Design Compiler™
Reference Manual: Constraints and Timing

Version 2002.05, June 2002

Comments?
E-mail your comments about Synopsys documentation to doc@synopsys.com

SYNOPSYS®

Printed in the U.S.A.

Document Order Number: 31734-000 MA
Design Compiler Reference Manual: Constraints and Timing, v2002.05

Contents

Figures

Tables

Preface

This preface includes the following sections:

What's New in This Release

This section describes the new features, enhancements, and changes included in Design Compiler version 2002.05. Unless otherwise noted, you can find additional information about these items in the Design Compiler documentation set, version 2002.05.

New Features

Design Compiler version 2002.05 includes the following new features:

- Improved Usability and Delay Quality of Results (QOR) With DC Ultra

  With this release, you can invoke a single dc_shell command, `compile_ultra`, for the complete DC Ultra flow. The `compile_ultra` command provides an automated flow that encapsulates DC Ultra data-path and timing optimization, thus significantly improving QOR.

  For details about this new feature, see the *Design Compiler User Guide*.

- DC Ultra data-path optimization

  This release introduces a DC Ultra data-path optimization flow. The new optimization flow encompasses and exceeds the functionality of the `partition_dp` and `transform_csa` commands because, in addition to creating optimized data-path blocks, it evaluates various resource-sharing and data-path architectures, returning the best result for the design constraints.

This new technique—creating highly optimized data-path blocks with shared arithmetic components after resource sharing—produces better timing, CPU time, and area in many cases.

For details about this new feature, see the *Design Compiler User Guide*.

- Automatic Extraction and Optimization of Finite State Machines

  The 2002.05 release introduces an automatic FSM extraction and compilation flow. The new automatic FSM flow requires minimal user input and delivers area, timing, and runtime benefits.The automatic FSM flow requires a DC Ultra license and can be applied to Verilog and VHDL designs, Synopsys state tables (.st files), and already compiled FSM designs in Synopsys database format (.db files).

  The existing manual FSM flow is still supported.

  For details about the FSM optimization methodologies, see the *Design Compiler Reference Manual: Optimization and Timing Analysis*.

- Support for Interface Logic Models in Synthesis

  Design Compiler now provides the capability to create interface logic models and use them in the optimization flow. Interface logic models allow designers to perform top-level optimization on large designs by reducing the memory requirements of subblocks. The interface logic model of a compiled block contains the mapped interface logic of the block and any logic that affects interface performance. Interface logic models preserve timing accuracy by retaining all interface logic and removing only register-to-register paths. Replacing the subblocks with their smaller interface logic models might alleviate capacity problems and improve compile times for the top-level design.

For details about this new feature, see the *Design Compiler User Guide*.

- Internal Pin Support Within Complex Models

  In the 2002.05 release, internal pins of extracted timing models and complex cells can serve as clock source pins. The dcsh `find` command and the dctcl `get_pins` command return the names of internal pins, allowing the application of clock constraints from outside the model.

  For details about this new feature, see the *Design Compiler Reference Manual: Constraints and Timing*.

Enhancements

Design Compiler version 2002.05 includes the following enhancements:

- Runtime Enhancements During Compile

  In the 2002.05 release of Design Compiler, new algorithms have been introduced to the delay optimization and area recovery phases of the compile process. These algorithms enhance the mapping and optimization of logic during these phases and provide faster compile time in the new release.

- Propagation of Ideal Nets

  This release introduces ideal networks as an extension to the existing ideal net capability. Previously, the `ideal_net` attribute applied only to the assigned net. In the 2002.05 release, you can specify that the `ideal_net` attribute propagate through the network automatically. Defining as ideal nets certain high fanout nets that you intend to synthesize separately, such as

scan-enable and reset nets, can reduce runtime by avoiding unnecessary timing steps and unwanted design changes during optimization.

For details about this enhancement, see the *Design Compiler Reference Manual: Constraints and Timing*.

- Automatic Ungrouping of Blocks Along the Critical Path

  The auto-ungrouping feature improves optimization of hierarchical designs by automatically ungrouping certain hierarchies, thus enabling synthesis algorithms to work on a larger block. In the 2002.05 release, this capability is enhanced to perform ungrouping on the blocks along the critical path. Suitable candidates include hierarchies containing a few cells and hierarchies containing a large number of nets driven by constants. Because these hierarchies are automatically ungrouped as part of optimization, both timing and area results can be improved.

  For details about this enhancement, see the *Design Compiler User Guide*.

- Register Retiming Support for Clock-Gated Designs

  In the 2002.05 release, register retiming in DC Ultra is enhanced to support clock-gated designs. With this feature, the optimization benefits of register retiming can be achieved for low-power designs. DC Ultra provides the capability to retime registers and to insert new pipeline stages.

- Support for Conditional Arcs

Design Compiler now supports the `when` construct in library cells, in conjunction with case analysis and logic constants. The timing arc corresponding to each `when` condition is used instead of the longest arc in all cases. Thus, timing results are now more accurate.

- Improved Support for Bidirectional Ports

  In the 2002.05 release, you can enable and disable bidirectional feedback loops. This enhancement allows you to disable erroneous timing paths in bidirectional feedback loops.

  For details about this enhancement, see the *Design Compiler Reference Manual: Optimization and Timing Analysis*.

- Input Port Isolation

  The `set_isolate_ports` command now supports input port isolation. You isolate input ports when they drive one or several pins on one or more multi-input cells, thus improving accuracy of timing models. The isolation logic is placed directly after the specified input port.

  For details about this enhancement, see the *Design Compiler Reference Manual: Constraints and Timing*.

- Generated Clocks on Output Ports

  You can now use the `create_generated_clock` command to create a generated clock at an output pin of a design or model when a clock drives output pins.

- Input Pin Transition for Clocks

  You can now use `set_input_transition` on clock ports to add transition information when the clock is also used as data.

Changes

Design Compiler version 2002.05 includes the following change:

- Improved Mapping for Sequential Cells

  Changes in the 2002.05 release improve timing and area results when the tool infers registers with synchronous control inputs. The changes in the sequential cell mapper and modeler used by Design Compiler provide better matching between the HDL code and inferred sequential cells.

Known Limitations and Resolved STARs

Information about known problems and limitations, as well as about resolved Synopsys Technical Action Requests (STARs), is available in the *Design Compiler Release Notes* in SolvNet.

To see the *Design Compiler Release Notes,*

1. Go to the Synopsys Web page at http://www.synopsys.com and click SolvNet.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, click New Synopsys User Registration.)
3. Click Release Notes in the Main Navigation section, find the 2002.05 Release Notes, then open the *Design Compiler Release Notes.*

About This Manual

The *Design Compiler Reference Manual: Constraints and Timing* describes concepts, commands, and use for Design Compiler product constraints. This manual is not a user guide but includes examples.

Some features shown (such as netlist formats) are options that you can purchase separately.

Audience

This manual is intended for logic designers and engineers who use the Synopsys synthesis tools to design ASICs, ICs, and FPGAs. Knowledge of high level techniques, a hardware description language, such as VHDL or Verilog is required. A working knowledge of UNIX is assumed.

Related Publications

For additional information about Design Compiler, see

- Synopsys Online Documentation (SOLD), which is included with the software for CD users or is available to download through the Synopsys Electronic Software Transfer (EST) system
- Documentation on the Web, which is available through SolvNet at http://solvnet.synopsys.com
- The Synopsys MediaDocs Shop, from which you can order printed copies of Synopsys documents, at http://mediadocs.synopsys.com

You might also want to refer to the documentation for the following related Synopsys products:

- Design Analyzer
- Design Vision
- DesignWare Components
- DFT Compiler
- Floorplan Manager
- FPGA Compiler II
- HDL Compiler (Presto Verilog)
- Module Compiler
- PrimeTime
- Power Compiler
- VHDL Compiler

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
Courier bold	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* [*`pin2 ... pinN`*]
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet,

1. Go to the SolvNet Web page at http://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, click New Synopsys User Registration.)

If you need help using SolvNet, click SolvNet Help in the Support Resources section.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (Synopsys user name and password required), then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to support_center@synopsys.com.
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

1

Basic Constraint Concepts

Constraints are declarations that define your design's goals in measurable circuit characteristics such as timing, area, and capacitance. Without constraints, the Design Compiler tool cannot effectively optimize your design.

In this chapter, you will learn about the following:

- Design Compiler Constraint Types
- Design Rule Constraints
- Optimization Constraints
- Managing Constraint Priorities
- Constraint Reports

Design Compiler Constraint Types

When Design Compiler optimizes your design, it uses two types of constraints:

Design rule constraints

These are implicit constraints; the technology library defines them. These constraints are requirements for a design to function correctly, and they apply to any design using the library. You can make these constraints more restrictive than optimization constraints.

Optimization constraints

These are explicit constraints; you define them. Optimization constraints apply to the design on which you are working for the duration of the dc_shell session and represent the design's goals. They must be realistic.

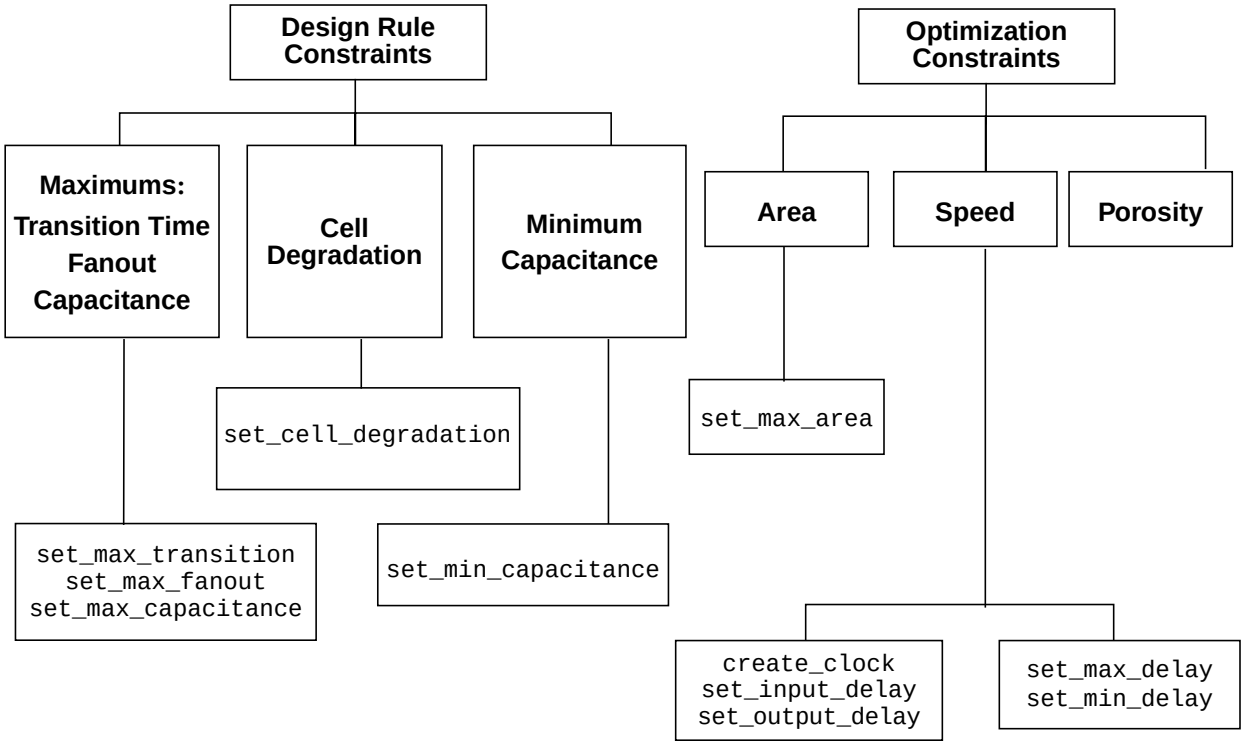
Design Compiler tries to meet both design rule constraints and optimization constraints, but design rule constraints take precedence.

Design Compiler calculates two cost functions: one for design rule constraints and one for optimization constraints. During gate-level optimization, Design Compiler reports the value of each cost function whenever a change is made to the design.

You specify constraints interactively on the command line or in a constraints file.

Figure 1-1 shows the major Design Compiler design rule constraints and optimization constraints and the dc_shell interface commands to set the constraints.

Figure 1-1 Major Design Compiler Constraints



Note:

If you use the Power Compiler tool from Synopsys, `max_dynamic_power` and `max_leakage_power` are optimization constraints. To use these constraints, you need a Power Optimization license and supporting libraries characterized for power.

Design Rule Constraints

Design rule constraints reflect technology-specific restrictions your design must meet in order to function as intended.

Design rules constrain the nets of a design but are associated with the pins of cells from a technology library. Most technology libraries specify default design rules. Design Compiler cannot violate design rule constraints, even if it means violating optimization constraints (delay and area goals). You can apply more restrictive design rules, but you cannot apply less restrictive ones.

The design rule constraints comprise

- Maximum transition time
- Maximum fanout
- Minimum and maximum capacitance
- Cell degradation

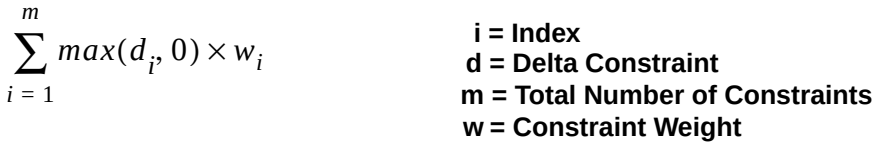
You cannot remove `max_transition`, `max_fanout`, `max_capacitance`, and `min_capacitance` attributes set in a technology library, because they are requirements for the technology, but you can set more-restrictive values. If both implicit and explicit values are set on a design or a port, the more restrictive value applies. You can remove values you have set.

Design Rules Cost Function

Design Compiler helps you fix design rule violations. If there are multiple violations, Design Compiler tries to fix the violation with the highest priority first. When possible, it also evaluates and selects alternatives that reduce violations of other design rules. Design Compiler uses the same approach with delay violations.

Figure 1-2 shows the design rule cost function equation.

Figure 1-2 Design Rule Cost Equation



In a compilation report, the DESIGN RULE COST field reports the cost function for the design rule constraints on the design.

Maximum Transition Time

Maximum transition time is a design rule constraint.

The maximum transition time for a net is the longest time required for its driving pin to change logic values. Many technology libraries contain restrictions on the maximum transition time for a pin, creating an implicit transition time limit for designs using that library.

Design Compiler attempts to make the transition time of each net less than the `max_transition` value. For example, it can be done by buffering the output of the driving gate.

Transition times on nets are computed by use of timing data from the technology library.

To change or add to the implicit transition time values from a technology library, use the `set_max_transition` command.

If both a library `max_transition` and a design `max_transition` attribute are defined, Design Compiler tries to meet the smaller (more restrictive) value.

If your design uses multiple technology libraries and each has a different default `max_transition` value, Design Compiler uses the smallest `max_transition` value globally across the design.

Defining Maximum Transition Time

To set a maximum transition time attribute on ports or designs, use the `set_max_transition` command, described in Chapter 2, “Constraining Designs.”

Maximum Fanout

Maximum fanout is a design rule constraint.

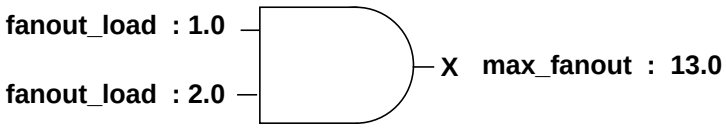
Most technology libraries place fanout restrictions on driving pins, creating an implicit fanout constraint for every driving pin in designs using that library.

You can set a more conservative fanout constraint on an entire library or define fanout constraints for specific pins in the library description of an individual cell.

If a library fanout constraint exists and you specify a `max_fanout` attribute, Design Compiler tries to meet the smaller (more restrictive) value.

Design Compiler models fanout restrictions by associating a `fanout_load` attribute with each input pin and a `max_fanout` attribute with each output (driving) pin on a cell. Figure 1-3 shows these fanout attributes.

Figure 1-3 fanout_load and max_fanout Attributes



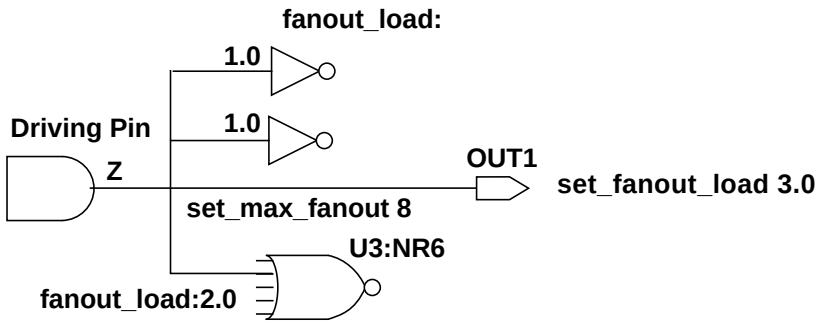
To evaluate the fanout for a driving pin (such as X in Figure 1-3), Design Compiler calculates the sum of all the `fanout_load` attributes for inputs driven by pin X and compares that number with the number of `max_fanout` attributes stored at the driving pin X.

- If the sum of the fanout loads is not more than the `max_fanout` value, the net driven by X is valid.
- If the net driven by X is not valid, Design Compiler tries to make that net valid, perhaps by choosing a higher-drive component.

Maximum Fanout Calculation Example

Figure 1-4 shows how maximum fanout is calculated.

Figure 1-4 Calculation of Maximum Fanout

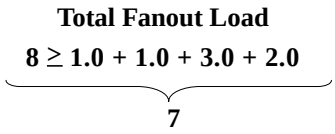


You can set a maximum fanout constraint on every driving pin and input port as follows:

```
dc_shell> set_max_fanout 8 find(design, ADDER)
```

To check whether the maximum fanout constraint is met for driving pin Z, Design Compiler compares the specified `max_fanout` attribute against the fanout load.

In this case, the design meets the constraints.



The fanout load imposed by a driven cell (U3) is not necessarily 1.0. Library developers can assign higher fanout loads (for example, 2.0) to model internal cell fanout effects.

You can also set a fanout load on an output port (OUT1) to model external fanout effects.

Fanout load is a dimensionless number, not a capacitance. It represents a numerical contribution to the total effective fanout.

Defining Maximum Fanout

To set a `max_fanout` attribute on specified objects, use the `set_max_fanout` command, described in Chapter 2, “Constraining Designs."

To set the expected fanout load value for the specified output ports, use the `set_fanout_load` command, described in Chapter 2, “Constraining Designs.”

Maximum Capacitance

Maximum capacitance is a design rule constraint.

The maximum capacitance design rule constraint allows you to control the capacitance of nets directly. (The design rule constraints `max_fanout` and `max_transition` limit the actual capacitance of nets indirectly.) You can limit capacitance directly for the nets attached to specified ports or to all the nets in the design. The `max_capacitance` constraint operates similarly to

max_transition, but the cost is based on the total capacitance of the net, rather than the transition time. The max_capacitance attribute functions independently, so you can use it with max_fanout and max_transition.

The maximum capacitance constraint has priority over the cell degradation constraint.

If both a library max_capacitance attribute and a design max_capacitance attribute exist, Design Compiler tries to meet the smaller (more restrictive) value.

Defining Maximum Capacitance

To set a maximum capacitance attribute on specified objects, use the set_max_capacitance command, described in Chapter 2, “Constraining Designs.”

Minimum Capacitance

Minimum capacitance is a design rule constraint.

Some technology libraries specify minimum capacitance.

The min_capacitance design rule specifies the minimum load a cell can drive. It specifies the lower bound of the range of loads with which a cell has been characterized to operate. During optimization, Design Compiler ensures that the load driven by a cell meets the minimum capacitance requirement for that cell. When a violation occurs, Design Compiler fixes the violation by sizing the driver.

You can use the `min_capacitance` design rule with the existing design rules. The `min_capacitance` has higher priority than the maximum transition time, maximum fanout, and maximum capacitance constraints.

You can set a minimum capacitance for nets attached to input ports to determine whether violations occur for driving cells at the input boundary. If violations are reported after compilation, you can fix the problem by recompiling the module driving the ports. Use `set_min_capacitance` for input or inout ports; you cannot set minimum capacitance on a design.

If library `min_capacitance` and design `min_capacitance` attributes both exist, Design Compiler tries to meet the larger (more restrictive) value.

Defining Minimum Capacitance

To set a minimum capacitance for nets attached to input ports, use the `set_min_capacitance` command, described in Chapter 2, “Constraining Designs.”

Cell Degradation

Cell degradation is a design rule constraint.

Some technology libraries contain cell degradation tables. The tables list the maximum capacitance that can be driven by a cell as a function of the transition times at the inputs of the cell.

The cell_degradation design rule specifies that the capacitance value for a net is less than the cell degradation value. To fix cell degradation design rules, set the `compile_fix_cell_degradation` variable to true.

The `cell_degradation` design rule can be used with other design rules, but the `max_capacitance` design rule has a higher priority than the `cell_degradation` design rule. If the `max_capacitance` rule is not violated, applying the `cell_degradation` design rule does not cause it to be violated.

