: This command does not work on pins or ports with multi-driver nets.

Options and Arguments

pin	Specifies the name of the pin(s) to which the command applies.
port	Specifies the name of the port(s) to which the command applies.

dc::set_max_capacitance

```
set_max_capacitance
     float {design | port}...
```

Specifies the maximum capacitance that can be driven by all ports in the specified design(s) or that can be driven by the specified ports.

Sets the `max_capacitance` attribute on the specified design(s) or port(s).

Options and Arguments

design	Specifies the name of the design to which the maximum capacitance design rule constraint applies.
float	Specifies the maximum capacitance value.
port	Specifies the names of the ports to which the maximum capacitance design rule constraint applies.

Example

The following command constrains the maximum capacitance that can be driven by port `in_port_1` to `0.5`.

```
dc::set_max_capacitance 0.5 [dc::get_ports in_port_1]
```

dc::set_max_delay

```
set_max_delay
     {{-from {port|pin|clock|instance}...
      |-rise_from {port|pin|clock|instance}...
      |-fall_from {port|pin|clock|instance}...}
     |{-to {port|pin|clock|instance}...
      |-rise_to {port|pin|clock|instance}...
      |-fall_to {port|pin|clock|instance}...}
     |{-through {port|pin|instance|net}...
      |-rise_through {port|pin|instance|net}...
      |-fall_through {port|pin|instance|net}...}...
     [-rise] [-fall]
     [-setup] [-hold]
     [-exception_name name] float
```

Specifies the maximum delay that the selected paths can have.

Note: This command is similar to the native `path_delay` command.

Options and Arguments

`-exception_name` *name*

Specifies the name of the timing exception you want to create.

Default: `del_`*n*

`-fall`

Limits the path selection to those paths who have a falling edge at the end points.

`-fall_from {`*port* | *pin* | *clock* | *instance*`}`

Selects all paths whose signals have a falling edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-fall_through {`*port* | *pin* | *instance* | *net*`}`

Selects all paths whose signals have a falling edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-fall_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a falling edge at the specified end points.The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`float` Specifies the maximum delay value.

`-from {`*`port | pin | clock | instance`*`}`

Selects all paths that start from the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-hold` Not supported by RTL Compiler.

Note: Although the tool does not perform any hold analysis, RTL Compiler does write out this constraint when the `write_sdc` command is given.

`-rise` Limits the path selection to those paths who have a rising edge at the end points.

`-rise_from {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-rise_through {`*`port | pin | instance | net`*`}`

Selects all paths whose signals have a rising edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-rise_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge on the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-setup` Applies the maximum delay constraint for setup analysis.

`-through {`*`port`* | *`pin`* | *`instance`* | *`net`*`}`

Selects all paths that traverse through the specified points. The points to traverse can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-to {`*`port`* | *`pin`* | *`clock`* | *`instance`*`}`

Selects all paths that end at the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

Example

The following command specifies a maximum delay of 10 time units to the path that starts from `inst1/CK` and ends at `inst2/D`.

```
dc::set_max_delay  -from inst1/CK  -to inst2/D 10
```

dc::set_max_dynamic_power

```
set_max_dynamic_power
     float unit [design]
```

Specifies the maximum dynamic power constraint for the design.

Options and Arguments

design	Specifies the name of the design to which the constraint applies.
float	Specifies the maximum dynamic power value.
unit	Specifies the power units. Following units can be specified: pW, nW, and uW.

Example

The following command sets the maximum dynamic power constraint to 0.5 pW.

```
dc::set_max_dynamic_power 0.5pW
```

Related Information

Sets this attribute:	max_dynamic_power
Related command:	dc::set_max_leakage_power on page 323

dc::set_max_fanout

```
set_max_fanout
     integer {design | port}...
```

Specifies the maximum fanout that can be driven by all input ports in the specified design(s) or that can be driven by the specified input ports.

Sets the `max_fanout` attribute on the specified design(s) or port(s).

Options and Arguments

design	Specifies the name of the design to which the maximum fanout design rule constraint applies.
integer	Specifies the maximum fanout value.
port	Specifies the names of the input ports to which the maximum fanout design rule constraint applies.