To specify cell degradation, use the `set_cell_degradation` command, described in Chapter 2, “Constraining Designs.”

Managing Design Rule Constraint Priorities

By default, design rule constraints have a higher optimization priority than optimization constraints. You can change that priority, however; see “Managing Constraint Priorities” on page 1-22.

Precedence of Design Rule Constraints

Design Compiler follows this descending order of precedence when it tries to resolve conflicts among design rule constraints (see Table 1-2 on page 1-22):

1. Minimum capacitance
2. Maximum transition
3. Maximum fanout
4. Maximum capacitance
5. Cell degradation

The following details apply to the precedence of design rule constraints:

- Maximum transition has precedence over maximum fanout. If a maximum fanout constraint is not met, investigate the possibility of a conflicting maximum transition constraint. Design Compiler does not make a transition time worse in order to fix a maximum fanout violation.
- Maximum fanout has precedence over maximum capacitance.
- Design Compiler calculates transition time for a net in two ways, depending on the library.
  - For libraries using the CMOS delay model, Design Compiler calculates the transition time by using the drive resistance of the driving cell and the capacitive load on the net.
  - For libraries using a nonlinear delay model, Design Compiler calculates the transition time by using table lookup and interpolation. This is a function of capacitance at the output pin and of the input transition time.
- The `set_driving_cell` and `set_drive` commands behave differently, depending on your technology library.
  - For libraries using the CMOS delay model, drive resistance is a constant. In this case, the `set_drive -drive_of` command and the `set_driving_cell` command give the same result.
  - For libraries using a nonlinear delay model, the `set_driving_cell` command calculates the transition time dynamically, based on the load from the tables. The `set_drive` command returns one value when the command is issued and uses a value from the middle of the range in the tables for load.

Both the `set_driving_cell` and the `set_drive` commands affect the port transition delay. The `set_driving_cell` command places the design rule constraints, annotated with the driving cell, on the affected port.

The `set_load` command places a load on a port or a net. The units of this load must be consistent with your technology library. This value is used for timing optimizations, not for maximum fanout optimizations.

Design Rule Scenarios

Typical design rule scenarios are

- `set_max_fanout` and `set_max_transition` commands
- `set_max_fanout` and `set_max_capacitance` commands

Typically, a technology library specifies a default `max_transition` or `max_capacitance`, but not both. To achieve the best result, do not mix `max_transition` and `max_capacitance`.

Summary of Design Rule Commands and Objects

Table 1-1 summarizes the design rule commands and the objects on which to set them.

Table 1-1 Design Rule Command and Object Summary

Command	Object
`set_max_fanout`	Input ports or designs
`set_fanout_load`	Output ports
`set_load`	Ports or nets

Table 1-1 Design Rule Command and Object Summary (Continued)

Command	Object
`set_max_transition`	Ports or designs
`set_cell_degradation`	Input ports
`set_min_capacitance`	Input ports

Optimization Constraints

Optimization constraints represent speed, area, and porosity design goals and restrictions that you want but that might not be crucial to the operation of a design. Speed (timing) constraints have higher priority than area and porosity.

By default, optimization constraints are secondary to design rule constraints. (That priority can be changed, however; see “Managing Constraint Priorities” on page 1-22.)

The optimization constraints comprise

- Timing constraints (performance and speed)
  - Input and output delays (synchronous paths)
  - Minimum and maximum delay (asynchronous paths)
- Maximum area (number of gates)
- Minimum porosity (routability)

Optimization Cost Function

During the first phase of mapping, Design Compiler works to reduce the optimization cost function. Design Compiler evaluates this function to determine whether a change to the design improves the cost.

The full optimization cost function takes into account the following components, listed in order of importance. Not all components are active on all designs.

1. Maximum delay cost
2. Minimum delay cost
3. Maximum area cost
4. Minimum porosity

Design Compiler evaluates cost function components independently in order of importance and accepts an optimization move if it decreases the cost of one component without increasing more-important costs. For example,

- An optimization move that improves maximum delay cost is always accepted
- A minimum porosity constraint is accepted if it does not increase maximum area

Optimization stops when all costs are zero or no further improvements can be made to the cost function.

Timing Constraints

When defining timing constraints you should consider that your design has synchronous paths and asynchronous paths.

Synchronous paths are constrained by specifying clocks in the design. Use the `create_clock` command to specify a clock. After specifying the clocks, it is recommended you also specify the input and output port timing specifications. Use the `set_input_delay` and `set_output_delay` commands.

For additional information, refer to Chapter 4, “Specifying Clocks and Clock Networks,” and Chapter 5, “Describing Logic Functions and Signal Interfaces.”

Asynchronous paths are constrained by specifying minimum and maximum delay values. Use the `set_max_delay` and `set_min_delay` commands to specify these point-to-point delays. The calculation of minimum and maximum delays are described in the following sections.

Maximum Delay

Maximum delay is an optimization constraint.

Design Compiler contains a built-in static timing analyzer for evaluating timing constraints. A static timing analyzer calculates path delays from local gate and interconnect delays but does not simulate the design. The Design Compiler timing analyzer performs critical path tracing to check minimum and maximum delays for every timing path in the design. The most critical path is not necessarily the longest combinational path in a sequential design, because paths can be relative to different clocks at path startpoints and endpoints.

The timing analyzer calculates minimum and maximum signal rise and fall path values based on the timing values and environmental information in the technology library.

Cost Calculation. Maximum delay is usually the most important portion of the optimization cost function. The maximum delay optimization cost guides Design Compiler to produce a design that functions at the speed you want.

The maximum delay cost includes multiple parts. Figure 1-5 shows the maximum delay cost equation.

Figure 1-5 Cost Calculation for Maximum Delay

$$\sum_{i=1}^{m} v_i \times w_i$$

i = Index
v = worst violation
m = number of path groups
w = weight

Maximum delay target values for each timing path in the design are automatically determined after considering clock waveforms and skew, library setup times, external delays, multicycle or false path specifications, and `set_max_delay` commands. Load, drive, operating conditions, wire load model, and other factors are also taken into account.

The maximum delay cost is affected by how paths are grouped by the `group_path` and `create_clock` commands.

- If only one path group exists, the maximum delay cost is the cost for that group: the amount of the worst violation multiplied by the group weight.
- If multiple path groups exist, the costs for all the groups are added together to determine the maximum delay cost of the design. The group cost is always zero or greater.

```
Delta = max(delta(pin1), delta(pin2), ... delta(pinN))
```

Minimum Delay

Minimum delay is an optimization constraint, but Design Compiler fixes the minimum delay constraint when it fixes design rule violations.

Minimum delay constraints are set explicitly with the `set_min_delay` command or set implicitly due to hold time requirements.

The minimum delay to a pin or port must be greater than the target delay.

Design Compiler considers the minimum delay cost only if the `set_fix_hold` command is used.

If `fix_hold` is not specified on any clocks, the minimum delay cost is not considered during compilation. If `fix_hold` or `min_delay` is specified, the minimum delay cost is a secondary optimization cost.

`set_min_delay`

Defines a minimum delay for timing paths in the design.

`set_fix_hold`

Directs Design Compiler to fix hold violations at registers during compilation. You can override the default path delay for paths affected by `set_fix_hold`, by using the `set_false_path` or `set_multicycle_path` commands.

Hold time violations are fixed only if the `fix_hold` command is applied to related clocks.

Cost Calculation. The minimum delay cost for a design is different from the maximum delay cost. The minimum delay cost is not affected by path groups, and all violations contribute to the cost. Figure 1-6 shows the minimum delay cost equation.

Figure 1-6 Cost Calculation for Minimum Delay

$$\sum_{i=1}^{m} v_i$$

i = index
m = number of paths affected by set_min_delay or set_fix_hold
v = minimum delay violation max(0,required_path_delay - actual_path_delay)

The minimum delay cost function has the same delta for single pins or ports and multiple pins or ports.

```
Delta = min_delay - minimum delay(pin or port)
```

Defining Minimum Delay. For information about the `set_min_delay` and `set_fix_hold` commands, see Chapter 2, “Constraining Designs."

Maximum Area

Maximum area is an optimization constraint.

Maximum area represents the number of gates in the design, not the physical area the design occupies.

Usually the area requirements for the design are stated as the smallest design that meets the performance goal. Defining a maximum area directs Design Compiler to optimize the design for area after timing optimization is complete.

The `set_max_area` command specifies the maximum allowable area for the current design. Design Compiler computes the area of a design by adding the areas of each component on the lowest level of the design hierarchy (and the area of the nets).

Design Compiler ignores the following components when it calculates circuit area:

- Unknown components
- Components with unknown areas
- Technology-independent generic cells

The area of a cell (component) is technology-dependent and obtained from the technology library.

Cost Calculation

The maximum-area cost equation is

```
cost = max (0, current area - max_area)
```

Defining Maximum Area

To specify a maximum-area goal, use the `set_max_area` command described in Chapter 2, “Constraining Designs.”

Minimum Porosity

Minimum porosity is an optimization constraint.

Porosity is the ratio of routing track area to cell area. You can optimize the porosity to increase the routability of an ASIC design. One measure of the routability of an ASIC design is the number and area of tracks available for routing over cells. In standard cell

technologies, the routing tracks are called feedthroughs. In gate array technologies, the routing tracks are called over-the-cell routing tracks. The ratio of the routing track area to the cell area is called the porosity of the cell.

Defining a minimum porosity directs Design Compiler to improve the design for porosity after timing optimization is complete if you use the `-add_porosity` option of the `compile` command.

If the technology library defines `default_min_porosity` and you explicitly set a value, the more restrictive value applies. If the design uses libraries with different `default_min_porosity` attributes, the most restrictive value applies.

Cost Calculation

The minimum-porosity cost equation is

```
cost = max (0, min_porosity - current_porosity)
```

Design Compiler computes the porosity of a design by dividing the sum of the routing track areas of each component on the lowest level of the design hierarchy by the sum of all components' areas. Design Compiler ignores the following components when calculating circuit area:

- Unknown components
- Components with unknown routing track areas
- Technology-independent generic cells

Setting Minimum Porosity

To set a target porosity value, use the `set_min_porosity` command, described in Chapter 2, “Constraining Designs.”

Managing Constraint Priorities

During optimization, Design Compiler uses a cost vector to resolve any conflicts among competing constraint priorities. Table 1-2 shows the default order of priorities.

Table 1-2 Constraints Default Cost Vector

Priority (descending order)	Notes
connection classes	
multiple_port_net_cost	
min_capacitance	Design Rule Constraint
max_transition	Design Rule Constraint
max_fanout	Design Rule Constraint
max_capacitance	Design Rule Constraint
cell_degradation	Design Rule Constraint
max_delay	Optimization Constraint
min_delay	Optimization Constraint
power	Optimization Constraint
area	Optimization Constraint
cell count	

Table 1-2 shows that, by default, design rule constraints have priority over optimization constraints. However, you can reorder the priorities of the constraints listed in **bold** type, by using the `set_cost_priority` command. For example, the following are circumstances under which you might want to move the optimization constraint `max_delay` ahead of the maximum design rule constraints.

- In many technology libraries, the only significant design rule violations that cannot be fixed without hurting delay are overconstrained nets, such as input ports with large external loads or around logic marked `dont_touch`. Placing `max_delay` ahead of the design rule constraints in priority allows these design rule constraint violations to be fixed in a way that does not hurt delay. Design Compiler might, for example, resize the drivers in another module.
- In compilation of a small block of logic, such as an extracted critical region of a larger design, the possibility of overconstraints at the block boundaries is high. In this case, design rule fixing might better be postponed until the small block has been regrouped into the larger design.

The syntax is

```
set_cost_priority [-default] [-delay]  cost_list
```

`-default`

Directs Design Compiler to use its default priority, as shown in Table 1-2 on page 1-22.

`-delay`

Specifies that `max_delay` has higher priority than the maximum design rule constraints.

cost_list

Specifies the order of priority (listing the highest first) of the following costs: max_delay, min_delay, max_transition, max_fanout, max_capacitance, cell_degradation, and max_design_rules.

Note:

Use of the cost_list option requires a DC Ultra license.

Examples

To prioritize max_delay ahead of the maximum design rule constraints, enter

```
dc_shell> set_cost_priority -delay
```

To assign top priority to max_capacitance, max_delay, and max_fanout—in that order—enter

```
dc_shell> set_cost_priority {max_capacitance max_delay max_fanout}
```

Design Compiler assigns any costs you do not list to a lower priority than the costs you do list. This example does not list max_transition, cell_degradation, or min_delay, so Design Compiler assigns them priority following max_fanout.

If you specify set_cost_priority more than once on a design, Design Compiler uses the most recent setting.

Note:

See the *Design Compiler Reference Manual: Optimization and Timing Analysis* for information about the final combinational mapping flow. In that flow, delay optimization occurs first—before

design rule fixing. Despite the order of events within the mapping flow, design rule constraint violations are always fixed, even if doing so worsens delays.

Constraint Reports

Reports provide information about design rule and optimization constraints. To generate reports that provide information about constraints, use these commands:

- `report_constraint`

  Reports the constraint values in the current design. See Chapter 2, “Constraining Designs.”

- `report_port`

  Reports the current port settings. See Chapter 5, “Describing Logic Functions and Signal Interfaces.”

- `report_clock`

  Reports clock status. See Chapter 4, “Specifying Clocks and Clock Networks.”

- `report_attribute`

  Reports the attributes and their values associated with the cell, net, pin, port, instance, or design.

- `report_timing_requirements`

  Reports the point-to-point exceptions.

2

Constraining Designs

Constraints define the goals for Design Compiler to meet as it synthesizes your design. Constraints are measurable circuit characteristics such as timing, area, and capacitance.

You express your design goals as optimization constraints. These constraints are the most essential part of the description that drives the synthesis process.

To constrain your design effectively, you need to know how to do the tasks described in the following sections:

- Setting Design Constraints
- Determining Realistic Goals
- Checking the Design
- Defining Timing Paths

- Grouping Paths for Maximum Delay
- Setting a Path Group Critical Delay Range
- Fixing Hold Violations
- Defining Maximum Area
- Determining the Smallest Design
- Defining Minimum Porosity
- Defining Maximum Transition Time
- Defining Maximum Fanout
- Defining Expected Fanout for Output Ports
- Isolating Input and Output Ports
- Defining Maximum Capacitance
- Defining Minimum Capacitance
- Specifying Cell Degradation
- Specifying Ideal Nets
- Specifying Ideal Networks
- Setting Ideal Latency and Ideal Transition Time
- Automatically Disabling Design Rule Fixing on Special Nets
- Constraining Designs for Single-Cycle Timing
- Constraining Multifrequency Designs
- Overriding Single-Cycle Timing
- Setting False Paths

- Setting Multicycle Paths
- Defining Maximum Delay for Paths
- Defining Minimum Delay for Paths
- Resetting Paths to Single-Cycle Timing
- Using Case Analysis to Set Constant Paths
- Using Mode Analysis to Set Active Modes
- Applying Time Borrowing to Level-Sensitive Latches
- Limiting or Disabling Time Borrowing
- Constraining Designs Containing Asynchronous Logic
- Reporting Constraints
- Removing Constraints

Before you constrain your design, read Chapter 1, “Basic Constraint Concepts.”

Setting Design Constraints

Constraints reflect your performance objective for a design. They consist of timing constraints—such as minimum delay, maximum delay, and clock specifications—and maximum area. Design Compiler uses constraints to guide the optimization process.

Timing and area are usually the primary considerations. Other constraints, such as how easily a design can be routed (porosity), can also be considered.

When you set a constraint, the change is made the next time you optimize the design.

To get the most effective results from Design Compiler, set the constraints close to your design goals. Sometimes you can improve the results by setting the constraints to be slightly (1 percent to 10 percent) higher than needed, but this improvement is not guaranteed. If no timing goal is set (the default), Design Compiler applies only design rule checks to the design. If timing goals are set to be small (for example, 0), Design Compiler adds buffers to critical paths or duplicates logic on heavily loaded nets when trying to meet this goal, which can cause a significant increase in area.

With realistic timing goals, Design Compiler produces the smallest circuit that most closely satisfies the goal.

The default attribute values are unrealistic, so the results of optimization on circuits that have no attributes set are usually not optimal. For example, by default, input ports have a default drive value of zero (or infinite drive strength) and output ports have a default load of zero.

If you do not define the drive strength for an input port, Design Compiler assumes infinite drive strength. If the input port drives a heavily loaded net, Design Compiler does not add buffers to support the net drive, because the port is defined as having infinite drive strength.

Determining Realistic Goals

Usually the clock period, timing numbers, and area of the circuit are provided in the design specification. However, goals might at times not be available in a design specification or you need to optimize only a portion (subdesign) of a larger design. In the latter case, you might know goals for the entire design but not for the subdesign.

When realistic goals are unknown, map the design or subdesign to gates, setting no constraints, to determine the current design speed. Use these results to determine the constraints to set on the design, then recompile the design.

If the design is already in netlist format (mapped to gates), you can extract the constraints by using the `derive_timing_constraints` command.

Checking the Design

After you set constraints on a design and before you compile it, run the `check_design` command to identify and correct any problems in the design. The `check_design` command checks that the internal representation of the design is correct and issues warnings and errors. For more information about `check_design`, see the *Design Compiler User Guide*.

Defining Timing Paths

A timing path is a path through logic along which signals can propagate. Paths normally start at primary inputs or clock pins of registers and end at primary outputs or data pins of registers. A register is any sequential cell—flip-flop, latch, or other leaf cell—with setup and hold requirements.

Setup time is a time specified in the technology library for sequential cells. Setup is the requirement that data be stable for a given time before the active clock edge. This time can be scaled by operating conditions if the library contains scaling factors for setup. Setup time on a cell creates a maximum delay requirement for paths leading to the data pin of the cell.

Hold time is a time specified in the technology library for sequential cells. Hold is the requirement that the signal on the data pin must remain stable for a given time after the active clock edge. A hold time can be scaled by operating conditions if the library contains scaling factors for hold. Hold time on a cell creates a minimum delay requirement for paths leading to the data pin of that cell. During compilation, hold time violations are fixed if the clock object has the `fix_hold` attribute.

Slack is the amount of margin by which maximum or minimum path delay requirements are met. Positive slack indicates that the requirement is met; negative slack indicates a violation. Slack is displayed in timing reports.

A violation indicates a constraint is not met.

- A setup violation occurs when a timing path is longer than its targeted maximum delay. The cost function considers the worst violator within each path group when calculating maximum delay cost.
- A hold violation occurs when a timing path is shorter than its targeted minimum delay. A violation is the same as a negative slack value.

Timing Path Types

There are four types of timing paths (see Figure 2-7):

- Primary input to register (in1 to FF1)

  These paths are usually constrained by specification of the clock for the register and setting of an input delay relative to a clock on the input port.

- Register to register (FF1 to FF2)

  These paths are constrained by specification of the clock for the registers.

- Register to primary output (FF2 to out1)

  These paths are usually constrained by specification of the clock for the register and setting of an output delay relative to a clock on the output port.

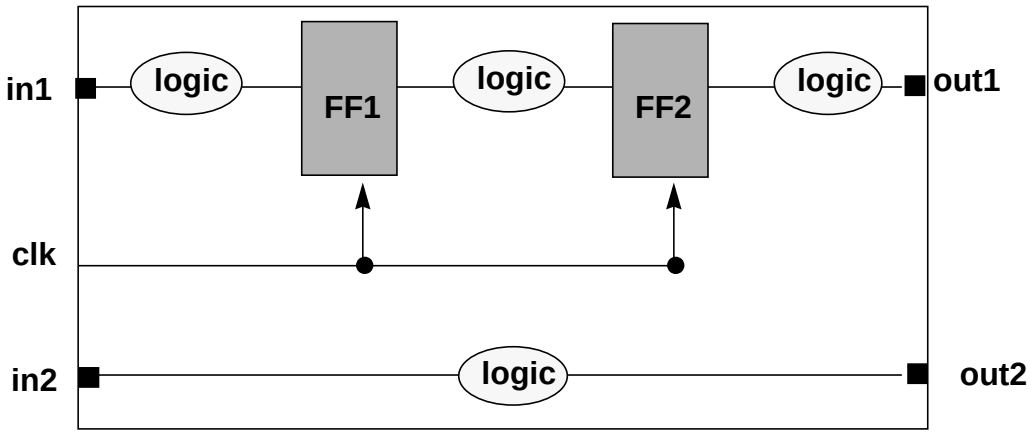
- Primary input to primary output (in2 to out2)

  You can constrain paths by setting an input delay relative to a clock on the input port and an output delay relative to a clock on the output port.

Paths to registers end at the data pins of registers. Paths from registers start at the clock pin of the register (not at the output pin).

Figure 2-7 shows the timing path types.

Figure 2-7 Timing Path Types



Path-Based Commands

Commands that operate on timing paths or endpoints are called path-based commands. They are

- `group_path`
- `set_max_delay`
- `set_min_delay`
- `set_false_path`
- `set_multicycle_path`
- `reset_path`

Path-based commands use `-from`, `-to`, and `-through` options to specify timing paths; they accept clocks in the *`from_list`* or *`to_list`*.

The general syntax of the path-based commands is

```
command_name      [options] -from  from_list
                  -through  through_list  -to  to_list
```

The following details apply to path-based commands:

- Path startpoints are usually input ports or clock pins of registers.
- Path endpoints are usually output ports or data pins of registers.
- When you specify a clock in the *`from_list`*, all path startpoints related to that clock are affected. This includes clock pins of registers in the transitive fanout of the clock sources and ports with input delay relative to the clock.
- When you specify a clock in the *`to_list`*, all path endpoints related to the clock are affected. This includes data pins of registers in the transitive fanout of the clock sources and ports with output delay relative to the clock.

You can use cell names in the *`from_list`* or *`to_list`* as a shorthand for the pin names when there is one relevant pin. For example, -to FF1 is the same as -to FF1/D. Do not use cell names for path endpoints when the cell is a JK flip-flop or a multiplexed flip-flop.

Using the `-through` option, you can control the timing on each specific path between a startpoint and an endpoint, even when different paths have different timing requirements.

Some path-based commands overwrite the information from other commands. For example, using `set_max_delay` on a path removes a previous multicycle path specification on the same path.

To remove maximum delay, minimum delay, multicycle path, and false path information for the specified paths, use the `reset_path` command.

To show the current path-based timing requirements on a design, use the `report_timing_requirements` command.

To report the timing path requirements in expanded format, use the `-expanded` option.

To report invalid path-based timing requirements, use the `report_timing_requirements -ignored` command and option.

To show the contents of all path groups in the design, use the `report_path_group` command.

- If you specify a *`to_list`* but no *`from_list`*, all paths leading to *`to_list`* are affected (interpreted as wildcards).
- If you specify a *`from_list`* but no *`to_list`*, all paths starting at *`from_list`* are affected.

Do not enter multiple commands with either *`from_list`* without *`to_list`* or *`to_list`* without *`from_list`* if paths common to the two specifications exist. Explicitly use both *`from_list`* and *`to_list`* for those common paths. The more-specific information takes precedence over the wildcard commands (if the command is issued after the wildcard commands). For additional information on precedence assessment, refer to “Precedence Assessment When Multiple Exceptions Satisfy a Path” on page 2-50.

To show the contents of a collection of path groups from the current design, use the `get_path_groups` command. This command creates a collection of path groups from the current design. You can

assign these path groups to a variable or pass them into another Tcl command. The `get_path_groups` command is available only in dctcl mode.

The syntax is

```
get_path_groups   patterns [-filter expression]
                  [-quiet] [-regexp] [-nocase]
```

Note:

For more information, see the `get_path_groups` man page.

Grouping Paths for Maximum Delay

A path group is a collection of paths considered together in cost function calculations. Each path group has a name and a weight and contributes to the maximum delay cost for the design. You can assign paths or endpoints to named groups. By default, Design Compiler creates a separate path group for each clock domain.

The cost function for maximum delay takes only the worst violator within each path group. This path allows you to control how the maximum delay cost is computed.

The default path group is an implicit path group. It contains the paths that are not in any other group. Its weight is 1.0.

The default weight for a path you create is 1.0, but you can specify different weights for various groups to favor certain paths or endpoints in the maximum delay cost calculation.

The path group affects the group's contribution to overall delay cost. For example,

```
design_max_delay_cost = sum of each group (cost * weight)
```

Design Compiler checks each timing path in the design for violation of maximum delay. The actual path delay for rising and falling signals is compared with a target path delay. The worst violation for a group is the path with the largest negative slack. Design Compiler works on the endpoint for the path with the most negative slack.

If you define the margin of delay (critical range) for a path group, Design Compiler works not only on the worst violation but also on the other endpoints that have negative slack and are within the defined range. Design Compiler optimizes only the worst path to each endpoint within the critical range.

To create a path group, use either the `group_path` command, described here, or the `create_clock` command described in Chapter 4, "Specifying Clocks and Clock Networks."

Additionally, use the `group_path` command to add paths to existing groups, to remove paths from groups, or to change the path group weight.

The syntax is

```
group_path    [-name  group_name  | -default]
              [-weight  weight_value ]
              [-critical_range  range_value ]
              [-from  from_list ] [-through  through_list ]
              [-to  to_list ]
```

Note:

For more information, see the `group_path` man page.

To return paths to the default group, use `group_path -default`.

Setting a Path Group Critical Delay Range

The `set_critical_range` command sets the critical range value for any path groups to which you do not assign a value with the `group_path` command. Critical range is a timing delay margin Design Compiler uses during optimization.

The syntax is

```
set_critical_range  range_value design_list
```

Note:

For more information, see the `set_critical_range` man page.

Examples

To set the critical range for the design top to 10, enter

```
dc_shell> set_critical_range 10 top
```

To undo `set_critical_range`, use the `remove_attribute` or `reset_design` command.

To show the critical range values for each path group, use the `report_path_group` command.

To show the critical range cost of the design, use the `report_constraint` command.

Fixing Hold Violations

The `set_fix_hold` command directs Design Compiler to fix hold violations during compilation. It places a `fix_hold` attribute on a clock object. Hold time violations are fixed during compilation only if the `fix_hold` attribute is applied to related clocks.

When set, the `fix_hold` attribute directs the `compile` program to insert a delay to fix hold violations for timing paths that end at registers fed by this clock.

Note:

Meeting hold time requirements is less difficult than meeting setup time requirements. Minimum delay violations are exaggerated under best-case operating conditions; use the worst-case operating condition until setup is met.

The syntax is

```
set_fix_hold  clock_list
```

`clock_list`

The clocks on which to set the `fix_hold` attribute.

Example

```
dc_shell> set_fix_hold CLOCK
```

To undo a `set_fix_hold` command, use the `remove_attribute` command. For example, enter

```
dc_shell> remove_attribute CLOCK fix_hold
```

Defining Maximum Area

The maximum area optimization constraint and its cost calculation are described in Chapter 1, “Basic Constraint Concepts.”

The `set_max_area` command specifies an area target and places a `max_area` attribute on the current design.

The syntax is

```
set_max_area [-ignore_tns]  area
```

Note:

For more information, see the `set_max_area` man page or the *Design Compiler User Guide*.

Examples

```
dc_shell> set_max_area 0.0
dc_shell> set_max_area 14.0
```

Determining the Smallest Design

Determining the smallest design can be helpful.

The following script guides Design Compiler to optimize for area only. It constrains a design only for minimum area (when you do not care about timing). For the timing to make sense, you must apply clocking and input and output delay.

```
/* example script for smallest design */
remove_constraint -all
remove_clock -all
set_max_area 0
```

Defining Minimum Porosity

The minimum porosity optimization constraint and its cost calculation are described in Chapter 1, “Basic Constraint Concepts.”

Porosity optimization is effective only for two-layer metal technology.

The `set_min_porosity` command sets a minimum porosity target value. It places a `min_porosity` attribute on the current design or named designs.

The syntax is

```
set_min_porosity  porosity  [ design_list ]
```

Note:

For more information, see the `set_min_porosity` man page.

Example

A porosity of 30 allows 30 percent of the total cell area to be used for over-the-cell routing. To set this value, enter

```
dc_shell> set_min_porosity 30
```

To undo a `set_min_porosity` command, use either `set_min_porosity 0` or `remove_attribute`.

Defining Maximum Transition Time

The maximum transition time design rule constraint and its cost calculation are described in Chapter 1, “Basic Constraint Concepts.”

The set_max_transition command sets a maximum transition time for the nets attached to the listed ports or to all the nets. The set_max_transition command places a max_transition attribute on the specified objects.

Note:

The max_transition attribute does not provide a direct way to limit the actual capacitance of nets. To limit capacitance directly, use set_max_capacitance.

The syntax is

```
set_max_transition  time object_list
```

Note:

For more information, see the set_max_transition man page.

Example

To set a maximum transition time of 3.2 for the design adder, enter

```
dc_shell> set_max_transition 3.2 find(design,adder)
```

To undo a set_max_transition command, use remove_attribute. Enter

```
dc_shell> remove_attribute find (design,adder) max_transition
```

Defining Maximum Fanout

The set_max_fanout command sets the maximum allowable fanout load for the listed input ports. The set_max_fanout command sets a max_fanout attribute on the listed objects.

The maximum fanout design rule constraint and its cost calculation are described in Chapter 1, “Basic Constraint Concepts.”

The syntax is

```
set_max_fanout  fanout object_list
```

Note:

For more information, see the `set_max_fanout` man page.

To undo a maximum fanout value set on an input port or design, use the `remove_attribute` command. For example,

```
dc_shell> remove_attribute find (port, port_name) max_fanout
dc_shell> remove_attribute find (design,design_name) max_fanout
```

Defining Expected Fanout for Output Ports

The `set_fanout_load` command sets the expected fanout load value for listed output ports.

Design Compiler adds the fanout value to all other loads on the pin driving each port in *`port_list`* and tries to make the total load less than the maximum fanout load of the pin.

The syntax is

```
set_fanout_load fanout port_list
```

Note:

For more information, see the `set_fanout_load` man page.

To undo a `set_fanout_load` command set on an output port, use the `remove_attribute` command. For example,

```
dc_shell> remove_attribute port fanout_load
```

To determine the fanout load, use the `get_attribute` command.

Examples

To find the fanout load on the input pin of library cell AND2 in library libA, enter

```
dc_shell> get_attribute "libA/AND2/i" fanout_load
```

To find the default fanout load set on technology library libA, enter

```
dc_shell> get_attribute libA default_fanout_load
```

Isolating Input and Output Ports

The `set_isolate_ports` command inserts isolation logic at specified input or output ports. You isolate input and output ports to improve the accuracy of timing models.

Input ports are isolated in the following cases:

- When they drive one or more input pins of a cell having several input pins (as a result of boundary optimization)
- When they drive one or more input pins belonging to different cells having several input pins

Output ports are isolated in the following cases:

- To ensure that the cell driving an output port does not also drive some internal logic within the design
- To specify particular driver cells at the output ports.

This is useful when you want each output port to be driven explicitly by its own driver (no sharing of output drivers by two or more ports) or when you want to compile the design in the context of the environment in which the design will be used

The syntax for this command is

```
set_isolate_ports <object_list> [-type buffer|inverter]
         [-driver <cellname>] [-force]
```

where

- *object_list* is the list of input or output ports you want to isolate
- `-type` specifies a buffer (`buffer`) or an inverter (`inverter`) from the target library as the isolation cell
- `-driver` specifies a particular cell from the target library as the isolation cell
- `-force`

  With input ports, `-force` indicates that isolation logic is inserted after the specified input port.

  With output ports, `-force` indicates that isolation logic is inserted even if no internal feedback from the output drivers occurs.

The isolation logic can be a buffer or a pair of inverters. Either Design Compiler selects the buffer or inverter from the target library, or you specify a particular cell from the target library. Note, however, that a user-specified cell must be a buffer or an inverter. Otherwise, Design Compiler outputs an error message.

The inserted isolation logic has the `size_only` attribute assigned to it. When you compile the design, therefore, only sizing optimization is allowed for this logic.

Note:

If the isolation logic is composed of an inverter pair, the `size_only` attribute is assigned only to the second inverter, which allows flexibility in optimizing the first inverter.

The syntax for this command is

```
set_isolate_ports <object_list> [-type buffer|inverter]
         [-driver <cellname>] [-force]
```

For example,

```
set_isolate_ports all_outputs() -driver IVDAP
```

inserts the isolation logic cell IVDAP on all output ports of the current design.

You issue the `set_isolate_ports` command before the `compile` command. That is, the isolation logic is inserted before the design is compiled. A sample script might be

```
target_library = lsi_10k.db
link_library = {“*” lsi_10k.db}
read -f verilog ./t1.v
current_design test1
link
set_isolate_ports all_outputs() -driver IVDAP
compile -map_effort medium
```

It is important to understand that port isolation can be applied only to the input or output ports of the *current* design. Therefore, to apply port isolation to a subdesign of your top-level design, you must first make the subdesign the current design. This condition is true as well when you use the simple compile mode.

Port isolation is currently intended for use only during bottom-up compilation. That is, isolating hierarchical instance pins of lower-level designs from the top-level design in a top-down compilation is not supported.

Also, in a bottom-up compilation, you cannot simply isolate the ports of a subblock that you have temporarily designated as the current design and then expect that isolation logic automatically to propagate upward when you compile the top-level design. To ensure that the isolation logic of a subblock remains while the top-level design is compiled, you must use the `propagate_constraints` command to propagate the constraints upward after the subblock ports are isolated and before you compile the top-level design.

The following script fragment shows you how to compile a subdesign with port isolation, followed by a top-level compile:

```
current_design test1
link
set_isolate_ports all_outputs() -type buffer -force
compile -map_effort medium
current_design top
propagate_constraints
compile -map_effort medium
```

Port isolation does not work if

- A `dont_touch` net is connected to the port
- The specified isolation cell is not in the target library

- The specified port is not an output or input port (inout ports and tristate ports are not supported)
- The specified type option (in the `set_isolate_ports` command) is not a buffer or an inverter

You can remove the port isolation attribute from designs by using the `remove_isolate_ports` command. The isolation cells are then removed during the next compile. You can also use the `remove_attribute` command.

To obtain a list of isolated input or output ports in a design, use the `report_isolate_ports` command. The `report_constraint` and `report_compile_options` command also provide information about the isolated input or output ports.

When you issue the `report_isolate_ports` command, you see a report similar to the following:

```
****************************************
Report : isolate_ports
Design : top
Version: 2001.08
Date   : Wed Apr 25 17:05:43 2001
****************************************


Port Name     Cell Name       Inst. Name      Type       Forced Insertion
--------------------------------------------------------------------------
stp           IVDA            U35             buffer     yes
rhcp          IVP             U34             inverter   yes
hip_1         IVDA            U32             buffer     no
--------------------------------------------------------------------------
```

Other commands that support the port isolation feature are

- `reset_design`

  This command removes the port isolation attribute, as well as other attributes from the design.

- `write_script`

  This command writes any `set_isolate_ports` commands (along with the other dc_shell commands) into the script file.

For more information on the commands discussed in this section, see the appropriate man pages.

Defining Maximum Capacitance

The maximum capacitance design rule constraint and its cost calculation are described in Chapter 1, “Basic Constraint Concepts.”

The `set_max_capacitance` command sets a maximum capacitance for the nets attached to named ports or to all the nets in a design. This command allows you to control capacitance directly and places a `max_capacitance` attribute on the listed objects.

To set a maximum capacitance attribute on specified objects, use the `set_max_capacitance` command.

The syntax is

```
set_max_capacitance  capacitance object_list
```

Note:

For more information, see the `set_max_capacitance` man page.

Examples

To set a maximum capacitance of 3 for the design adder, enter

```
dc_shell> set_max_capacitance 3 find(design,adder)
```

To undo a `set_max_capacitance` command, use `remove_attribute`. For example, enter

```
dc_shell> remove_attribute find(design,adder) max_capacitance
```

Defining Minimum Capacitance

The minimum capacitance design rule constraint and its cost calculation are described in Chapter 1, “Basic Constraint Concepts.”

The `set_min_capacitance` command sets a defined minimum capacitance value on listed input or bidirectional ports.

To set a minimum capacitance for nets attached to input or bidirectional ports, use the `set_min_capacitance` command.

The syntax is

```
set_min_capacitance  capacitance object_list
```

Note:

For more information, see the `set_min_capacitance` man page.

Example

To set a minimum capacitance value of 12.0 units on the port named high_drive, enter

```
dc_shell> set_min_capacitance 12.0 high_drive
```

To report only minimum capacitance constraint information, use the `-min_capacitance` option of the `report_constraint` command.

To get information about the current port settings, use the `report_port` command.

To undo a `set_min_capacitance` command, use the `remove_attribute` command.

Specifying Cell Degradation

The `set_cell_degradation` command sets the `cell_degradation` attribute to a specified value on specified input ports.

During compilation, if `cell_degradation` tables are specified in a technology library, Design Compiler tries to ensure that the capacitance value for a net is less than the specified value. The `cell_degradation` tables give the maximum capacitance that a cell can drive, as a function of the transition times at the inputs of the cell.

If cell_degradation tables are not specified in a technology library, you can set `cell_degradation` explicitly on the input ports. Design Compiler tries to fix `cell_degradation`, provided the `compile_fix_cell_degradation` is set to true.

By default, a port has no `cell_degradation` constraint.

Note:

Use of the `set_cell_degradation` command requires a DC Ultra license.

The syntax is

```
set_cell_degradation  cell_deg_value object_list
```

Note:

For more information, see the `set_cell_degradation` man page.

Example

This command sets a maximum capacitance value of 2.0 units on the port named late_riser:

```
dc_shell> set_cell_degradation 2.0 late_riser
```

To get information about optimization and design rule constraints, use the `report_constraint` command.

To remove the `cell_degradation` attribute, use the `remove_attribute` command.

Specifying Ideal Nets

You use the `set_ideal_net` command to specify nets that are visible from the current design as ideal nets. Defining certain high fanout nets that you intend to synthesize separately, such as scan-enable and reset nets, as ideal nets can reduce runtime by avoiding unnecessary retiming and unwanted design changes during optimization.

Ideal nets have the following properties:

- The nets are exempt from timing updates, delay optimization, and design rule fixing (DRC fixing)—that is, `max_capacitance`, `max_fanout`, and `max_transition` design rules are ignored.

The tool disables delay optimization by assigning the `dont_touch` attribute to an ideal net and the `size_only` attribute to any nonsequential cells connected to an ideal net. (Sequential cells connected to an ideal net are not assigned this attribute.) The tool disables DRC fixing by setting the DRC cost to 0 on these nets.

- The nets are assigned ideal timing conditions—that is, by default, ideal nets are assigned latency, transition time, and capacitance values of 0.

  You can, however, change the ideal latency and ideal transition values by using the `set_ideal_latency` and `set_ideal_transition` commands, respectively. For further information, see "Setting Ideal Latency and Ideal Transition Time" on page 2-33.

- The nets are assigned the `ideal_net` attribute.

Note:

Although clock nets are ideal by default, you should not use the `set_ideal_net` command to modify the properties of these nets. Use instead `create_clock` and related clock commands.

When the `report_timing` command is issued or when *any* timing value, such as latency, transition, or delay, is changed, the ideal nets are represented by their ideal latency, ideal transition, and ideal capacitance values in the timing update calculation. No specific timing update or optimization is performed on the ideal nets themselves.

The `ideal_net` attribute is not propagated to the other nets of a network. Furthermore, the attribute is not propagated through hierarchical pins. To propagate the `ideal_net` attribute through networks, use the `set_ideal_network` command, discussed in “Specifying Ideal Networks” on page 2-29.

There is no specific command that reports ideal nets. Use the `report_net` command. The ideal nets (and the pins they connect) are marked with an “i” in the net report. Also, in the timing report (`report_timing` command), a “^” marks the ideal latency and ideal transition values.

To remove the `ideal_net` attribute from specified nets of the current design, use the `remove_ideal_net` command.

For more information about the `set_ideal_net` and `remove_ideal_net` commands, see the appropriate man pages.

Specifying Ideal Networks

You use the `set_ideal_network` command to create ideal networks.The ideal network feature is an extension of the ideal net property. For a discussion of the ideal net property, see “Specifying Ideal Nets” on page 2-27.

Note:

Although clock nets and networks are ideal by default, you should not use the `set_ideal_network` command to modify the properties of these networks. Use instead `create_clock` and related clock commands.

The `set_ideal_network` command assigns the `ideal_net` attribute to the user-specified input ports and pins of a design. Any input port or internal pin, including a hierarchical pin, of the current design can be a source object. These ports and pins serve as the source objects from which the tool automatically spreads the `ideal_net` attribute to the nets, cells, and pins of the connected transitive fanouts toward the endpoints of the timing paths, according to certain propagation rules.

The `ideal_net` attribute propagation is governed by the following rules:

- A pin is treated as ideal if it is
  - A pin specified in the object list of the `set_ideal_network` command, or
  - A driver pin and its cell is ideal, or
  - A load pin attached to an ideal net
- A net is treated as ideal if all its driving cells are ideal.
- A combinational cell is treated as ideal if
  - All its input pins are ideal, or
  - At least one input pin is ideal and all nonideal input pins are attached to constant nets

  Note:

  The cell is not treated as ideal if any of the nonideal input pins is attached to case-analysis constant net.

A hierarchical pin can propagate the `ideal_net` attribute.

Propagation stops at the pins where these conditions are not met, These pins are referred to as network boundary pins. They are ideal pins.

Note:

Ideal network propagation can traverse combinational cells, but it stops at sequential cells, even if the sequential cells are connected to ideal clock pins. Also, if you used the `set_ideal_net` command to set the `ideal_net` attribute on one or more nets connected to the inputs of a combinational cell, the `ideal_net` attribute does not propagate through the cell.

As with ideal nets, the nets, cells, and pins of ideal networks are treated as ideal objects, that is, they have the following properties:

- Ideal objects are exempt from timing updates, delay optimization, and DRC fixing.