Example

The following command constrains the maximum fanout that can be driven by port `in_port_1` to `2`.

```
dc::set_max_fanout 2 [dc::get_ports in_port_1]
```

dc::set_max_leakage_power

```
set_max_leakage_power
     float unit [design]
```

Specifies the maximum leakage power constraint for the design.

Options and Arguments

design	Specifies the name of the design to which the constraint applies.
float	Specifies the maximum leakage power value.
unit	Specifies the power units. Following units can be specified: `pW`, `nW`, and `uW` .

Example

The following command sets the maximum leakage power constraint to 4pW.

```
rc:/> dc::set_max_leakage_power 4pW
rc:/> get_attribute max_leakage_power /designs/top
0.004
rc:/> get_attribute lp_power_unit /
nW
```

Related Information

Sets this attribute:	max_leakage_power
Related command:	dc::set_max_dynamic_power on page 321

dc::set_max_time_borrow

```
set_max_time_borrow
     float [clock | instance | pin | port ]...
```

Specifies the maximum amount of time that can be borrowed from the next clock cycle.

Sets the `latch_max_borrow` attribute.

Important

If the latch_borrow attribute has already been set on the specified object then the setting from this command is ignored.

Options and Arguments

{*clock* | *instance* | *pin* | *port*}

Specifies the object(s) to which the borrow value applies.

You can set the borrow value for latch instances, clocks, latch enable pins, data pins and top-level ports.

If the command is set on multiple objects that overlap each other, for example a latch instance and its data pin, the minimum value will be taken.

float

Specifies the borrow value to be applied.

Example

The following command sets a maximum borrow time of `0.5` for all latches clocked by `clk1`.

```
rc:/> dc::set_max_time_borrow 0.5 [find / -clock clk1]
```

Related Information

Sets this attribute: latch_max_borrow

dc::set_max_transition

```
set_max_transition
     [-clock_path string] [-data_path string]
     [-rise] [-fall] float [design | port]...
```

Specifies the maximum transition design rule for the specified design or port list.

Options and Arguments

`-clock_path`	Not supported by RTL Compiler.
`-data_path`	Not supported by RTL Compiler.
design	Specifies the name of the design to which the maximum transition design rule constraint applies.
`-fall`	Not supported by RTL Compiler.
float	Specifies the maximum transition. **Note:** The units are either the units set with the `set_time_unit` command or the units of the first technology library read in.
`-rise`	Not supported by RTL Compiler.
port	Specifies the names of the input ports to which the maximum transition design rule constraint applies.

Example

The following command sets the maximum transition on the design to 4.

```
rc:/designs/rct> dc::set_max_transition 4
rc:/designs/rct> get_attr max_transition
4000.0
```

In this example, the time unit that applied when the command was given was nanoseconds, while in the tool the value is stored in picoseconds.

Related Information

Sets these attributes:	(design) max_transition
	(port) max_transition

dc::set_min_delay

```
set_min_delay
     {{-from {port|pin|clock|instance}...
      |-rise_from {port|pin|clock|instance}...
      |-fall_from {port|pin|clock|instance}...}
     |{-to {port|pin|clock|instance}...
      |-rise_to {port|pin|clock|instance}...
      |-fall_to {port|pin|clock|instance}...}
     |{-through {port|pin|instance|net}...
      |-rise_through {port|pin|instance|net}...
      |-fall_through {port|pin|instance|net}...}...
     [-rise] [-fall]
     [-exception_name name]
     float
```

Constrains the specified timing paths by the given delay value for hold analysis. This value is not used for timing analysis by RTL Compiler. But it will be written out using `write_sdc`.

Options and Arguments

`-exception_name` Specifies the name of the timing exception you want to create.

Default: `del_n`

`-fall` Limits the path selection to those paths who have a falling edge at the end points.

`-fall_from {port | pin | clock | instance}`

Selects all paths whose signals have a falling edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-fall_through {port |pin | instance | net}`

Selects all paths whose signals have a falling edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-fall_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a falling edge at the specified end points.The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`float`

Specifies the the delay constraint value, in picoseconds, by which the path has to be constrained for hold analysis.

`-from {`*`port | pin | clock | instance`*`}`

Selects all paths that start from the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-rise`

Limits the path selection to those paths who have a rising edge at the path end points.