  The tool disables delay optimization of an ideal network by assigning the `dont__touch` attribute to the ideal cells and ideal nets and the `size_only` attribute to the nonsequential boundary cells of the network. (Sequential boundary cells are not assigned either attribute.) If there are unmapped nonsequential cells in the network, such as generic technology cells (GTECH cells) or synthetic cells, these cells are mapped before they are assigned the `size_only` attribute. In addition, the tool disables DRC fixing by setting the DRC cost to 0 for the network.

- Ideal objects are assigned ideal timing properties (ideal latency, ideal transition, and ideal capacitance).

  By default, the ideal networks are assigned ideal latency, ideal transition time, and ideal capacitance values of 0. You can change the latency and transition values by using the

`set_ideal_latency` and `set_ideal_transition` commands, respectively. For further information, see “Setting Ideal Latency and Ideal Transition Time” on page 2-33.

If an ideal network overlaps a clock network, the clock timing (clock latency and transition values) overrides the ideal timing for the clock portion of the overlapped networks.

To remove the `ideal_net` attribute from the ideal networks of the current design and return the cells and nets of the connected networks to their initial, nonideal state, use the `remove_ideal_network` command.

Use the `report_ideal_network` command to display information about ports, pins, nets, and cells of the ideal networks in the current design. The arguments for the command include `-net`, `-cell`, `-load_pin, -timing,` and *`object_list`*. If you do not specify any arguments, all ideal network sources (input ports and pins) are displayed but no additional information about the ideal networks is displayed. If you specify a source port or pin with the command, all the ideal network information (ideal nets, ideal cells, load pins, and internal pins with ideal timing that belong to the current design) are displayed. You can limit the report by selecting command arguments.

Commands such as `report_attribute`, `report_timing`, `report_cell`, and `report_net` indicate the `ideal_net` attribute propagated to nonsource objects of an ideal network, as well as the source ports and pins of the network. The `get_attribute` command, however, indicates the `ideal_net` attribute only for the source ports and pins of the network. The propagated attribute is not indicated.

For more information about the `set_ideal_network`, `remove_ideal_network`, and `report_ideal_network` commands, see the appropriate man pages.

Setting Ideal Latency and Ideal Transition Time

The default latency and transition values for ideal networks and ideal nets is 0. You can override these defaults by using the following commands:

- `set_ideal_latency`
- `set_ideal_transition`

Note:

The timing of ideal networks and ideal nets is updated whenever you issue the `set_ideal_latency` or `set_ideal_transition` command. (Timing is also updated for ideal networks if you change the source pins.)

You can use these commands to set the ideal latency and ideal transition on the source pin (top-level input ports and leaf cell pins) of an ideal net or network and on any nonsource pin of an ideal network. The specified values override any library cell values or net delay values. For ideal networks, the ideal latency and transition values are propagated from the source pins to the network boundary pins.

The total ideal latency at any given point of an ideal network is the sum of the source pin ideal latency and all the ideal latencies of the leaf cell pins along the path to the given point. The ideal transition values specified at the various source and leaf cell pins are

independent and noncumulative. The transition for an unspecified input pin is the ideal transition of the closest pin with a specified ideal transition value. This rule applies to boundary pins as well.

The `set_input_delay` command is applicable to ideal networks. This delay is treated as the off-block latency or source latency.

Note:

The `characterize` and `write_script` commands support ideal timing. The `characterize` command captures the ideal timing attributes and automatically produces a set of ideal timing commands for the characterized block. The `write_script` command automatically writes out the ideal timing commands for the current design.

You can remove ideal latency and ideal transition values by using the following commands:

- `remove_ideal_latency`
- `remove_ideal_transition`

These commands remove the ideal latency and ideal transition values from the specified source pins (top-level input ports and pins). In the case of ideal networks, the command also removes the ideal latency and transition values from any nonsource pins on which they were set and from any pins to which the ideal attributes were propagated. The default value of 0 is restored to the ideal pins.

For more information about the `remove_ideal_latency` and `remove_ideal_transition` commands, see the appropriate man pages.

Automatically Disabling Design Rule Fixing on Special Nets

You use the `set_auto_disable_drc_nets` command to enable or disable DRC fixing on clock, constant, or scan nets. The command acts on all the nets of a given type in the current design; that is, depending on the options you specify, it acts on all the clock nets, or all the constant nets, or all the scan nets, or on any combination of these three net types. Nets that are disabled with respect to DRC fixing are marked with the `auto_disable_drc_nets` attribute.

Note:

This command has the same functionality as the `set_auto_ideal_nets` command and replaces that command.

By default, the clock and constant nets of a design have DRC fixing disabled (this is the same as using the `-default` option of the command); the scan nets do not. You can use the `-all` option to disable DRC fixing on all three net types. Alternatively, you can independently disable or enable DRC fixing for the clock nets, constant nets, or scan nets by assigning a true or false value to the `-clock`, `-constant`, or `-scan` options, respectively. Finally, you can use the `-none` option to *enable* DRC fixing on all three types of nets.

Note:

Clock nets are ideal nets by default. Using the `set_auto_disable_drc_nets` command to *enable* DRC fixing does not affect the ideal timing properties of clock nets. You must use the `set_propagated_clock` command to affect the ideal timing of clock nets.

You cannot use the `set_auto_disable_drc_nets` to override disabled DRC fixing on ideal networks marked with the `ideal_net` attribute. This command never overrides the settings specified by the `set_ideal_net` or the `set_ideal_network` command.

For more information about this command, see the appropriate man page.

Constraining Designs for Single-Cycle Timing

Single-cycle timing is the default timing requirement for a path. Design Compiler automatically infers single-cycle timing from clock waveforms and from input delay and output delay information. Single-cycle timing means that data from an active edge of the startpoint clock is expected to be available at the path endpoint before the active edge of the endpoint clock that follows the startpoint edge. You can restore paths marked as nonsingle-cycle to single-cycle timing, by using the `reset_path` command.

Design Compiler automatically determines the maximum and minimum path delay requirements for the design. It examines the clock waveforms at the path startpoint and endpoint. The default for setup is to allow a single clock cycle for data to reach the path endpoint. The path length calculation considers library setup time and clock delay and uncertainty values. The hold data is launched one cycle later at the path startpoint.

To constrain a design for single-cycle timing, specify the following:

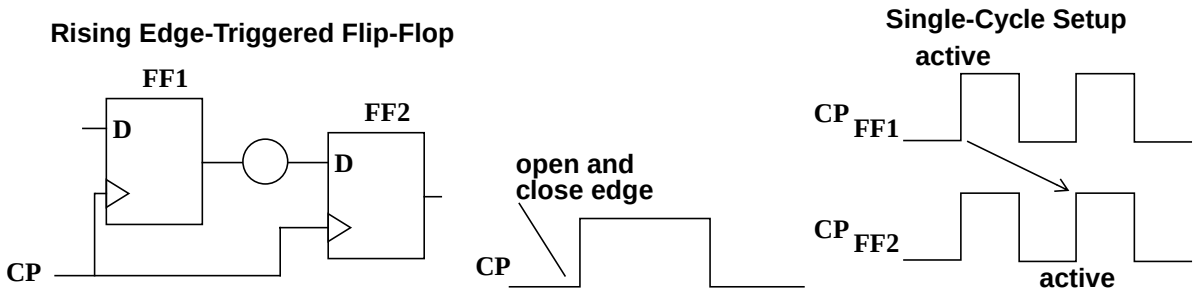
- Clocks (with period and waveform information)
- Input delays to input ports (with the relative clock)

- Output delays from output ports (with the relative clock)

Design Compiler calculates the default setup and hold relations and derives single-cycle timing, based on active edges. The data must be available before the active edge of the endpoint clock (following an active edge at the startpoint) to satisfy the single-cycle setup relation.

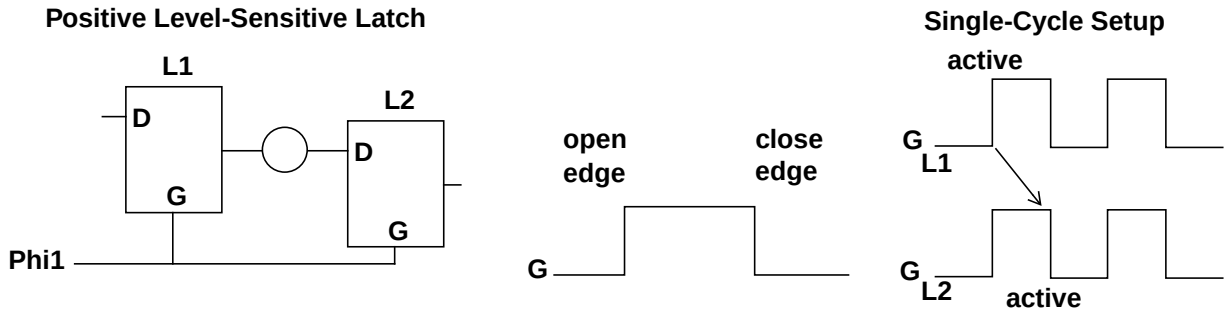
- At the startpoint, the active edge is the register's opening edge.
- At the endpoint, the active edge is the register's closing edge.
- For a rising-edge-triggered flip-flop, the rising edge is both the open and the close edges, as shown in Figure 2-8.

Figure 2-8 Single-Cycle Timing Active Edges and Setup for Rising-Edge-Triggered Flip-Flop



- For a positive level-sensitive latch, the rising edge is the open edge and the falling edge is the close edge, as shown in Figure 2-9.

Figure 2-9 Single-Cycle Timing Active Edges and Setup for Positive Level-Sensitive Latch

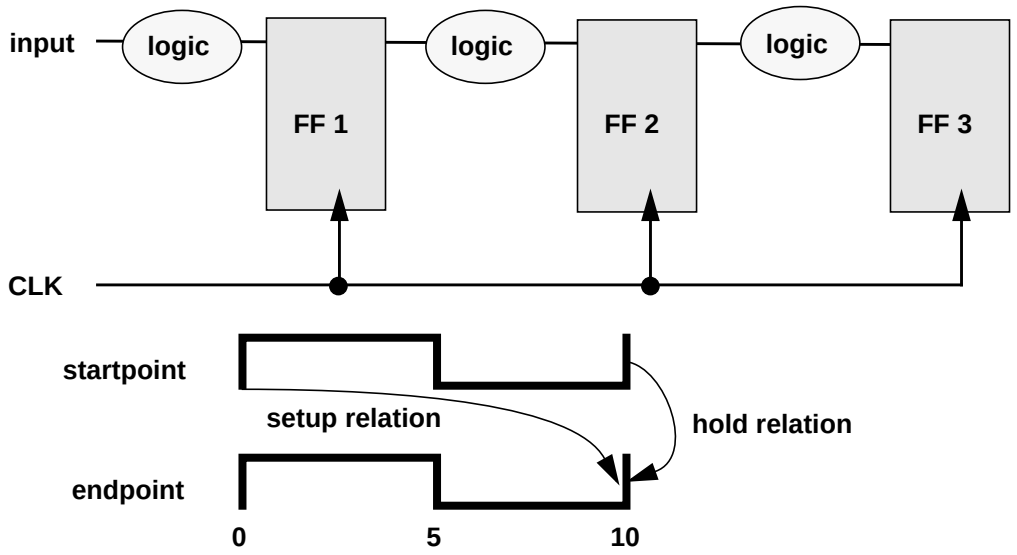


The following examples illustrate edge-triggered devices. The arrows show the default single-cycle relations for setup and hold.

Single-Phase Design Example

In Figure 2-10, the path from FF1 to FF2 is one cycle (10 units) for setup. For hold, the path must be longer than the library hold time of FF2.

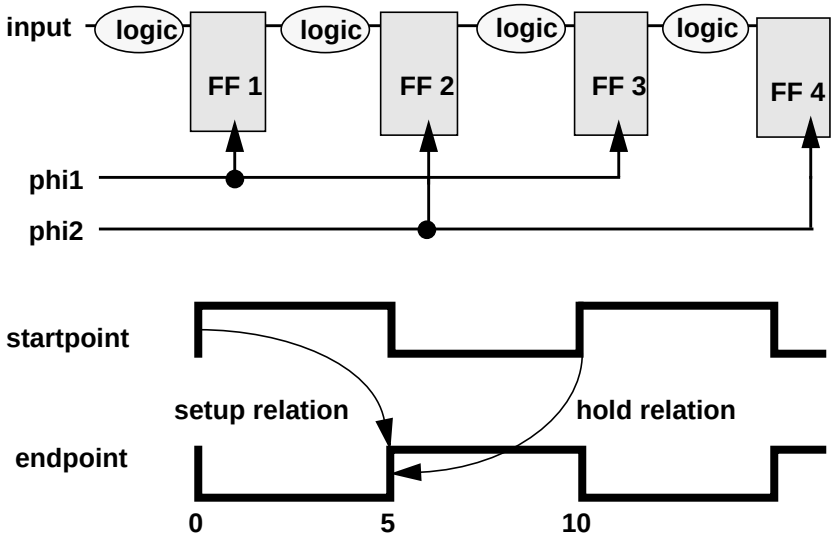
Figure 2-10 Single-Phase Design



Two-Phase Design Example

In Figure 2-11, the path length must be one-half cycle (5 units) or less because the active clock edge at the path endpoint occurs 5 units after the active edge of the path startpoint.

Figure 2-11 Two-Phase Design



Constraining Multifrequency Designs

Some designs have clocks of different cycle times with data paths between registers of the various clocks. Design Compiler determines all the edge dependencies in such cases. The length for a path must meet the most restrictive setup and hold relation for that path.

Setup and Hold Timing Checks in Multifrequency Designs

Design Compiler applies design rules based on active edges to determine single-cycle timing relationships for paths between clocked elements.

- For flip-flops, a single active edge both launches and captures data.

- For latches, the open edge launches data and the close edge latches data.

Setup Check

The setup check ensures that the correct data signal is available on destination registers in time to be properly latched.

The rule for setup is as follows:

- For multifrequency designs, multiple setup relations might exist between two clocks. For every latch edge of the destination clock, find the nearest launch edge that precedes each capture edge. The smallest difference between the setup latch edge and the setup launch edge determines the maximum delay requirement for this path.
- You can override the default relationship (single-cycle setup), by using the `set_multicycle_path` or `set_max_delay` commands. You can apply these commands to clocks, pins, ports, or cells. For example, setting the setup path multiplier to 2, using the `set_multicycle_path` command, delays the latch edge one clock pulse.

Changing the setup multiplier affects the default hold check.

Hold Check

The hold check verifies two things:

- Data from the source clock edge that follows the setup launch edge must not be latched by the setup latch edge.
- Data from the setup launch edge must not be latched by the destination clock edge that precedes the setup latch edge.

The hold check is determined in relation to each valid setup relationship, after application of multicycle path multipliers. The largest (most restrictive) difference between the hold latch edge and the hold launch edge is the minimum delay requirement for the path.

The hold relation is conservative. In some cases, you might need to override it. For multifrequency designs, using the `set_min_delay` command to override the default is more straightforward than using the `set_multicycle_path -hold` command.

To move the data launch time backward, use the `set_multicycle_path -setup -start` command.

To move the hold relation relative to the end clock, use the `set_multicycle_path -hold -end` command.

Setup and Hold Timing Check Examples

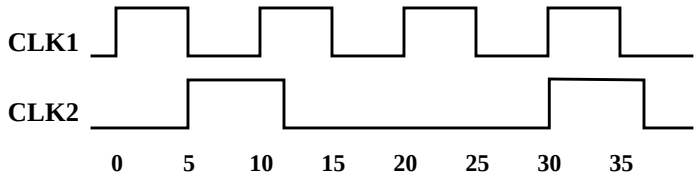
In this section, examples assume rising-edge-triggered devices.

Example 1

In this 10-ns-clock-to-25-ns-clock example, for a path from a CLK1 register to a CLK2 register,

- The most restrictive setup relation is 5 ns (from CLK1 edge at 0 to CLK2 edge at 5)
- The most restrictive hold time is 0 ns (from CLK1 edge at 30 to CLK2 edge at 30)

```
create_clock -period 10 -waveform {0 5} CLK1
create_clock -period 25 -waveform {5 12.5} CLK2
```

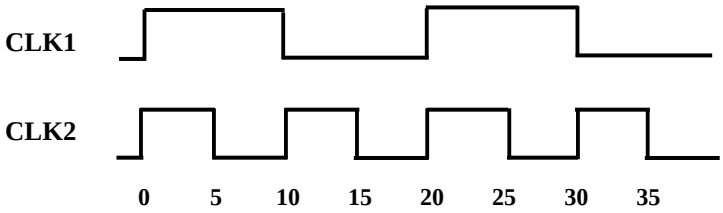


Example 2

In this 20-ns-clock-to-10-ns-clock example, for a path from a CLK1 register to a CLK2 register,

- The most restrictive setup relation is 10 ns (from CLK1 edge at 0 to CLK2 edge at 10).
- The most restrictive hold time is 0 ns (from CLK1 edge at 0 to CLK2 edge at 0).

```
create_clock -period 20 -waveform {0 10} CLK1
create_clock -period 10 -waveform {0 5} CLK2
```



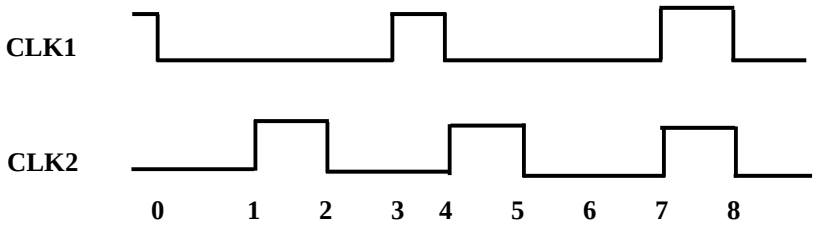
Example 3

In this 4-ns-clock-to-3-ns-clock example, for a path from a CLK1 register to a CLK2 register,

- The most restrictive setup relation is 1 ns (from CLK1 edge at 3 to CLK2 edge at 4).

- The most restrictive hold time is 0 ns (from CLK1 edge at 7 to CLK2 edge at 7).

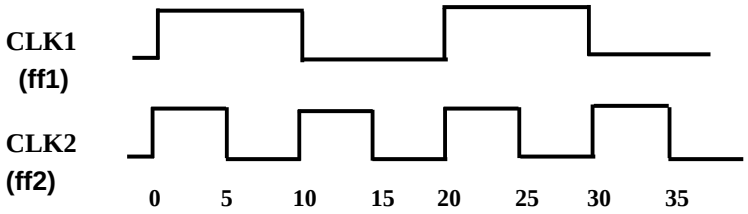
```
create_clock -period 4 -waveform {3 4} CLK1
create_clock -period 3 -waveform {1 2} CLK2
```



Example 4

In this 20-ns-clock-to-10-ns-clock example with a multicycle path of 2, the single-cycle setup relation is from the CLK1 edge at 0 ns to the CLK2 edge at 10 ns. With the multicycle path, the setup relation is now 20 ns (from CLK1 edge at 0 ns to CLK2 edge at 20 ns). The hold relations are determined according to that setup relation.

```
create_clock -period 20 -waveform {0 10} CLK1
create_clock -period 10 -waveform {0 5} CLK2
set_multicycle_path 2 -setup -from ff1/CP -to ff2/D
```



- Data from the source clock edge that follows the setup launch edge must not be latched by the setup latch edge. This implies a hold relation of 0 ns (from CLK1 edge at 20 ns to CLK2 edge at 20 ns).
- Data from the setup launch edge must not be latched by the destination clock edge that precedes the setup latch edge. This implies a hold relation of 10 ns (from CLK1 edge at 0 ns to CLK2 edge at 10 ns).

Because the most restrictive (largest) hold relation is used, the minimum delay requirement for this path is 10 ns. This is a conservative check that might not apply to certain designs. Often you know that the destination register will be disabled for the clock edge at 10 ns. To get a 0-ns requirement, you can modify this default hold relation by using either this command:

```
dc_shell> set_multicycle_path 1 -hold -end -from ff1/CP -to ff2/D
```

or this command:

```
dc_shell> set_min_delay 0 -from ff1/CP -to ff2/D
```

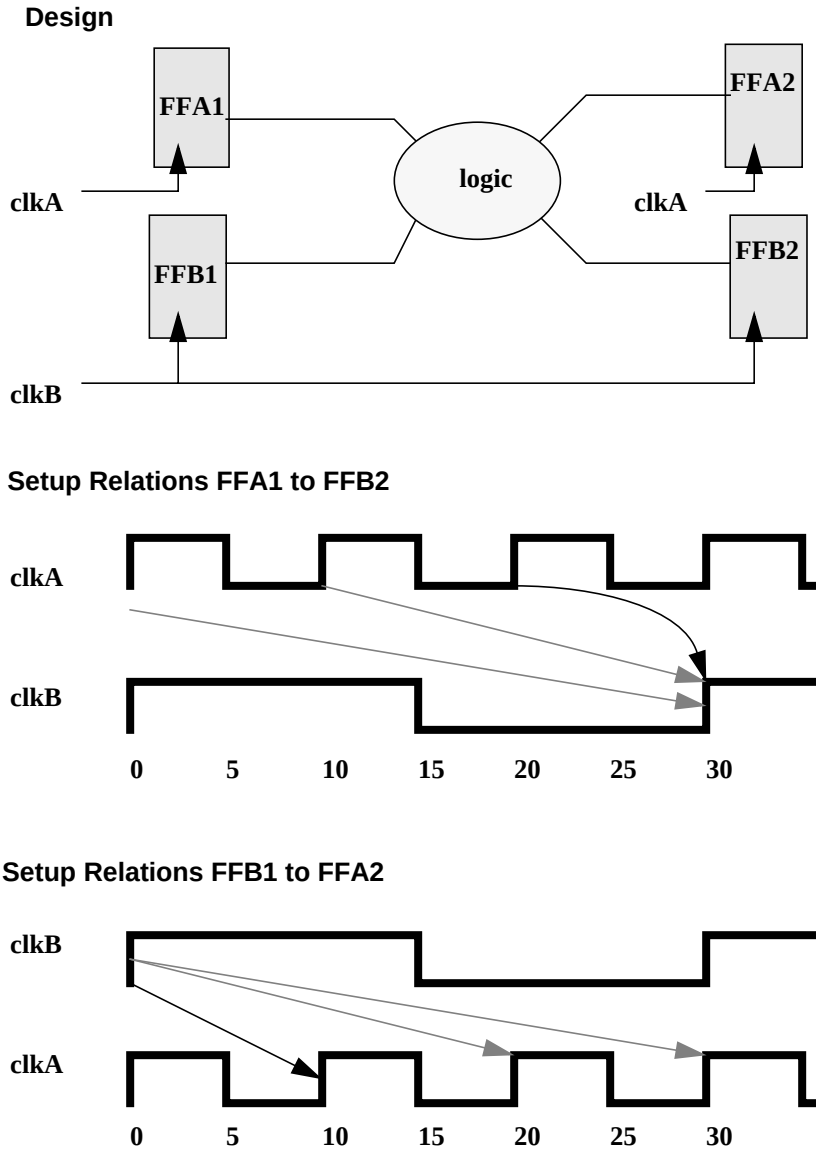
Example 5

In Figure 2-12,

- The most restrictive setup relation for the path from FFA1 to FFB2 is from the edge at 20 to the edge at 30 (10 units).
- The most restrictive setup relation for the path from FFB1 to FFA2 is from the edge at 0 to the edge at 10 (10 units).

The less-restrictive setup relations (shown as gray arrows) are ignored, because they are automatically satisfied if the most restrictive relations are satisfied.

Figure 2-12 Multifrequency Design and Setup Relations



The timing report for the design in Figure 2-12, FFA1 to FFB2 and FFB1 to FFA2, follows.

```
Startpoint: FFA1 (rising edge-triggered flip-flop clocked by clkA)
 Endpoint: FFB2 (rising edge-triggered flip-flop clocked by clkB)
 Constraint Group: clkB
 Path Type: max
```

```
Point                                       Incr       Path
-------------------------------------------------------------
clock clkA (rise edge)                     20.00      20.00
clock network delay (ideal)                 0.00      20.00
FFA1/CP (FF)                                0.00      20.00 r
FFA1/Q (FF)                                 1.00      21.00 r
...
FFB2/D (FF)                                 1.00      22.00 r
data arrival time                                     22.00
clock clkB (rise edge)                     30.00      30.00
clock network delay (ideal)                 0.00      30.00
clock uncertainty                           0.00      30.00
FFB2/CP (FF)                                0.00      30.00 r
library setup time                          0.00      30.00
data required time                                    30.00
-------------------------------------------------------------
data required time                                    30.00
data arrival time                                    -22.00
-------------------------------------------------------------
slack (MET)                                           8.00
Startpoint: FFB1 (rising edge-triggered flip-flop clocked by clkB)
Endpoint: FFA2 (rising edge-triggered flip-flop clocked by clkA)
Constraint Group: clkA
Path Type: max
Point                                       Incr       Path
-------------------------------------------------------------
clock clkB (rise edge)                      0.00       0.00
clock network delay (ideal)                 0.00       0.00
FFB1/CP (FF)                                0.00       0.00 r
FFB1/Q (FF)                                 1.00       1.00 r
...
FFA2/D (FF)                                 1.00       2.00 r
data arrival time                                      2.00
clock clkA (rise edge)                     10.00      10.00
clock network delay (ideal)                 0.00      10.00
clock uncertainty                           0.00      10.00
FFA2/CP (FF)                                0.00      10.00 r
library setup time                          0.00      10.00
data required time                                    10.00
-------------------------------------------------------------
data required time                                    10.00
data arrival time                                     -2.00
-------------------------------------------------------------
slack (MET)                                           8.00
```

Overriding Single-Cycle Timing

Single-cycle timing might not be appropriate for every path in your design.

The following path-based commands can be used to override single-cycle timing on a path:

- `set_false_path`
- `set_max_delay`
- `set_min_delay`
- `set_multicycle_path`
- `reset_path`

When used to override single-cycle timing, these path-based commands are called timing exception commands.

Timing Exception Command Storage and Expansion

Timing exception commands operate on a “from” set, a “to” set, and several “through” sets. The paths covered by these sets go through one element from every set — that is, the cross product of these sets. Therefore, moderate numbers of sets with moderate set sizes can yield large numbers of paths.

In releases prior to version 1999.05, Design Compiler stored exceptions in the .db file by explicitly expanding the given sets into the cross product, generating and storing the resulting large numbers of paths. When this method of storage and expansion was

used, the CPU time and memory needed to analyze and optimize the design could be extensive, increasing with the number of exceptions specified.

To resolve these problems, beginning with version 1999.05, Design Compiler stores exception commands by default in the .db file in compressed format—that is, without expanding the path — and it interprets them accordingly.

For example, if a dc_shell script contained the following command,

```
set_false_path  -from {a,b} -through {c,d} -to {e,f}
```

Design Compiler would store the result of issuing the command as one compressed path. In releases prior to version 1999.05, Design Compiler would store the result as eight explicit paths.

Using compressed paths for internal storage results in less memory usage and quicker exception load time.

For .db files created prior to version 1999.05 of Design Compiler, exceptions are stored in the .db file in expanded form and Design Compiler version 1999.05 treats them as such. Design Compiler version 1999.05 also stores exceptions in expanded form if you type them in that way.

Design Compiler writes out commands in the form in which they are stored in memory. However, if you want to expand timing exceptions stored in compressed form, using version 1999.05, you can issue the `report_timing_requirements` command with its `-expanded` option. The expanded option is a switch that causes the `report_timing_requirements` command to report in expanded form all exceptions stored in memory in compressed form.

Specifying Timing Exception Commands

Timing exception commands use the `-from`, `-to`, or `-through` option to modify paths. The `-from` option indicates a path startpoint, `-to` indicates a path endpoint, and `-through` enables control over a specific path or multiple paths between a given startpoint and endpoint.

For example,

```
set_max_delay 3 -through find (cell, "U1")
```

This example shows how you would set a maximum delay of 3 for all paths passing through cell U1. (See “Defining Maximum Delay for Paths” on page 2-59 for the full syntax of the `set_max_delay` command.)

You can enter leaf-level pins, leaf-level cells, hierarchical pins, or top-level ports as through points. If you specify leaf-level pins or cells, Design Compiler marks them as “size-only” during optimization. The size-only marker prevents the through points from being optimized out of existence.

Precedence Assessment When Multiple Exceptions Satisfy a Path

If a given path satisfies more than one timing exception, Design Compiler follows these rules to determine which exceptions take precedence:

- set_false_path > set_max_ or set_min_delay > set_multicycle_path
- pin > clock

- -from > -to > -through
- tighter constraint > looser constraint

More specifically, Design Compiler uses the following rules—in this order—to prioritize the exceptions.

1. `group_path` exceptions and other exceptions are exclusive—that is, two group_path exceptions can conflict with each other, but a group_path exception does not conflict with any other type of exception. (Thus, the remaining rules apply for two group_path exceptions or two non-group_path exceptions.)
2. If both exceptions are false paths, there is no conflict.
3. If one exception is a `max_delay` and another is a `min_delay`, there is no conflict.
4. If one exception is a `set_multicycle_path -hold` and another is a `set_multicycle_path -setup`, there is no conflict.
5. If one exception is a `false_path` and the other is not, the `false_path` takes precedence.
6. If one exception is a `max_delay` and the other is not, the `max_delay` takes precedence.
7. If one exception is a `min_delay` and the other is not, the `min_delay` takes precedence.
8. If one exception has a `-from` pin and another does not, the exception with the `-from` pin takes precedence.
9. If one exception has a `-to` pin and another does not, the exception with the `-to` pin takes precedence.

10. If one exception includes a `-through` and another does not, the `-through` exception takes precedence.
11. If one exception has a `-from` clock and another does not, the exception with the `-from` clock takes precedence.
12. If one exception has a `-to` clock and another does not, the exception with the `-to` clock takes precedence.
13. The exception with the more restrictive constraint takes precedence. For `set_max_delay` and `set_multicycle_path -setup`, the more restrictive constraint is the one with the lower value. For `set_min_delay` and `set_multicycle_path -hold`, the more restrictive constraint is the one with the higher value.

For example, consider a path with two exceptions: a false_path and a `max_delay`. Design Compiler works down the list of rules until it gets to rule 5: The `false_path` takes precedence. There is no conflict with rule 6 because the answer is found, so the analysis stops at rule 5.

Setting False Paths

A false path is a timing path that cannot propagate a signal. You can use the `set_false_path` command to identify these paths. Timing constraints are removed from these paths so that the Design Compiler optimizer does not optimize them. Note that setting false paths increases runtime.

The `set_false_path` command is a path-based command. False path information takes precedence over multicycle path information.

The syntax is

```
set_false_path [-setup | -hold] [-rise | -fall]
     [-from  from_list ] [-through  through_list ]
    [-to  to_list ][-reset_path]
```

Note:

For more information, see the `set_false_path` man page.

Example

In Figure 2-12 on page 2-46, if the path from FFB1/CP to FFB2/D cannot occur during normal circuit operation, you can set the path to false. Enter

```
dc_shell> set_false_path -from FFB1/CP -to FFB2/D
```

To undo a `set_false_path` command, use either the `reset_design` command or the `reset_path` command with similar options. For example, setup paths from IN2 to FF12 are marked false. To restore this path to single-cycle timing, enter the `reset_path` command:

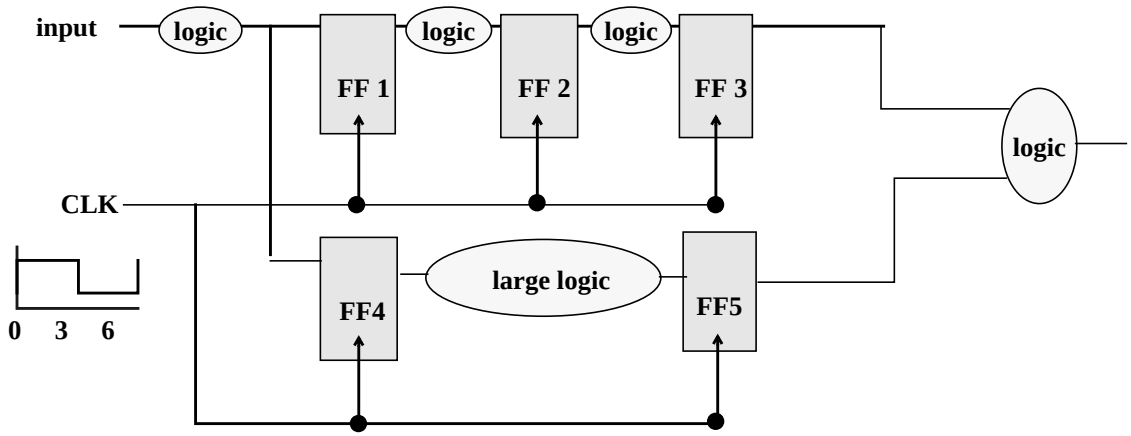
```
dc_shell> set_false_path -setup -from IN2 -to FF12
dc_shell> reset_path -setup -from IN2 -to FF12
```

Setting Multicycle Paths

A multicycle path is a timing path that is not expected to propagate a signal in one cycle. Normally, all paths are constrained for single-cycle timing. Multicycle paths are exceptions to the default single-cycle timing. Set multicycle paths to direct Design Compiler to allow two clock cycles for data to propagate along a path.

Figure 2-13 on page 2-54 shows a two-cycle path from FF4 to FF5.

Figure 2-13 Multicycle Path



The `set_multicycle_path` command modifies the single-cycle timing relationship of a constrained path. It designates timing paths that have nondefault setup or hold relations.

The `set_multicycle_path` command is a path-based command. False path information takes precedence over multicycle path information. Specific `set_max_delay` or `set_min_delay` commands override a general `set_multicycle_path` command.

Setting multicycle paths increases runtime.

The syntax is

```
set_multicycle_path    path_multiplier
                        [-setup | -hold]
                        [-rise | -fall]
                        [-start | -end]
                        [-from  from_list ]
                        [-through  through_list ]
                        [-to  to_list ]
                        [-reset_path]
```

Note:

For more information, see the `set_multicycle_path` man page.

To undo a `set_multicycle_path` command, use the `reset_path` or `reset_design` command.

To disable setup or hold calculations for paths, use the `set_false_path` command.

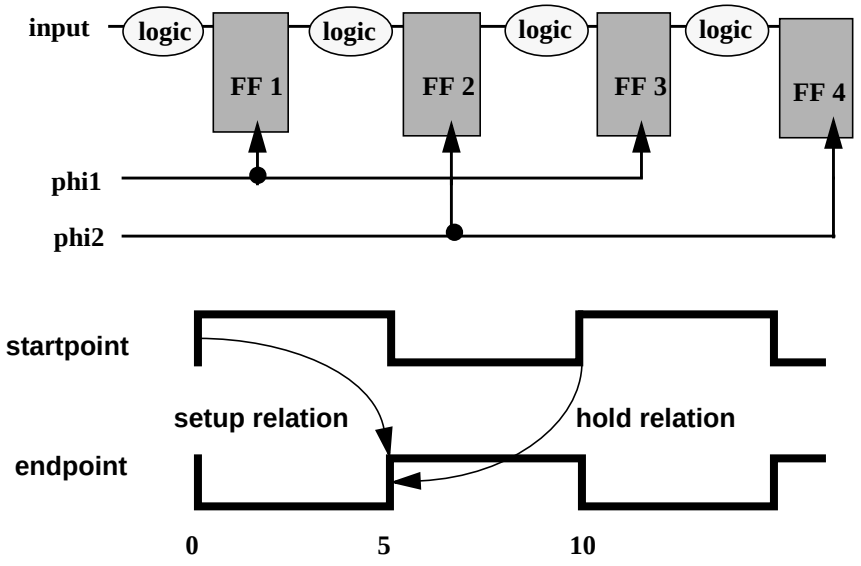
To list the point-to-point exceptions, use the `report_timing_requirements` command.

Defining Multicycle Paths Examples

Examples of `set_multicycle_path` commands and their effects follow.

Figure 2-14 shows the default setup and hold relation for the two-phase design used in this section.

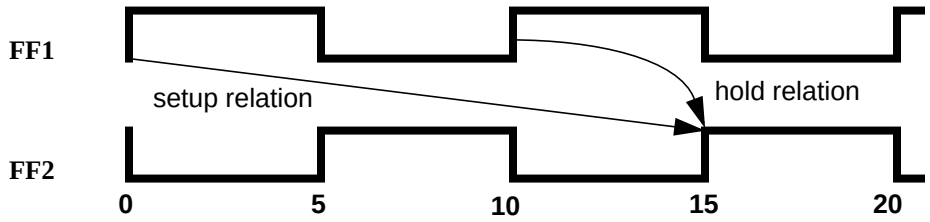
Figure 2-14 Two-Phase Design With Default Setup and Hold



Setup Multiplier 2 and Hold Multiplier 0 Example

To define setup multiplier 2 for the path from FF1 to FF2 but maintain a hold multiplier 0, enter

```
set_multicycle_path 2 -from FF1 -to FF2
```

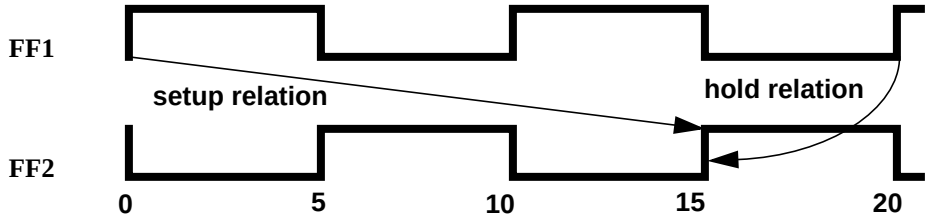


The hold multiplier of 0 signifies that the next active edge at the path startpoint is checked against the same active edge at the path endpoint.

Setup Multiplier 2 and Hold Multiplier 1 Example

To define a setup multiplier of 2 for the path from FF1 to FF2 and a hold multiplier of 1, enter

```
set_multicycle_path 1 -hold -from FF1 -to FF2
```



The hold multiplier 1 signifies that the next active edge plus 1 at the path startpoint is checked against the same active edge at the path endpoint.

Defining Setup and Hold for Multifrequency Designs Examples

Examples of `set_multicycle_path` commands and their effects follow.

Figure 2-15 shows the multifrequency design used in this section.

Figure 2-15 Multifrequency Design

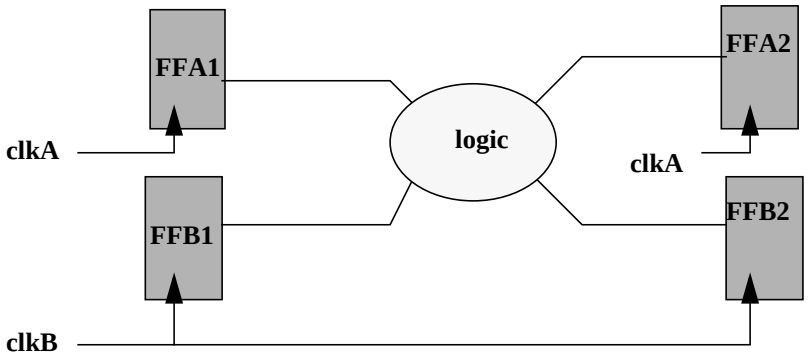
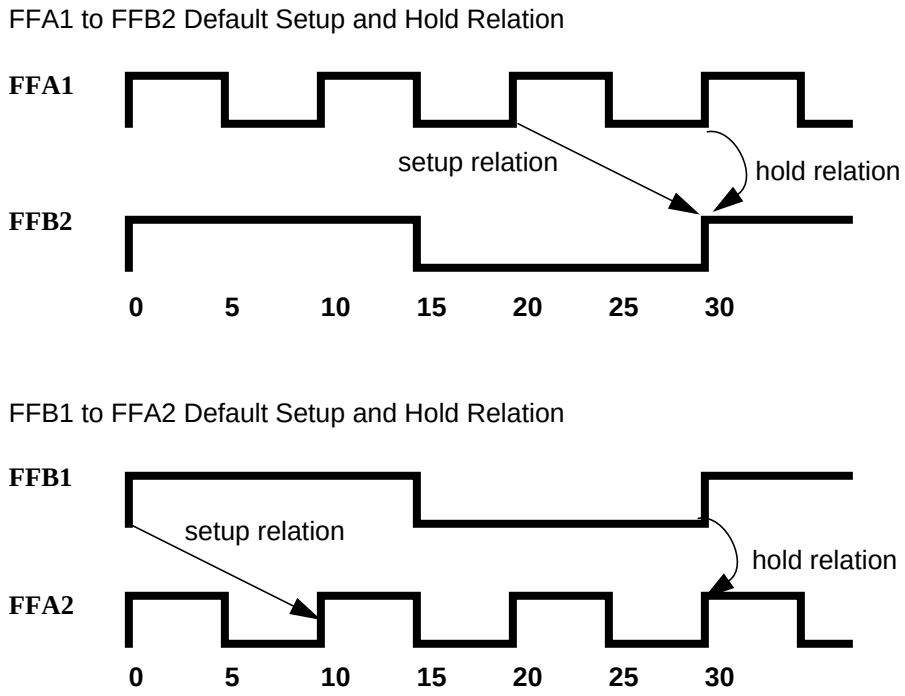


Figure 2-16 on page 2-58 shows the default setup and hold relations for the path FFA1 to FFB2.