`-rise_from {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-rise_through {`*`port |pin | instance | net`*`}`

Selects all paths whose signals have a rising edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-rise_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge on the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list of end points.

`-through {`*`port |pin | instance | net`*`}`

Selects all paths that traverse through the specified points. The points to traverse can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-to {`*`port`* | *`pin`* | *`clock`* | *`instance`*`}`

Selects all paths that end at the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list of end points.

Example

The following command constrains the specified timing paths from port `in` by 0.6 time units for hold analysis.

```
rc:/> dc::set_min_delay -from [find / -port in] 0.6
/designs/topmost/timing/exceptions/path_delays/del_2
```

dc::set_mode

```
set_mode
     mode
```

Specifies the mode to which subsequent constraints—entered *interactively* at the RTL Compiler prompt—must be applied.

Once this command is set, all constraints that need a mode specification will apply the constraint to the specified mode.

Note: You do not need to explicitly set `dc::set_mode` before each constraint.

Tip

When reading in the SDC file using the `read_sdc` command, use the `-mode` option to specify the mode to which all the constraints in the SDC file must be applied.

Options and Arguments

mode	Specifies the mode to which subsequent constraints must be applied.

dc::set_multicycle_path

```
set_multicycle_path
     {{-from {port|pin|clock|instance}...
      |-rise_from {port|pin|clock|instance}...
      |-fall_from {port|pin|clock|instance}...}
     |{-to {port|pin|clock|instance}...
      |-rise_to {port|pin|clock|instance}...
      |-fall_to {port|pin|clock|instance}...}
     |{-through {port|pin|instance|net}...
      |-rise_through {port|pin|instance|net}...
      |-fall_through {port|pin|instance|net}...}...
     [-rise] [-fall]
     [-setup] [-hold]
     [-start] [-end] float
     [-exception_name name]
```

Creates a timing exception object that overrides the default clock edge relationship for paths that meet the path selection criteria. Multicycle paths are timing paths that require more than one clock period for execution. By default, all paths are considered as one cycle paths for setup analysis.

Note: This command is similar to the native `multi_cycle` command.

Options and Arguments

`-end`
: Specifies whether the multi-cycle path constraint must be applied relative to the ending clock edge for the path.

  If neither the `-start` or `-end` options are specified, the constraint applies to the ending clock edge of the path. If both options are specified, the constraint applies to the starting clock edge of the path.

`-exception_name` *`name`*
: Specifies the name of the timing exception you want to create.

  Default: `mc_`*`n`*

`-fall`
: Limits the path selection to those paths who have a falling edge at the end points.

`-fall_from {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a falling edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-fall_through {`*`port |pin | instance | net`*`}`

Selects all paths whose signals have a falling edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-fall_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a falling edge at the specified end points.The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`float` Specifies the number of cycles allowed for the path.

`-from {`*`port | pin | clock | instance`*`}`

Selects all paths that start from the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-hold` Not supported by RTL Compiler.

Note: Although the tool does not perform any hold analysis, RTL Compiler does write out this constraint when the `write_sdc` command is given.

Default: setup

`-rise` Limits the path selection to those paths who have a rising edge at the path end points.

`-rise_from {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-rise_through {`*`port |pin | instance | net`*`}`

Selects all paths whose signals have a rising edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-rise_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge on the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list of end points.

`-setup`

Applies the multi-cycle path constraint for setup analysis.

Default: setup

`-start`

Specifies whether the multi-cycle path constraint must be applied relative to the starting clock edge for the path.

If neither the `-start` or `-end` options are specified, the constraint applies to the ending clock edge of the path. If both options are specified, the constraint applies to the starting clock edge of the path.

`-through {`*`port |pin | instance | net`*`}`

Selects all paths that traverse through the specified points. The points to traverse can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list of start points.

`-to {`*`port | pin | clock | instance`*`}`

Selects all paths that end at the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list of end points.

Example

The following command specifies that all paths from instance `ff1` to instance `ff2` require 2 cycles for execution for setup analysis.

```
dc::set_multicycle_path -setup 2 -from ff1 -to ff2
```

dc::set_operating_conditions

```
set_operating_conditions
     [-analysis_type string] [-object_list string]
     { -max_library string [-min_library string ]
     | -library string }
     {-min string -max string |operating_condition}
```

Specifies the operating conditions to be used for timing. Sets the `operating_conditions` attribute.