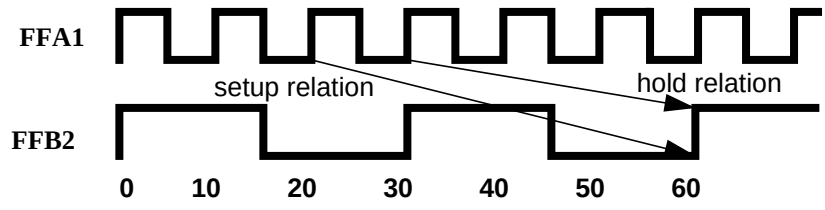
Figure 2-16 Default Setup and Hold Relations



Defining the Setup Relation Relative to Clock at the Endpoint

To define a setup multiplier of two cycles for the path from FFA1 to FFB2 relative to clkB (the clock at the endpoint), enter

```
set_multicycle_path 2 -from FFA1 -to FFB2
```

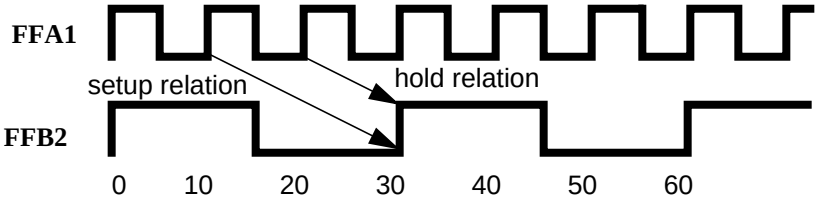


The new setup relation is 40 units (the original 10 plus an extra cycle of clkB). With the hold multiplier set to 0, the hold relation is 30 units, because the next edge of clkA is 30, and the same edge of clkB is 60.

Defining the Setup Relation Relative to Clock at the Startpoint

To define a setup multiplier of 2 relative to the clock at the startpoint, enter

```
set_multicycle_path 2 -start -from FFA1 -to FFB2
```



The setup relation is 20 (the original 10 plus an extra cycle of clkA). The hold relation is 10 units, because the next edge of clkA is 20, and the same edge of clkB is 30.

Defining Maximum Delay for Paths

The `set_max_delay` command specifies a maximum delay for timing paths.

Use `set_max_delay` in two cases:

- To set a target delay for output ports in a combinational design
- To override the default single-cycle timing for paths where `set_multicycle_path` is not sufficient

When you set a maximum delay,

- If a path startpoint is on a sequential device, clock skew is included in the computed delay.
- If a path startpoint has an input delay specified, that delay value is added to the path delay.
- If a path endpoint is on a sequential device, clock skew and library setup time are included in the computed delay.
- If the endpoint has an output delay specified, that delay is added into the path delay.

A specific `set_max_delay` command overrides a general `set_multicycle_path` timing command.

The syntax is

```
set_max_delay     delay_value  [-rise | -fall]
                  [-from  from_list ]
                  [-through  through_list ]
                  [-to  to_list ]
                  [-group_path  group_name ]
                  [-reset_path]
```

Note:

For more information, see the `set_max_delay` man page.

Examples

To specify that the paths to all output ports in a combinational design are less than 15 units, enter

```
dc_shell> set_max_delay 15 -to all_outputs()
```

To specify a maximum delay value of 3.7 for the path from port Reset to pin FF1/CDN, enter

```
dc_shell> set_max_delay 3.7 -from Reset -to FF1/CDN
```

To specify a maximum delay of 3 for all paths passing through cell U1, enter

```
dc_shell> set_max_delay 3 -through find (cell, "U1")
```

To specify a maximum delay of 3 for all paths that first pass through cell U1 and later pass through cell U2, enter

```
dc_shell> set_max_delay 3 -through find (cell, "U1")\
          -through find (cell, "U2")
```

To undo a `set_max_delay` command, use the `reset_path` command with similar options. For example, to undo the previous commands, enter

```
dc_shell> reset_path -to all_outputs()
dc_shell> reset_path -from Reset -to FF1/CDN
```

To modify paths grouped with the `-group_path` option, use the `group_path` command to place the paths in another group or the default group.

Defining Minimum Delay for Paths

The `set_min_delay` command sets a target minimum delay for timing paths.

Use the `set_min_delay` command to

- Set a target minimum delay for the output ports in a combinational design.
- Override the default hold relation for the paths in a sequential design, where `set_multicycle_path` is not sufficient.

To set minimum delay, select the pins or ports, then set the constraint. The unit of delay (ns, ps, and so forth) is technology-dependent and defined in the technology library.

Calculations consider the following information:

- If a path startpoint is on a sequential device, clock skew is included in the computed delay.
- If a path startpoint has an input external delay specified, the delay value is added to the path delay.
- If a path endpoint is on a sequential device, clock skew and library setup time are included in the computed delay.
- If the endpoint has an output external delay specified, the delay is added to the path delay.

The syntax is

```
set_min_delay     delay_value  [-rise | -fall]
                  [-from  from_list ]
                  [-through  through_list ]
                  [-to  to_list ] [-reset_path]
```

Note:

For more information, see the `set_min_delay` man page.

Example

To set the minimum delay to 3.5 for output ports in a combinational design, enter

```
dc_shell> set_min_delay 3.5 -to all_outputs()
```

To set a minimum delay of 6.3 for the path from ff12/CP and ff13/CP to ff14/D, enter

```
dc_shell> set_min_delay 6.3 -from {ff12/CP ff13/CP} -to ff14/D
```

To undo a `set_min_delay` command, use the `reset_path` command with similar options. For example, to undo the previous commands, enter

```
dc_shell> reset_path -to all_outputs()
dc_shell> reset_path -from {ff12/CP ff13/CP} -to ff14/D
```

Resetting Paths to Single-Cycle Timing

The `reset_path` command returns paths to the default single-cycle timing. Use `reset_path` to remove multicycle information set with the following commands:

`set_max_delay`

`set_min_delay`

`set_false_path`

`set_multicycle_path`

The syntax is

```
reset_path        [-setup | -hold] [-rise | -fall]
                  [-from  from_list]
                  [-through through_list]
                  [-to  to_list ]
```

Note:

For more information, see the `reset_path` man page.

Resetting the Paths to All the Output Ports

To reset paths to all the output ports, enter

```
dc_shell> reset_path -to all_outputs()
```

Resetting the Paths to Specific Internal Pins for Setup

To reset paths to three named internal pins for setup, enter

```
dc_shell> reset_path -setup -to {u1/D u2/D u3/D}
```

Using Case Analysis to Set Constant Paths

Case analysis provides a means of setting constant paths that you want Design Compiler to ignore during timing analysis and optimization but not to remove from the design as a result of the optimization. That is, case analysis defines constant paths only for the purposes of timing analysis and costing; it does not remove them. These paths remain functional in the optimized design.

You use the `set_case_analysis` command to assign case analysis constants to specified ports or pins of either a GTECH or mapped design. A case analysis constant can take the value 0 or 1. The constant value is assigned as a `user_case_value` attribute to the user-specified port or pin.

Design Compiler propagates forward each case analysis constant from the specified port or pin to establish a constant path if either of the following two conditions is met:

- The constant value is a controlling value.
- The constant value is not a controlling value, but all other inputs to the gate are constant.

The constants are only propagated forward. That is, case analysis does not propagate the constant values specified on the ports or pins backward through the design. Case analysis constants can propagate through hierarchy.

Constant path propagation ends at cells where a nonconstant output can occur. The propagated constant values are assigned as `case_value` attributes to the propagated pins.

If you also specify logic constants by using the `set_logic_one` and `set_logic_zero` commands, case analysis treats these constants as case analysis constants for the purposes of timing and costing the design. Also, Design Compiler ties unconnected pins to constants, and these constants can be used to disable arcs based on case analysis. However, logic constant paths are still simplified during optimization, whether or not you use the case analysis feature.

Note:

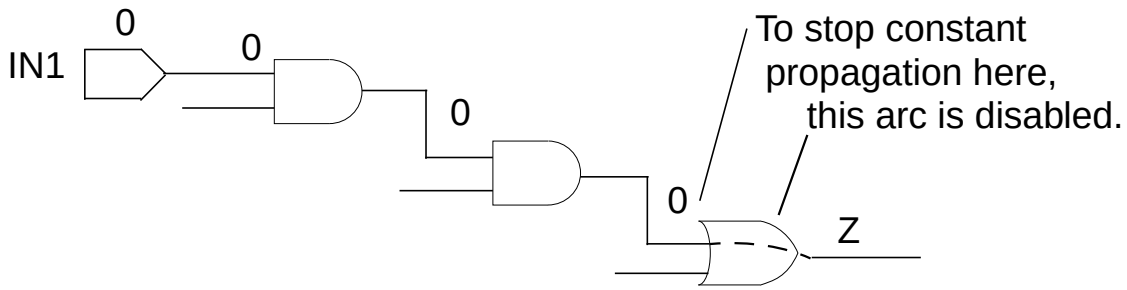
Design Compiler case analysis supports conditional arcs for cells with “when” conditions.

In Design Compiler, the case analysis constants are not propagated through sequential cells. Also, case analysis does *not* support a rise/fall option.

To ensure that the constant paths are not constrained during analysis or optimization, Design Compiler disables the final arc in each constant path. All other arcs of the path remain enabled. Case analysis does *not* place `size_only` or a `dont_touch` attribute on the cells with disabled timing arcs.

Figure 2-17 shows a typical combinational path with a disabled final arc.

Figure 2-17 A Disabled Combinational Path



In this example, to set a logic constant of 0 at the IN1 port, you enter

```
dc_shell> set_case_analysis 0 [select_port "IN1"]
```

Other disabled arcs occur in complex combinational cells, multiplexers, and three-state cells.

Figure 2-18 shows an example of a disabled AND-OR gate. Using case analysis, a constant 0 is placed on input B. This implies that a constant 0 should be propagated to the internal node n1, but the output at Z is not constant because input C to the OR-gate part of the complex cell is not constant. In this case, the arc from input A to output Z is not synthesizable, and it cannot affect the worst timing path to Z. Therefore, this path is disabled.

Figure 2-18 A Disabled Complex Combinational Cell

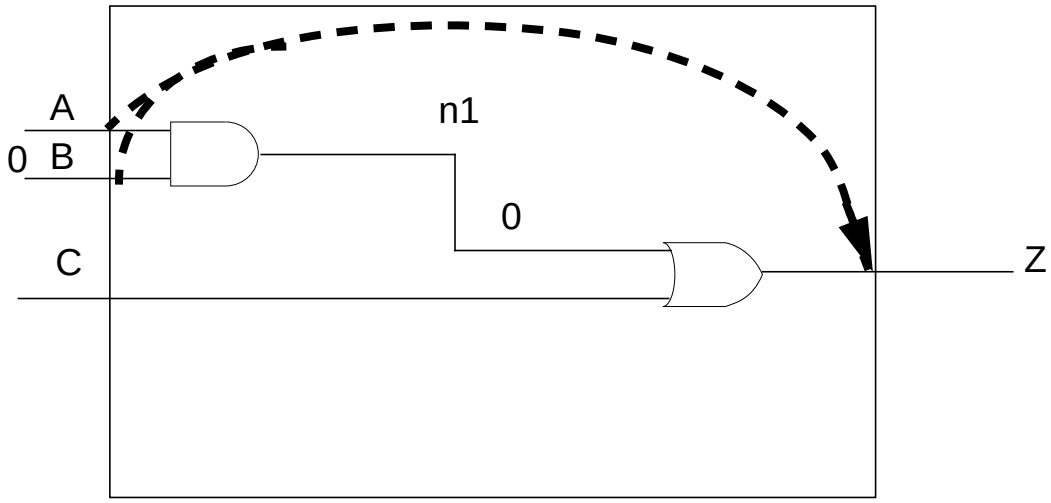
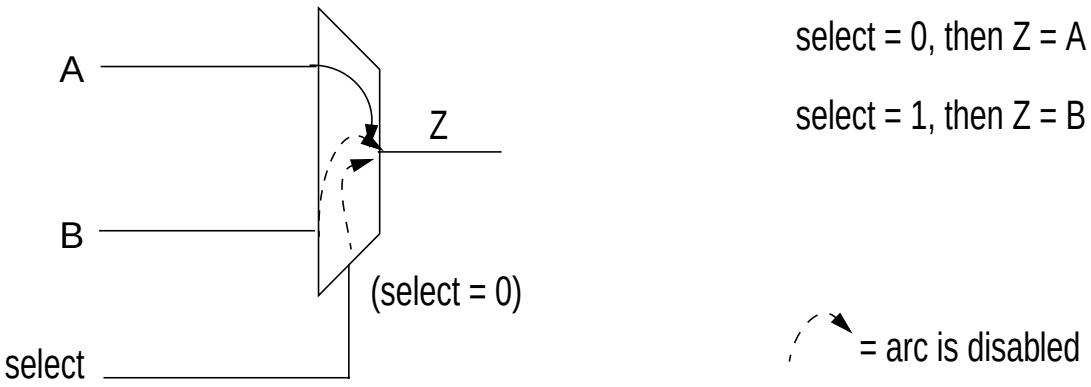


Figure 2-19 shows an example of a multiplexer with input signals A and B and output signal Z. Applying case analysis, a logic 0 is set on the select line. This select line value disables the arc from B to Z, blocking the data from B to Z. If the constant is changed to logic 1, the arc from A to Z is disabled.

Figure 2-19 Constant Blocking of a Multiplexer Data Line



In addition, for three-state cells, if a case analysis constant value of 0 (1) is present on an active high (low) enable pin, the delay arcs from the data input to the three-state output pin are disabled.

The syntax for the `set_case_analysis` command is

```
set_case_analysis [0|1] port_or_pin_list
```

For more information about the `set_case_analysis` command and its options, see the man page for this command.

Removing Case Analysis Attributes

To remove the case analysis constant values placed on user-specified ports and pins (the `user_case_value` attributes), you use the `remove_case_analysis` command. For example, to remove the case analysis attributes from input port IN1, enter

```
dc_shell> remove_case_analysis {"IN1"}
```

Note:

Any propagated case analysis constants (the `case_value` attributes) that were previously propagated from these user-specified ports or pins are also removed.

Controlling Case Analysis Constant Propagation

By default, Design Compiler propagates forward both case analysis and logic constants along timing paths. You can control this property with respect to case analysis by setting the variable `disable_case_analysis`. For example, to disable propagation of case analysis constants, enter

```
dc_shell> set disable_case_analysis true
```

To enable constant propagation, set this variable to false.

Note:

Setting the `disable_case_analysis` variable does not affect the propagation of constants set by other means (tied-high and tied-low pins or `set_logic_one` and `set_logic_zero` constants).

Reporting Case Analysis Results

Use the `report_case_analysis` command to get a report that lists all the ports and pins on which you specified case analysis constants (`user_case_value` attributes).

To report the case analysis values, enter

```
dc_shell> report_case_analysis
```

Design Compiler displays a report similar to this:

```
****************************************
Report : case_analysis
Design : top
Version: 2000.11-SI1
Date   : Mon Sep 11 11:25:07 2000
****************************************

    Pin name          User case analysis value
--------------------------------------------------
    A                 1
    B                 1
    C                 1

    Pin name          Logic constant value
--------------------------------------------------
    D                 1
    E                 1
    F                 1
```

Note that this report does not list any propagated case analysis constants (`case_value` attributes). To report both user-specified and propagated case analysis constants, include the `-all` option in the `report_case_analysis` command. That is, enter

```
dc_shell> report_case_analysis -all
```

Design Compiler displays a report similar to this:

```
****************************************
Report : case_analysis
Design : top
Version: 2000.11-SI1
Date   : Mon Sep 11 11:25:07 2000
****************************************

    Pin name          User case analysis value
-------------------------------------------------
    A                 1
    B                 1
    C                 1

    Pin name          Logic constant value
-------------------------------------------------
    D                 1
    E                 1
    F                 1

    Pin name          Case analysis value
-------------------------------------------------
    U9/A              0
    U9/B              0
    U9/Z              1
    Y_reg/CR          1
    Y_reg/D           1
    U10/A             1
    U10/B             1
    U10/C             1
    U10/Z             0
    U11/A             1
    U11/B             1
    U11/C             1
    U11/Z             0

    Pin name          Internal logic constant
-------------------------------------------------
    Logic1/output     1
    *cell*2/output    1
    *cell*3/output    1
    *cell*4/output    1
```

Also, you can use the `report_disable_timing` command to report the arcs disabled for timing analysis.

To report the case analysis values, enter

```
dc_shell> report_disable_timing -nosplit
```

Design Compiler displays a report similar to this:

```
****************************************
Report : disable_timing
Design : top
Version: 2000.11-PROD
Date   : Wed Nov 1 17:25:07 2000
****************************************

Flags:        c    case-analysis
              l    loop breaking
              u    user-defined

Cell or Port       From       To         Sense                Flag
--------------------------------------------------------------------------------
  Y_reg            CP         CR         setup_clk_rise       c
  Y_reg            CP         CR         hold_clk_rise        c
  Y_reg            CP         D          setup_clk_rise       c
  Y_reg            CP         D          hold_clk_rise        c
  Z                                                           c
```

Generating a Log File of Constant Propagation

Because case analysis constant propagation only disables the timing arc at the boundary of the path domain, knowing the origin of the constant value propagated to a given pin of the design can be difficult. To help you determine the timing arcs disabled by case analysis, you can set the `case_analysis_log_file` variable to a file name. The constant propagation process is logged to the named file.

Note:

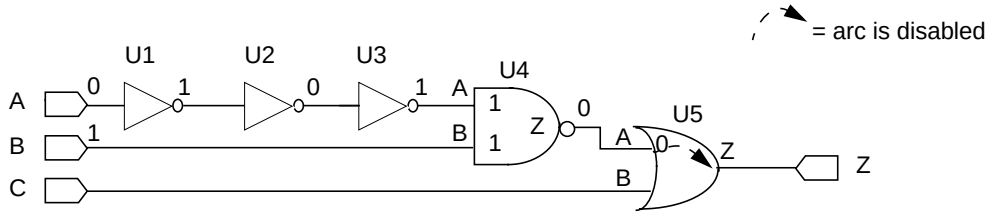
Design Compiler updates the case analysis constants incrementally. Therefore, as long as the case analysis log file is opened, every incremental change in the constants is recorded. To close the log file, set `case_analysis_log_file = ""`.

These reporting capabilities are presented in the following example.

Example

This command sequence sets case analysis constants 0 and 1 on input ports A and B, respectively, and reports the disabled timing arcs to the log file my_design_cnst.log for the example circuit.

```
dc_shell> set_case_analysis 0 {get_port "A"}
dc_shell> set_case_analysis 1 {get_port "B"}
dc_shell> set case_analysis_log_file "my_design_cnst.log"
```



In this example circuit, Design Compiler propagates the constants 0 and 1 from A and B, respectively, to U5/A. However, Design Compiler disables the arc in U5 from U5/A to U5/Z because the output at U5/Z is not constant due to the nonconstant input from C.

The constant propagation log file my_design_cnst.log looks like this:

```
****************************************
Report : case_analysis propagation
Design : my_design
****************************************


Forward propagation of constants
--------------------------------
  Propagation of logic '0' from pin 'A' on net 'A'
  Propagation of logic '1' from pin 'B' on net 'B'
  Propagation of logic '0' from pin 'A' on net 'A'
     > pin 'U1/A' is at logic '0'
  Propagation of logic '1' from pin 'B' on net 'B'
     > pin 'U4/B' is at logic '1'
  Cell 'U1' (libcell 'IV')
     input pin 'U1/A' is at logic '0'
     > output pin 'U1/Z' is at logic '1'
  Propagation of logic '1' from pin 'U1/Z' on net 'w1'
     > pin 'U2/A' is at logic '1'
  Cell 'U2' (libcell 'IV')
     input pin 'U2/A' is at logic '1'
     > output pin 'U2/Z' is at logic '0'
  Propagation of logic '0' from pin 'U2/Z' on net 'w2'
     > pin 'U3/A' is at logic '0'
  Cell 'U3' (libcell 'IV')
     input pin 'U3/A' is at logic '0'
     > output pin 'U3/Z' is at logic '1'
  Propagation of logic '1' from pin 'U3/Z' on net 'w3'
     > pin 'U4/A' is at logic '1'
  Cell 'U4' (libcell 'ND2')
     input pin 'U4/A' is at logic '1'
     input pin 'U4/B' is at logic '1'
     > output pin 'U4/Z' is at logic '0'
  Propagation of logic '0' from pin 'U4/Z' on net 'w4'
     > pin 'U5/A' is at logic '0'
```

Finally, the disable timing report looks like the following example.

```
report_disable_timing

****************************************
Report : disable_timing
Design : my_design
Version: 2000.11-PROD
Date   : Wed Nov  1 17:29:41 2000
****************************************


Flags:       c  case-analysis
             l  loop breaking
             u  user-defined

  Cell or Port        From        To             Sense               Flag
--------------------------------------------------------------------------------

  U5                  A           Z              positive_unate      c

1
```

A Compile Methodology Using Case Analysis

Here is a simplified compile methodology that uses case analysis.

```
read_verilog design_B_rtl.v
current_design design_B
link
source design_B_constraints.tcl
set_case_analysis 1 [get_ports {...}]
set_case_analysis 0 [get_ports {...}]
set disable_case_analysis false
set case_analysis_log_file “design_B_case_analysis.log”
compile
report_case_analysis -all
report_disable_timing
report_timing
...
remove_case_analysis [get_ports {...}]
...
```

For more information on these commands and environment variables, see the appropriate man pages.