Options and Arguments

`-analysis_type` *`string`*

The only accepted value is `single`. This indicates that only one operating condition is to be used.

`-library` *`string`* Same as `-max_library`.

`-max` *`string`* Specifies the operating condition for setup analysis.

`-max_library` Specifies the library under which the operating condition is present. This operating condition is used for setup analysis.

`-min` *`string`* Specifies the operating condition for hold analysis. Hold analysis is not supported by RTL Compiler, but the value is written out by `write_sdc`.

`-min_library` Specifies the library under which the operating condition is present for hold analysis. Hold analysis is not supported by RTL Compiler, but the value is written out by `write_sdc`.

`-object_list` Not supported by RTL Compiler.

`operating_condition`

Specifies the operating conditions to be used for timing analysis.

Example

The following command specifies to use operating condition `typical_case` from library `tutorial` for timing analysis.

```
dc::set_operating_conditions -library tutorial typical_case
```

dc::set_opposite

```
set_opposite
     port port
```

Declares the two specified ports to be logically opposite.

Note: The command does not work on output ports and ports with multidriven nets.

Example

The following command declares ports `in` and `in1` as logically opposite.

```
rc:/> dc::set_opposite [find / -port in] [find / -port in1]
```

dc::set_output_delay

```
set_output_delay
     [-clock clock]
     [-clock_fall] [-clock_rise]
     [-level_sensitive]
     [-network_latency_included]
     [-source_latency_included]
     [-rise] [-fall] [-max] [-min]
     [-add_delay] [-name name]
     delay {port|pin|port_bus}...
```

Constrains output and inout ports, port busses, and pins (that are valid end points) within the design relative to a clock edge. Calls the native `external_delay` command.

If you omit both the `-min` and `-max` options, the delay is assumed to apply for both setup and hold analysis.

If you omit both the `-rise` and `-fall` options, the delay applies for both the rising and falling edges of the output signal.

Note: This command also sets the `external_delays` attribute on the specified pins or ports.

Options and Arguments

`-add_delay`	Specifies to add the specified output delay information for the specified pins or ports. Use this option to capture the delay information if multiple paths lead to an output pin or port that are relative to different clocks or clock edges. If you omit this option, the specified delay information overwrites all existing output delays for the specified pins or ports.
`-clock` *`clock`*	Specifies the reference clock. The output delay is defined relative to this clock. If no clocks are specified, the output delay is with respect to all clocks.
`-clock_fall`	Specifies to use the falling edge of the reference clock as reference edge. *Default:* `-clock_rise`

`-clock_rise` Specifies to use the rising edge of the reference clock as reference edge.

Default: `-clock_rise`

`delay` Specifies the output delay value. This is a floating number.

`-fall` Specifies that the delay is measured with respect to the falling edge of the output signal.

`-level_sensitive` Specifies that the constraint goes to a level-sensitive latch.

By default, the constraint goes to an edge-triggered flip-flop.

`-max` Indicates that the specified delay applies for setup analysis.

`-min` Indicates that the specified delay applies for hold analysis. Hold analysis is not supported by RTL Compiler, but the value is written out with `write_sdc`.

`-name` *`name`* Associates a name with the specified timing constraint.

`-network_latency_included`

Specifies that the output delay includes the clock network latency values. This implies that the tool does not need to consider the clock network latency values when optiming this path.

Sets the `clock_network_latency_included` attribute to `true`.

{*`port`* | *`pin`* | *`port_bus`*}

Specifies the output and inout pins, ports, and port buses to which the specified output delay value applies.

`-rise` Specifies that the delay is measured with respect to the rising edge of the output signal.

`-source_latency_included`

Specifies that the output delay includes the clock source latency values. This implies that the tool does not need to consider the clock source latency values when optiming this path.

Sets the `clock_source_latency_included` attribute to `true`.

Example

The following commands set an output delay on output port `gck` with respect to clocks `clk1` and `clk2`. The example shows the effect of using the `-add_delay` option. When the second `set_output_delay` is specified without the `-add_delay` option, the first output delay is removed. However, when the third `set_input_delay` is specified with the `-add_delay` option, the delay information is stored.

```
rc:/> dc::set_output_delay -clock clk1 3 [find / -port gck]
rc:/> get_attr external_delays [find / -port gck]
/designs/test2/timing/external_delays/ou_del
rc:/> dc::set_output_delay -clock clk2 5 [find / -port gck]
   Removing external delay 'ou_del'.
Info    : Removed object. [TUI-58]
       : Removed external_delay '/designs/test2/timing/external_delays/ou_del'.
rc:/> get_attr external_delays  [find / -port gck]
/designs/test2/timing/external_delays/ou_del_1_1
rc:/> dc::set_output_delay -clock clk1 3 [find / -port gck] -add_delay
rc:/> get_attr external_delays  [find / -port gck]
/designs/test2/timing/external_delays/ou_del_1_1 /designs/test2/timing/
external_delays/ou_del
```

Related Information

Sets these attributes: clock_network_latency_included

clock_source_latency_included

external_delays

dc::set_path_adjust

```
set_path_adjust
     {{-from {port|pin|clock|instance}...
      |-rise_from {port|pin|clock|instance}...
      |-fall_from {port|pin|clock|instance}...}
     |{-to {port|pin|clock|instance}...
      |-rise_to {port|pin|clock|instance}...
      |-fall_to {port|pin|clock|instance}...}
     |{-through {port|pin|instance|net}...
      |-rise_through {port|pin|instance|net}...
      |-fall_through {port|pin|instance|net}...}...
     [-rise] [-fall]
     [-setup] [-hold]
     [-exception_name name]
     float
```

Specifies the delay constraint value by which the path has to be adjusted. It calls the RTL Compiler native command `path_adjust`.

Options and Arguments

`-exception_name` Specifies the name of the timing exception you want to create.

Default: `adj_n`

`-fall` Limits the path selection to those paths who have a falling edge at the end points.

`-fall_from {port | pin | clock | instance}`

Selects all paths whose signals have a falling edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-fall_through {port |pin | instance | net}`

Selects all paths whose signals have a falling edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-fall_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a falling edge at the specified end points.The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`float`

Specifies the delay constraint value, in picoseconds, by which the path has to be adjusted. A positive adjustment relaxes the clock constraint and a negative adjustment tightens it..

`-from {`*`port | pin | clock | instance`*`}`

Selects all paths that start from the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-hold`

Not supported by RTL Compiler.

Note: Although the tool does not perform any hold analysis, RTL Compiler does write out this constraint when the `write_sdc` command is given.

Default: setup

`-rise`

Limits the path selection to those paths who have a rising edge at the path end points.

`-rise_from {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge at the specified start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list.

`-rise_through {`*`port |pin | instance | net`*`}`

Selects all paths whose signals have a rising edge on the specified through points. The through points can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-rise_to {`*`port | pin | clock | instance`*`}`

Selects all paths whose signals have a rising edge on the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list of end points.

`-setup`

Applies the path adjust constraint for setup analysis.

Default: setup

`-through {`*`port |pin | instance | net`*`}`

Selects all paths that traverse through the specified points. The points to traverse can be ports of your design, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances or nets. Specify a Tcl list.

`-to {`*`port | pin | clock | instance`*`}`

Selects all paths that end at the specified end points. The end points can be output ports of your design, clock pins of flip-flops, clock objects, instances, or a combination of these. Specify a Tcl list of end points.

Example

The following example relaxes the timing constraints on all timing paths starting from port `in` by 0.5 time units.

```
rc:/> dc::set_path_adjust -from [find / -port in] 0.5
/designs/topmost/timing/exceptions/path_adjusts/adj_1
```

Related Information

Related command: path_adjust

dc::set_port_fanout_number

`set_port_fanout_number` *`integer port_list`*

Sets the number of external fanout pins for ports in the current design.

The number of pins is used (along with wire load models) to calculate capacitance and resistance of nets.

The number of pins minus one is equal to the fanout of this port outside the design. This value is set on the `external_wireload_fanout` attribute.

Options and Arguments

`integer`	Specifies the number of external fanout pins.
`port_list`	Specifies the ports for which the number is specified.