Using Mode Analysis to Set Active Modes

A complex circuit can have multiple timing paths that are dependent on the enabled functional modes of certain cells in the circuit. (For example, a RAM cell with read and write modes has different timing properties depending on which mode is enabled.) The timing analysis for your design depends on how Design Compiler handles the cell instances with functional modes.

In previous releases, you could not control which modes Design Compiler used when performing timing analysis. For designs with cell instances taken from Synopsys .lib library cells, all modes were enabled for the purposes of timing analysis, irrespective of functionality. The most restrictive arcs were used in the analysis. Therefore, in some situations, the analyzed timing paths might not be meaningful, or the timing analysis might be too conservative.

For designs with cell instances taken from DCL library cells, only the predefined default mode could be used in the timing analysis. This meant that for cells with multiple modes, you could not perform timing analysis for other modes.

Mode analysis uses the `set_mode` command to control which modes are enabled for a timing analysis. For cell instances taken from a Synopsys .lib library, you can restrict the timing analysis to paths associated with specific functional modes by using this command to enable and disable one or more modes. For cell instances taken from a DCL library, you are still limited to performing a single mode timing analysis; but using this command, you can select modes other than the default mode.

Defining Active Modes for Cell Instances

You define the active modes of the individual cell instances of a design by using the `set_mode` command. The command syntax is

```
set_mode [mode_list][instance_list]
```

Basically, you list the modes you want to enable for timing analysis in the *mode_list*, and these modes are enabled for the cell instances of the *instance_list*.

For DCL library cells only one mode is listed and enabled. For Synopsys .lib library cells, you can list one or more modes in the mode list. But in the case of Synopsys .lib library cells, how Design Compiler interprets the mode list depends on whether the library cells with functional modes were organized into mode groups.

Generally, for mode groups, unrelated or functionally exclusive modes are grouped together. For example, a RAM block often has read, write, latch, and transparent modes. These modes are usually grouped into sets of mutually exclusive functions, such as a read-write mode group for the read and write modes and a latch-transparent group for the latch and transparent modes. Before performing timing analysis, you can use the `set_mode` command to specify the read and transparent modes as the two active modes for

the RAM block. The write and latch modes are implicitly disabled, and the subsequent timing analysis is restricted to the paths associated only with the read and transparent modes. Thus, to enable and disable these modes for the “U1/U2/core” cell, you would enter

```
dc_shell> set_mode {read, transparent} U1/U2/core
```

Specifically, the mode-list rules for Synopsys .lib library cells are as follows:

- If the library cells are not organized into mode groups, only the modes you list in the `set_mode` command are enabled for the cell instances of your design. All unlisted modes are disabled.
- If the library cells do have mode groups, only one mode from a group should be listed.
- Those modes listed in the `set_mode` command that belong to mode groups are enabled, and all other (unlisted) modes belonging to these groups are automatically disabled.
- The mode groups are independent, that is, setting the mode for one mode group has no effect on any other mode group.
- Mode groups that are *not* represented by a mode in the mode list maintain their previous or default mode setting. That is, enabled modes belonging to unreferenced mode groups remain enabled.

Note:

A DCL library does not use a mode group configuration.

For more information, see the `set_mode` man page.

Reporting Modes for Cell Instances

The `report_mode` command reports all cell instances that have modes and specifies whether each mode is enabled or disabled. If the cell has mode groups defined, the modes are listed by the mode group they belong to.

To produce a mode report, enter

```
dc_shell> report_mode
```

Design Compiler displays a report similar to this for cell instances taken from a Synopsys .lib library:

```
****************************************
Report : mode
Design : misc
****************************************

Cell                                    Mode(group)          Status
--------------------------------------------------------------------------------
U1/U2/core(RAM1_core)                   read(rw)             DISABLED
                                        write(rw)            ENABLED
--------------------------------------------------------------------------------
                                        latching(out_latch)  ENABLED
                                        transparent(out_latch)DISABLED
--------------------------------------------------------------------------------
U1/U4/core(RAM1_core)                   read(rw)             ENABLED
                                        write(rw)            DISABLED
--------------------------------------------------------------------------------
                                        latching(out_latch)  ENABLED
                                        transparent(out_latch)DISABLED
--------------------------------------------------------------------------------
```

This report shows that rw and out_latch mode groups were defined for both cells and that only one mode in each mode group is enabled.

For more information see the `report_mode` man page.

Resetting Functional Modes on Cell Instances

The `reset_mode` command resets the modes of the cell instances of a design. After the command is issued, the default settings are in effect. That is, for instances of Synopsys .lib library cells, all modes are enabled for timing analysis, and for instances of DCL library cells, only the default mode is enabled.

For more information see the `reset_mode` man page.

Applying Time Borrowing to Level-Sensitive Latches

Design Compiler applies special timing, called time borrowing, to designs containing level-sensitive latches. A latch is transparent for the duration of the active clock pulse, meaning that data appearing on the input is propagated directly to the output of the latch.

Time borrowing is a property used in timing level-sensitive latches. If the path leading to the data pin of a latch is too long, sometimes time can be “borrowed” from the next cycle because the latch is enabled for a certain length of time.

Design Compiler automatically uses time borrowing when borrowing can reduce a setup violation on a latch. Time can be borrowed from either the previous stage or the next stage in a level-sensitive design. You can limit time borrowing.

Time borrowing can increase Design Compiler runtime for multifrequency designs with level-sensitive latches.

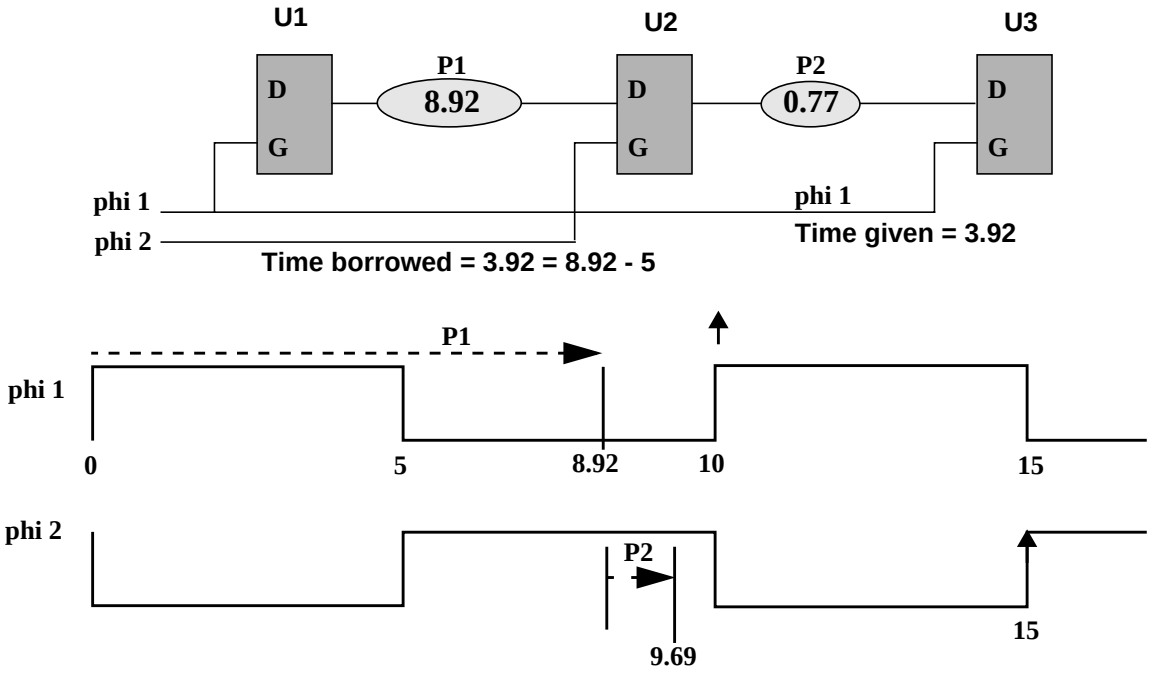
Note:

The DC Professional tool from Synopsys does not include time borrowing.

Example

Figure 2-20 shows a latch-based design using a simple two-phase clocking scheme. The timing report follows the explanation of Figure 2-20.

Figure 2-20 Latch-Based Design



U1, U2, and U3 are positive level-sensitive latches (active when G=1). P1 and P2 are combinational logic paths. Assuming a library setup time of zero for the latches and path lengths of P1 = 8.92 and P2 = 0.77, it appears that there is a violation at U2. Path P1 is longer than the rising edge of phi2. However, the latch enable time of phi2 is 5, and path P1 can use this time if path P2 has enough slack.

Note:

Although one latch-to-latch path might be greater than the period of the clock, any two successive paths must be less than twice the period for a single-phase design.

The timing report for the latch-based design follows. In this report, **bold** type helps you locate the time-borrowing information.

```
****************************************
Report: timing
        -path short
        -delay max
        -max_paths 1
Design: time_borrow
Version: v3.3b
Date  :Tues Jan 14 1997
****************************************
Wire Loading Model Mode: enclosed
  Startpoint: U1 (positive level-sensitive latch clocked by Phi1)
  Endpoint: U2 (positive level-sensitive latch clocked by Phi2)
  Path Group: Phi2
  Path Type: max
  Point                                    Incr       Path
  -----------------------------------------------------------
  clock Phi1 (rise edge)                   0.00       0.00
  clock network delay (ideal)              0.00       0.00
  U1/G (LATCH)                             0.00       0.00 r
  U1/Q (LATCH)                             0.57       0.57 r
  ...
  U2/D (LATCH)                             8.35       8.92 r
  data arrival time                                   8.92
  clock Phi2 (rise edge)                   5.00       5.00
  clock network delay (ideal)              0.00       5.00
  U2/G (LATCH)                             0.00       5.00 r
  time borrowed from endpoint              3.92       8.92
  data required time                                  8.92
  -----------------------------------------------------------
  data required time                                  8.92
  data arrival time                                  -8.92
-----------------------------------------------------------
  slack (MET)                                         0.00

  Time Borrowing Information
  -------------------------------------------
  Phi2 pulse width                         5.00
```

```
  library setup time                      -0.46
  ------------------------------------------------
  max time borrow                          4.54
  actual time borrow                       3.92
  ------------------------------------------------
Startpoint: U2 (positive level-sensitive latch clocked by Phi2)
  Endpoint: U3 (positive level-sensitive latch clocked by Phi1)
  Path Group: Phi1
  Path Type: max
  Point                                    Incr       Path
  -----------------------------------------------------------
  clock Phi2 (rise edge)                   5.00       5.00
  clock network delay (ideal)              0.00       5.00
  time given to startpoint              3.92       8.92
  U2/D (LATCH)                                        0.00       8.92 r
  U2/Q (LATCH)                                        0.53       9.45 r
  ...
  U3/D (LATCH)                                       0.24       9.69 f
  data arrival time                                    9.69
  clock Phi1 (rise edge)                  10.00      10.00
  clock network delay (ideal)             0.00       10.00
  U3/G (LATCH)                                             0.00      10.00 r
  time borrowed from endpoint             0.00       10.00
  data required time                                  10.00
  -----------------------------------------------------------
  data required time                                  10.00
  data arrival time                                   -9.69
  -----------------------------------------------------------
  slack (MET)                                          0.31

  Time Borrowing Information
  ------------------------------------------------
  Phi1 pulse width                         5.00
  library setup time                      -0.49
  ------------------------------------------------
  max time borrow                          4.51
  actual time borrow                       0.00
  ------------------------------------------------
```

For a positive level-sensitive latch, Design Compiler uses the rising edge as the reference edge.

- For the U1 to U2 path, a setup violation appears at U2, because the signal arrives at 8.92 and the clock arrives at 5.00. Because the U2 latch is transparent, 3.92 ns can be borrowed from the following U2 to U3 path and still meet the setup time at the U2 falling edge. The first timing path shows 3.92 ns being borrowed. This is reported in the timing report for the first path.
- For the U2 to U3 path, time must be added to compensate for the time borrowed, so 3.92 ns is added to the U2 launch time. This is reported in the timing report for the second path. Because the P2 path has enough slack, neither path is a violation.

Limiting or Disabling Time Borrowing

In some cases, you might not want time borrowing enabled for all or part of a design. You can limit or disable time borrowing by using the `set_max_time_borrow` command.

The `set_max_time_borrow` command places a `max_time_borrow` attribute of a specified value on clocks, latch cells, data pins, or clock (enable) pins to constrain the amount of time borrowing for level-sensitive latches. To meet delay targets, time borrowing prevents automatic use of all or part of the enabling clock pulse on a latch. If the attribute is set on a cell and the cell is replaced during optimization, the attribute is moved from the cell to the enable pin.

To prevent time borrowing, specify a time limit of 0.0.

The `max_time_borrow` attribute is ignored when it is placed on clocks that affect no latches, on pins other than latch enable pins, and on cells other than latches.

The increase that can occur in timing analysis runtime for multifrequency designs containing level-sensitive latches occurs because Design Compiler considers all dominant pulse relations for paths where there can be time borrowing. These considerations can be time-consuming if the clocks are of very different frequencies (there are numerous pulse relations between such clocks).

The syntax is

```
set_max_time_borrow  time_limit object_list
```

Note:

For more information, see the `set_max_time_borrow` man page.

Preventing Time Borrowing for the Entire Design

To prevent time borrowing for the entire design, enter

```
dc_shell> set_max_time_borrow 0.0 all_clocks()
```

Limiting Time Borrowing to Named Instances

To limit time borrowing to 1.2 units for U1/G and U2/G, enter

```
dc_shell> set_max_time_borrow 1.2 {U1/G U2/G}
```

Undoing a Time-Borrowing Command

To undo a `set_max_time_borrow` command, use the `remove_attribute` command. For example, to undo the previous commands, enter

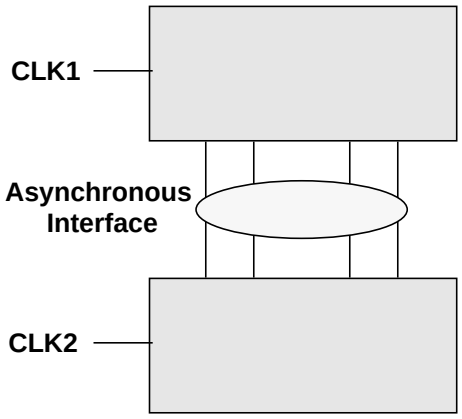
```
dc_shell> remove_attribute all_clocks() max_time_borrow
dc_shell> remove_attribute {U1/G U2/G} max_time_borrow
```

Constraining Designs Containing Asynchronous Logic

Use Design Compiler with synchronous sequential designs and combinational designs. Design Compiler cannot automatically synthesize or verify the behavior of asynchronous logic. However, using hierarchy and delay targets properly permits analyzing designs that contain some asynchronous interface logic.

Figure 2-21 shows a design with an asynchronous interface between two parts, each with its own clock.

Figure 2-21 Design Containing an Asynchronous Interface



Optimize each subdesign independently, with maximum and minimum path delay requirements for the interface logic. Partitioning the design into two synchronous blocks allows optimization and timing analysis to correctly constrain sequential behavior.

For paths between the blocks, use the `set_max_delay` and `set_min_delay` commands to constrain the interblock delays, based on your knowledge of the asynchronous behavior. If possible, separate the asynchronous interface into a third design and optimize it independently. When analyzing the top-level design, use point-to-point `set_max_delay` commands or `set_false_path` commands for the asynchronous paths to direct Design Compiler not to use the default single-cycle timing for these paths.

When the design is complete, you must perform a full functional simulation to verify the asynchronous interface behavior, which cannot be verified adequately by static timing analysis.

Reporting Constraints

The `report_constraint` command reports the constraint values in the current design, enabling you to check design rules and optimization goals.

The constraint report lists the following for each constraint in the current design:

- Whether the constraint was met or violated
- By how much the constraint value was met or violated
- The design object that is the worst violator
- The maximum delay information, showing cost by path group. This includes violations of setup time on registers or ports with output delay as well as violations of the `set_max_delay` command.

- The minimum delay cost, which includes violations of hold time on registers or ports with output delay as well as violations of the `set_min_delay` command.

The syntax is

```
report_constraint      [-all_violators] [-verbose]
                       [-significant_digits  digits ]
                       [-max_area] [-max_delay]
                       [-critical_range]
                       [-min_delay][-max_capacitance]
                       [-min_capacitance]
                       [-max_transition] [-max_fanout]
                       [-cell_degradation] [-min_porosity]
                       [-max_dynamic_power]
                       [-max_leakage_power]
                       [-connection_class]
                       [-multiport_net] [-nosplit]
```

Note:

For more information, see the `report_constraint` man page.

The constraint report summarizes the constraints in the order in which you set the priorities. (Constraints not present in your design are not included in the report.)

For additional examples of the `report_constraint` command, see the man page.

Example

```
dc_shell> report_constraint
****************************************
Report : constraint
Design : counter
Version: v1998.02
Date   : Wed Jan 14 1998
****************************************
                            Weighted
    Group (max_delay/setup)   Cost       Weight     Cost
    -----------------------------------------------------------
    CLK                          0.001.00           0.00
    default                      0.001.00           0.00
    -----------------------------------------------------------
    max_delay/setup                                 0.00
                            Total Neg  Critical
    Group (critical_range)    Slack    Endpoints   Cost
    -----------------------------------------------------
    CLK                          0.00          0      0.00
    default                      0.00          0      0.00
    -----------------------------------------------------
    critical_range                                   0.00
    Constraint                                       Cost
    -----------------------------------------------------
    max_transition                                   0.00 (MET)
    max_fanout                                       0.00 (MET)
    max_delay/setup                                  0.00 (MET)
    critical_range                                   0.00 (MET)
    min_delay/hold                                   0.40 (VIOLATED)
    max_leakage_power                                6.00 (VIOLATED)
    max_dynamic_power                               14.03 (VIOLATED)
    max_area                                        48.00 (VIOLATED)
    min_porosity                                     2.00 (VIOLATED)
```

Listing Detailed Constraint Information

To list more information in a constraint report, use the `-verbose` option of the `report_constraint` command.

Example

```
dc_shell> report_constraint -verbose
****************************************
Report: constraint
        -verbose
Design: counter
Version: v1998.02
Date : Wed Jan 14 1998
****************************************
 Startpoint: ffb (rising edge-triggered flip-flop clocked by CLK)
 Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
  Constraint Group: CLK
  Path Type: max
  Point                                    Incr       Path
  -----------------------------------------------------------
  clock CLK (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  ffb/CP (FD3)                             0.00       0.00 r
  ffb/QN (FD3)                             2.42       2.42 r
  w/Z (ND4)                                0.59       3.01 f
  q/Z (EO)                                 1.13       4.14 f
  j/Z (AO2)                                1.08       5.22 r
  ffd/D (FDS2)                             0.00       5.22 r
  data arrival time                                   5.22
  clock CLK (rise edge)                   10.00      10.00
  clock network delay (ideal)              0.00      10.00
  clock uncertainty                        0.00      10.00
  ffd/CP (FDS2)                            0.00      10.00 r
  library setup time                      -0.90       9.10
  data required time                                  9.10
  -----------------------------------------------------------
  data required time                                  9.10
  data arrival time                                  -5.22
  -----------------------------------------------------------
  slack (MET)                                        3.88
  Startpoint: RESET (input port)
  Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
  Constraint Group: CLK
  Path Type: min
  Point                                    Incr       Path
  -----------------------------------------------------------
  clock (input port clock) (rise edge)     0.00       0.00
  clock network delay (ideal)              0.00       0.00
  input external delay                     0.00       0.00 r
  RESET (in)                               0.00       0.00 r
  ffd/CR (FDS2)                            0.00       0.00 r
  data arrival time                                   0.00
```

```
  clock CLK (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  clock uncertainty                        0.00       0.00
  ffd/CP (FDS2)                            0.00       0.00 r
  library hold time                        0.40       0.40
  data required time                                  0.40
  -----------------------------------------------------------
  data required time                                  0.40
  data arrival time                                   0.00
  -----------------------------------------------------------
  slack (VIOLATED)                                   -0.40
    Design: counter
    max_area              35.00
  - Current Area          78.00
  ------------------------------
    Slack                -43.00  (VIOLATED)
   Design: counter
    min_porosity          30.00
  - Current Porosity      15.00
  ------------------------------
    Slack                -15.00  (VIOLATED)
```

Listing All Constraint Violators

To list all constraint violators, use the `-all_violators` option of the `report_constraint` command.

Example

```
dc_shell> report_constraint -all_violators
****************************************
Report : constraint
        -all_violators
Design : counter
Version: v1998.02
Date   : Wed Jan 14 1998
****************************************
   max_delay ('CLK' group)
                             Required        Actual
   Endpoint                 Path Delay     Path Delay        Slack
   -------------------------------------------------------------------
   ffd/D                        9.10           5.22 r         3.88  (MET)
   min_delay ('CLK' group)
                             Required        Actual
   Endpoint                 Path Delay     Path Delay        Slack
```

```
-------------------------------------------------------------------
ffd/CR                        0.40           0.00 r        -0.40  (VIOLATED)
min_porosity
                            Required        Actual
Design                      Porosity        Porosity        Slack
-------------------------------------------------------------------
counter                       30.00          15.00         -15.00  (VIOLATED)
max_area
                            Required        Actual
Design                        Area            Area           Slack
-------------------------------------------------------------------
counter                       35.00          78.00         -43.00  (VIOLATED)
```

Removing Constraints

Design Compiler provides the `remove_constraint` command and other commands for removing constraints.