Example

The following command specifies 5 fanout pins for port `out1`. The tool stores the value 4 (5-1) on the `external_wireload_fanout` port attribute for `out1`.

```
rc:/> dc::set_port_fanout_number 5 out1
rc:/> get_attr external_wireload_fanout out1
4
```

Related Information

Sets this attribute: external_wireload_fanout

dc::set_timing_derate

```
set_timing_derate
     [-cell_delay] [-cell_check] [-net_delay]
     [-early] [-late] [-clock] [-data]
     float {libcell | instance}...
```

Models the on-chip variation (OCV) or delay change across a die due to variations in manufacturing process, voltage, and temperature that must be taken into account during timing analysis.

Note: Omitting both the `-cell_check` and `-cell_delay` options results in the derating factor being applied to both delay and constraint arcs.

Options and Arguments

`-cell_check`	Applies the derating factor to the constraint arcs.
`-cell_delay`	Applies the derating factor to the cell delay arcs.
`-clock`	Not supported by RTL Compiler.
`-data`	Applies the derating factor to data paths.
`-early`	Not supported by RTL Compiler.
`-late`	Applies the derating factors for the late mode.
`-net_delay`	Applies the derating factor to net delays. **Note:** This option is only available in the physical flows.
`float`	Specifies the derating value.
`instance`	Not supported by RTL Compiler.
`libcell`	Specifies the library cells to which the derating factor applies.

Examples

- The following command scales the delay of libcell `DFFHQX1` by `2.3`.

  ```
  dc::set_timing_derate -cell_delay 2.3 [find /lib*/* -libcell DFFHQX1]
  ```

  This command is equivalent to the following native command:

  ```
  set_attribute cell_delay_multiplier 2.3 [find /lib*/* -libcell DFFHQX1]
  ```

- The following command scales the constraint arcs of libcell `DFFHQX1` by `3.1`.

  ```
  dc::set_timing_derate -cell_check 3.1 [find /lib*/* -libcell DFFHQX1]
  ```

This command is equivalent to the following native command:

```
set_attribute constraint_multiplier 3.1 [find /lib*/* -libcell DFFHQX1]
```

dc::set_time_unit

```
set_time_unit
     {-picoseconds | -nanoseconds } [float]
```

Specifies the time unit used inside the SDC file or as part of the `dc::` command

Note: Although supported by the `read_sdc` command, it is *not* an official SDC command.

Options and Arguments

`float`	Indicates the number that any timing value in subsequent SDC commands should be multiplied with.
`-nanoseconds`	Indicates that the timing values are specified in nanoseconds.
`-picoseconds`	Indicates that the timing values are specified in picoseconds.

Examples

- The following command indicates that each timing unit in the SDC file equals 3 picoseconds.

  ```
  dc::set_time_unit -picoseconds 3
  ```

dc::set_unconnected

```
set_unconnected
     port_list
```

Edits the netlist by disconnecting the output port.

Note: Although supported by the `read_sdc` command, it is *not* an official SDC command.

Options and Arguments

port_list	Specifies the names of the ports that must be disconnected.

dc::set_units

```
set_units
     [-capacitance string] [-resistance string]
     [-voltage string] [-current string]
     [-time string] [-power string]
```

Specifies the unit scale used in the SDC file.

Note: This command was added in SDC 1.7.

Options and Arguments

`-capacitance string`

Specifies the capacitance unit. String can have the following format:

`[float] {fF | pF | nF | uF | mF | kF | MF}`

`-current string`	Not supported by RTL Compiler.
`-power string`	Not supported by RTL Compiler.
`-resistance string`	Not supported by RTL Compiler.
`-time string`	Specifies the time unit. String can have the following format: `[float] {fs | ps | ns | us | ms | ks | Ms}`
`-voltage string`	Not supported by RTL Compiler.

Examples

- The following command specifies that one load unit is 0.5pF.

  ```
  dc::set_units capacitance 0.5fF
  ```

- The following command specifies that one time unit is 0.78ps.

  ```
  dc::set_units time 0.78ps
  ```

dc::set_wire_load_mode

```
set_wire_load_mode {default | top | enclosed| segmented}
```

Sets the `wireload_mode` root attribute with the specified value.

Options and Arguments

`default`	Uses the default wire-load model from the library
`enclosed`	Uses the wire-load model of the smallest block that fully encloses the net to compute the load of the net. Hierarchical boundary pins are not counted as fanouts.
`segmented`	Divides nets that cross hierarchical boundaries into segments with one segment for each level of hierarchy. Separate load values are computed for each segment (counting the hierarchical boundary pins as individual fanouts) and the load values are added together.
`top`	Uses the wire-load model of the top-level design for all nets in all subdesigns. Hierarchical boundary pins are not counted as fanouts.

Example

```
rc:/> dc::set_wire_load_mode top
```

Related Information

Sets this attribute: wireload_mode

dc::set_wire_load_model

```
set_wire_load_model
     -name {auto_select | custom_wireload | none | inherit}
     [-library library] [-min] [-max]
     {design | subdesign | instance | port}...
```

Forces RTL Compiler to use a specific wire load model for the given design, subdesigns, hierarchical instance or port.

Sets the `force_wireload` attribute if the specified object is a design, subdesign or a hierarchical instance. If the specified object is a hierarchical instance, the attribute on the corresponding subdesign is set.

Sets the `external_wireload_model` attribute if the specified object is a port.

Options and Arguments

`auto_select`	Automatically selects wire-load models according to the wire-load selection table or default wire-load model in the technology library. **Note:** This value can only be applied for the design, subdesigns, or hierarchical instances.
`custom_wireload`	Forces RTL Compiler to use the specified custom wire-load model. Specify the hierarchical path to the wire-load model to be used. You can obtain the path using the `find` command.
`design` | *`subdesign`* | *`instance`* | *`port`*	Specifies the design, subdesign, hierarchical instance, or port for which you want to specify the wire load model. If no object is specified, the wire load model is applied to the top-level design.
`inherit`	Causes the same behavior as `auto_select`. **Note:** This value can only be applied for the design, subdesigns, or hierarchical instances.
`-library` *`library`*	Specifies the technology library to which the selection group belongs.
`-max`	Not supported by RTL Compiler.

`-min`	Not supported by RTL Compiler.
`none`	Prevents the use of any wire-load models. **Note:** This value can only be applied for the design, subdesigns, or hierarchical instances.

Examples

The following example forces the use of wire load model `10x10` for design `test_top`:

```
rc:/> dc::set_wire_load_model -name [find /libraries -wireload "10x10"] \
[find / -design test_top]
```

Related Information

Specifying a Wire-load Model

Sets these attributes:	force_wireload
	external_wireload_model
Related attributes:	wireload_mode
	wireload_selection

dc::set_wire_load_selection_group

```
set_wire_load_selection_group
     [-library library] [-min] [-max]
     [-object_list string] {default | none | table}
```

Indicates whether to use a wire-load selection table to choose default wire-load models for blocks based on their cell areas.

Sets the `wireload_selection` root attribute with the specified value.

Options and Arguments

`default`	Reverts the environment to the default settings. RTL Compiler behaves as if the attribute had never been set.
`-library` *`library`*	Specifies the technology library to which the selection group belongs. If this option is omitted, the command searches for the specified wire-load selection table in the list of the libraries loaded. If the same table name appears in mulitple libraries, the selection table from the earlier loaded library is used.
`-max`	Not supported by RTL Compiler.
`-min`	Not supported by RTL Compiler.
`none`	Specifies not to perform automatic wire-load selection by area. The only wire-load models that will be used are the ones that are set with the `force_wireload` attribute on individual modules or the default wireload model specified in the library.
`-object_list`	Not supported by RTL Compiler.
`table`	Specifies the wire-load selection table to use. Specify the hierarchical path to the wire-load selection table to be used. You can obtain the path using the `find` command.

Example

Some libraries contain multiple selection tables, such as for different numbers of metal layers). In such cases you can indicate which `wireload_selection` table should be used:

```
dc::set_wire_load_selection_group [find / -wireload_selection "4_layer"]
```

Related Information

Specifying Wire-load Models

Sets this attribute: wireload_selection

Related attributes: force_wireload

wireload_mode

dc::sizeof_collection

```
sizeof_collection list
```

Returns the length of the collection passed.

Options and Arguments

collection	Specifies a list whose length needs to be returned.

Example

```
rc:/> dc::sizeof_collection [dc::get_cells *]
4
```

Index

A

B

C

D

S

T

U

V

W