The `remove_constraint` command removes user-defined constraints for delay, area, fanout and transition targets, and clocks from the design.

The syntax is

```
remove_constraint -all
```

Constraints removed include information set by the following commands:

`create_clock`

`group_path`

`set_max_area`

`set_max_delay`

`set_min_delay`

```
set_false_path
set_max_fanout
set_max_capacitance
set_max_time_borrow
set_max_transition
set_min_capacitance
set_min_porosity
set_multicycle_path
```

Table 2-3 lists other commands for removing constraints.

Table 2-3 Commands for Removing Constraints

To remove these constraints	Use this command
Individual constraints (Use `report_attribute -design` to list the attributes set on the design.)	`remove_attribute`
All user-defined constraints, attributes, and clocks set on a design	`reset_design`
Designs from Design Compiler memory	`remove_design`
Clock definitions from the design	`remove_clock`
Input delay from an input port	`remove_input_delay`
Output delay from an output port	`remove_output_delay`

See the man pages for detailed information about these commands.

3

Describing the Design Environment

Define the environment in which the design must operate to ensure that acceptable performance levels are maintained over a range of conditions.

In most technologies, variations in operating temperature, supply voltage, and the manufacturing process can strongly affect circuit performance (speed). These factors are called operating conditions. Most technology libraries have predefined sets of operating conditions, timing ranges, and wire load models. The Design Compiler timing analyzer uses this library information when it performs a static delay analysis. This analysis helps you determine whether your design meets performance requirements.

To ensure that acceptable performance levels are maintained over a range of operating conditions, you can direct the optimizer to simulate variations in process, temperature, and voltage.

Wire load modeling estimates the effect of wire length and fanout on resistance, capacitance, and area.

This chapter explains how to describe the environment in which the design must operate. The sections are

- Defining Operating Conditions
- Defining Timing Ranges
- Modeling Wire Loads

Defining Operating Conditions

The operating conditions of a design usually include the process, voltage, and temperature ranges a design encounters. Design Compiler optimizes your design according to an operating point on the process, voltage, and temperature curves and scales cell and wire delays according to your operating conditions. By default, operating conditions are specified in the technology library for your design.

Operating conditions are defined in a technology library in an `operating_conditions` group. The default operating conditions can differ from vendor to vendor. The operating conditions are

Process variation

Accounts for deviations in the semiconductor fabrication process. Usually, the process variation is treated as a percentage variation in any performance calculation.

Voltage variation

Considers deviations of the design supply voltage from the established ideal value during day-to-day operation. Often it involves a complex calculation (using a shift in threshold voltages), but a simple linear scaling factor is also used for logic-level performance calculations.

Ambient temperature variations

Are unavoidable in the everyday operation of a design. Effects on performance because of temperature fluctuation are most often a linear scaling effect, but some submicron silicon processes use a nonlinear calculation.

It is important for Design Compiler to consider the worst-case and best-case scenarios for the expected variations in the process, temperature, and voltage factors.

To see the operating conditions for the library you are using, use the `report_lib` command.

Example

This operating-conditions specification from a technology library defines best-case commercial (BCCOM) operating conditions.

```
operating_conditions("BCCOM") {
    process     : 0.6;
    temperature : 0;
    voltage     : 5.25;
    tree_type   : "best_case_tree";
}
```

Operating Conditions Parameters

The `operating_conditions` parameters are

name

Set of process, temperature, voltage, and tree-type values (for example, BCCOM for best-case commercial).

`process`

Scaling factor to account for variations in the semiconductor manufacturing process. The default value is 1.0.

`temperature`

Ambient temperature under which the circuit operates. A common vendor-defined default is 25 degrees C.

`voltage`

Supply voltage of the design. If the supply voltage is negative, you specify it as the absolute value of the actual voltage. A common vendor-defined default is 5.

`tree_type`

Interconnection model for driving pins and network loads. The supported tree types are `best_case_tree`, `balanced_tree`, and `worst_case_tree`.

Interconnect Models

You can control design analysis by selecting from three wire interconnect models. The interconnect model is defined by the `tree_type` specification in each technology library's set of operating conditions.

The wire interconnect models are

- BEST (best case)
- TYPICAL (balanced case; the default)
- WORST (worst case)

These models reflect different positions of the load pins with respect to the driver. The position of the load pins determines the effect of wire resistance and capacitance on your design. You can explore the possible effects of placement and routing by using these three models.

Example

```
operating_conditions(BEST) {
      process     : 1.1;
      temperature : 11.0;
```

```
      voltage     : 4.6;
      tree_type   : "best_case_tree";
 }
operating_conditions(TYPICAL) {
      process     : 1.3;
      temperature : 31.0;
      voltage     : 4.6;
      tree_type   : "balanced_tree";
 }
operating_conditions(WORST) {
     process      : 1.7;
     temperature  : 55.0;
     voltage      : 4.2;
     tree_type    : "worst_case_tree";
}
```

tree_type Values

`best_case_tree`

Models the case where the load pin is physically adjacent to the driver. All the wire capacitance is incurred, but none of the wire resistance needs to be overcome.

`worst_case_tree`

Models the case where the load pin is at the extreme end of the wire. Each load incurs the full wire capacitance and wire resistance.

`balanced_tree`

Models the case where all load pins are on separate, equal branches of the interconnect wire. Each load incurs an equal percentage of the total wire capacitance and wire resistance.

If you do not define the operating conditions, Design Compiler uses `balanced_tree`.

Setting the Interconnect Model as Part of the Operating Conditions

Use the `set_operating_conditions` command to set the interconnect model as part of the operating conditions. For example, enter

```
dc_shell> set_operating_conditions TYPICAL
Using operating conditions 'TYPICAL' in library 'my_lib'.
```

Listing Operating Conditions Defined in a Technology Library

The `report_lib` command lists the operating conditions defined in a technology library. For example, to generate the report that follows, enter

```
report_lib my_lib
****************************************
Report : library
Library: my_lib
Version: v1997.01
Date   : Tues Jan 14 1997
****************************************

Operating Conditions:

Name   Library   Process   Temp    Volt  Interconnect Model
-------------------------------------------------------------
WCCOM  my_lib    1.50      70.00   4.75  worst_case_tree
WCIND  my_lib    1.50      85.00   4.75  worst_case_tree
WCMIL  my_lib    1.50     125.00   4.50  worst_case_tree
BCCOM  my_lib    0.60       0.00   5.25  best_case_tree
BCIND  my_lib    0.60     -40.00   5.25  best_case_tree
BCMIL  my_lib    0.60     -55.00   5.50  best_case_tree
```

Note:

To see a list of the libraries you have in memory, use the `list_libs` command.

Getting Design Environment Information From the Local Link Library

The `set_wire_load_model`, `set_operating_conditions`, and `set_timing_ranges` commands search the design's local link library before searching the system link library.

During compilation or translation, Design Compiler searches the link library for the default values. During the `update_timing` or `characterize` commands, Design Compiler searches the first library on the link path for the default values. If Design Compiler cannot find the default values, it uses the following values:

```
wire_load_model:       (none)
wire_load_mode:        TOP
operating_conditions:  (system defaults)
```

Use the following commands to indicate a library name:

```
set_wire_load_model -name model_name -library library_name

set_operating_conditions -library library_name

set_timing_ranges -library library_name
```

The *`library_name`* must either be a fully defined file name or exist in memory. If the requested information (wire load, operating conditions, or timing range) is not in *`library_name`*, Design Compiler searches the libraries in the link path. When Design

Compiler finds the information, it reports the library name. Design Compiler never searches the link library unless you define it as *library_name*.

For more information about link libraries, see the *Design Compiler User Guide*.

Specifying Operating Conditions

The `set_operating_conditions` command specifies the operating conditions or environmental characteristics under which you want Design Compiler to optimize your design. You can specify either minimum, maximum, or both options for one optimization run.

The syntax is

```
set_operating_conditions
              [-min  min_operating_condition ]
              [-max  max_operating_condition ]
              [-min_library  min_library_name ]
              [-max_library  max_library_name ]
              [-library  library_name ]
              [operating_condition ]
```

Note:

For more information, see the `set_operating_conditions` man page.

Creating Your Own Operating Conditions

You can create your own operating conditions without a Library Compiler license, if none are available that satisfy your design requirements. You must use Library Compiler syntax. After you create the operating conditions file, you must compile it to make the operating conditions available to Design Compiler.

Example

This example shows how to create operating conditions and store them in the library extra.lib.

```
extra.lib
library ("extra") {
operating_conditions ("SLOW") {
    process : 1.75 ;
    temperature : 100;
    voltage : 4.66;
    tree_type : "worst_case_tree";}
}
```

This command sequence compiles the operating conditions and makes them available to Design Compiler.

```
dc_shell> read_lib extra.lib
dc_shell> write_lib extra -output extra.db
dc_shell> link_library = {"*" target_library extra.db}
dc_shell> set_operating_conditions "SLOW" -library "extra"
```

Defining Timing Ranges

Design Compiler optimizes designs according to operating temperature, supply voltage, and manufacturing process variations. To model statistical variations in the nominal operating conditions, use timing ranges, which affect timing constraints.

A timing range consists of two multipliers used to calculate the range of delays for the best-case and worst-case timing conditions. These multipliers, or scaling factors, are applied to the nominal path delays when Design Compiler times the design.

Design Compiler uses these scaling factors to modify the calculations of the following timing constraints:

- Maximum path delay (setup) multiplied by the slowest (largest) factor of all the specified ranges
- Minimum path delay (hold) multiplied by the fastest (smallest) factor of all the specified ranges

The timing range is defined by the `timing_range` specification in the technology library.

Example

```
timing_range("SLOW_RANGE") {
    faster_factor : 1.05
    slower_factor : 1.2
}
timing_range("FAST_RANGE") {
    faster_factor : 0.90
    slower_factor : 0.96
}
```

In this example,

- The `SLOW_RANGE` scaling factors for timing calculation are 1.05 for the best case and 1.2 for the worst case.
- The `FAST_RANGE` scaling factors for timing calculation are 0.96 for the worst case and 0.90 for the best case.

Timing ranges affect timing reports and delay-cost computation for the `compile` and `report_constraint` commands. Use the `report_constraint` and `report_timing` commands to get information about the range and factor used to scale a path delay.

This modeling produces accurate results in cases where all delays have a linear dependency on the operating conditions. In cases where delays do not scale linearly with operating conditions, this method provides only a first-order approximation.

For more information about maximum and minimum path delay computation, see Chapter 2, “Constraining Designs.”

Setting Timing Ranges

The `set_timing_ranges` command defines one or two timing ranges for the current design. If no timing range is set, no timing range is used.

The syntax is

```
set_timing_ranges {range1  [ , range2 ]}
```

range1 and the optional *range2* are names of timing ranges defined in the technology library. If you specify *range2*, separate the values with a comma.

Example

To set the timing ranges for the current design to `SLOW_RANGE` and `FAST_RANGE`, enter

```
dc_shell> set_timing_ranges {SLOW_RANGE, FAST_RANGE}
```

Listing Library-Defined Timing Ranges

Use the `report_lib` command to list the timing ranges defined in a library.

Example

```
dc_shell> report_lib my_lib
****************************************
Report : library
Library: my_lib
Version: v3.4
Date   : Thu 1995
****************************************
. . .
Timing Ranges:
Name         Library    Slower Factor    Faster Factor
-----------------------------------------------------
FAST_RANGE   my_lib        0.9600           0.9000
SLOW_RANGE   my_lib        1.2000           1.0500
```

Modeling Wire Loads

Design Compiler uses wire load models to estimate the capacitance, resistance, and area of nets before floorplanning or layout. In the absence of physical design information, Design Compiler uses statistically generated wire load models to estimate wire lengths. The wire load models, provided in the technology library, define a fanout-to-length relationship.

Design Compiler can automatically create wire load models. See “Selecting the Wire Load Model Automatically” on page 3-25.

Design Compiler estimates the wire length of pin-to-pin connections, based on fanout count, then uses that estimate to calculate the effects of cell placement and wire routing.

- Fanout is the total number of pins on the net, excluding the driver pin. The wire load definition contains one or more fanout length pairs.

- If fanout values are not consecutive integers, Design Compiler uses linear interpolation to calculate the intermediate `fanout_length` pairs.
- If Design Compiler encounters a fanout count greater than the largest `fanout_length` pair, it uses the slope value to extrapolate the wire length.

The wire load is defined by the `wire_load` specification in the technology library.

Design Compiler determines which wire load model to use for a design in one of the following ways:

- According to the wire load model you define with the `set_wire_load_model` command.
- According to the wire load indicated by the technology library's `wire_load_selection` group (in the absence of an explicitly set wire load model).
- According to the wire load model identified by the `default_wire_load` attribute in the library (in the absence of both preceding factors).

If none of the preceding are defined, no wire load model is used. If no wire load model is used, the timing information from synthesis is optimistic; that is, your nets have no loading and propagation times. The assumed net resistance and capacitance values will be zero and the net delays will also be zero. With no wire load model selected, Design Compiler does not have complete information about the behavior of your target technology.

You can direct Design Compiler to use different wire load models for minimum and maximum timing delay analysis.

Calculating the Capacitance of a Net

To calculate the capacitance of a net,

1. Determine the fanout of the net.
2. Look up the length from the `fanout_length` pairs in the `wire_load` model.
3. Calculate the capacitance by multiplying the length by the capacitance coefficient.

Example

```
wire_load("90x90") {
    capacitance :  2.0 ;   /* C per unit-length */
    resistance  :100.0 ;   /* R per unit-length */
    area        :  0.5 ;   /* net-area per unit-length */
    slope       :  1.5 ;   /* extrapolation slope */
    fanout_length(1,1) ;   /* fanout_length pairs */
    fanout_length(2,2.2);
    fanout_length(3,3.3);
    fanout_length(4,4.4);
}
```

To determine lumped net capacitance, total net resistance, and the design area due to the net, Design Compiler multiplies the capacitance, resistance, and area scaling factors by the estimated wire length.

Design Compiler calculates the net fanout value as the total number of pins on the net excluding one driver pin. For example, on a net with 2 fanin pins and 3 fanout pins, the fanout value is

```
(2 + 3) - 1 = 4
```

The lumped net capacitance is calculated by multiplication of the estimated wire length by the capacitance factor 2.0. For a fanout value of 4, the last fanout_length defines the wire length as 4.4, resulting in a calculated lumped net capacitance of

```
2.0 * 4.4 = 8.8
```

The total net resistance is calculated by multiplication of the estimated wire length by the resistance factor, yielding a value of

```
4.4 * 100.0 = 440.0
```

When calculating pin-to-pin RC delays, Design Compiler interprets the total net resistance on the basis of the current operating condition's tree_type attribute.

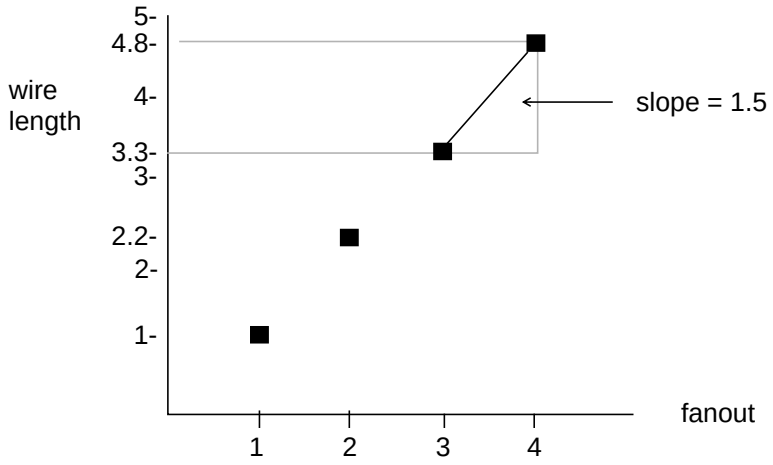
The net area is calculated by multiplication of the estimated wire length by the area factor, yielding a value of

```
4.4 * .5 = 2.2
```

Design Compiler uses the net area to measure the impact of different optimization possibilities on the total design area of the chip. An accurate net area estimate guides Design Compiler in selecting the best optimizations for minimizing area.

Figure 3-22 shows how Design Compiler calculates wire loads.

Figure 3-22 Wire Load Calculation



Using the previous wire load model for a net with a fanout value of 8 and a slope value of 1.5, Design Compiler calculates the wire length and capacitance as follows:

```
length = 4.4 + ((8 - 4) * 1.5) = 10.4
capacitance = 10.4 * 2.0 = 20.8
```

Note:

Define a positive slope value. A positive slope ensures that Design Compiler takes into account that adding fanouts to a net increases its length (and therefore its delay and area).

Choosing a Wire Load Model

Which wire load model you choose depends on how the design is ultimately implemented. Hierarchy restrictions imposed on layout results in more-accurate wire load estimates.

Wire load models are typically provided for different sizes of designs based on the observation that same-size blocks generally exhibit similar fanout-to-length relationships. A wire load model must completely contain the design block under consideration.

- If a design is not limited to one contiguous region of the physical chip, the wire load model must reflect that its nets can span the whole chip. In this case, the size of the block that completely contains the design is identical to the size of the chip.
- If a design is limited to a contiguous region of the physical chip, use a wire load model created for designs of the same size. Nets fully enclosed within the design are typically shorter and more predictable than those that potentially span the whole chip.

Another criterion for selecting a wire load model is the degree of pessimism used in its creation. When the potential exists for significantly underestimating the length of a net, use a wire load model that employs pessimism. By using a more pessimistic wire load model in the appropriate situations, you are less likely to have design violations after layout. Consult your Synopsys library supplier for details about how the wire load models provided with the library are created.

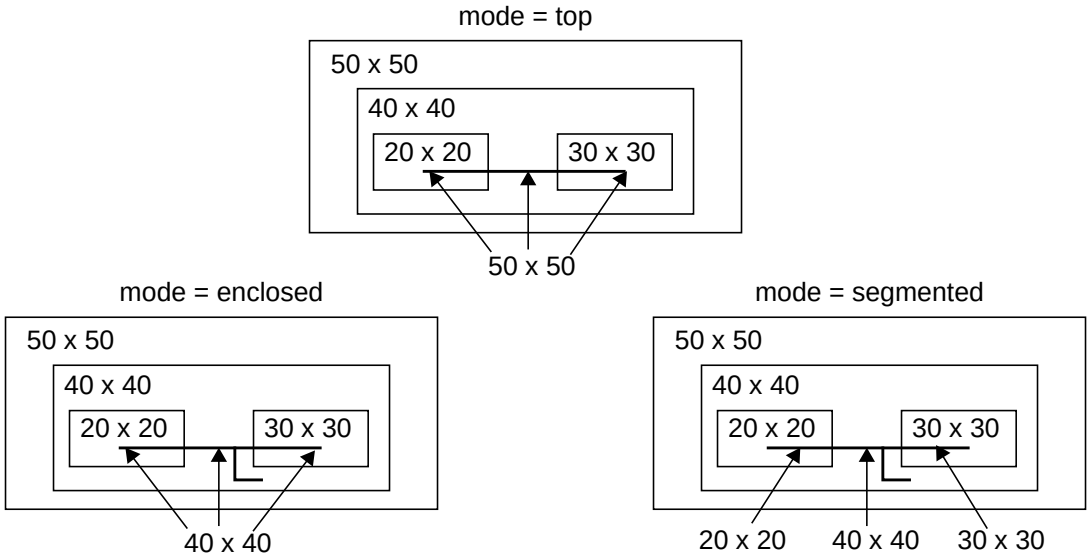
- For large blocks, use wire load models that are more pessimistic in their fanout-to-length estimates than those for small blocks, because the net lengths in large blocks vary more than in small blocks.
- For large fanouts, use fanout-to-length estimates that are more pessimistic than for small fanouts, because the net lengths for high fanout nets are harder to predict accurately.

Wire Load Model Modes

The wire load model mode defines the wire load model to use for nets in subdesigns. Technology libraries have a `default_wire_load_mode` attribute that defines the default mode. You can override a mode by using the `set_wire_load_mode` command.

The cell area used to select a wire load model for a net is determined by the wire load mode. Figure 3-23 shows the wire load model modes.

Figure 3-23 Wire Load Model Modes



top

Causes the wire load model and cell area for the top level to be used. Wire load models set on subdesigns and subclusters have no effect. The top mode models nets as if the design has no hierarchy; it is pessimistic.

enclosed

Causes the wire load model and the cell area of the smallest design that fully encloses the net to be used. If the design enclosing the net has no wire load model, the design hierarchy is traversed upward until a wire load model is found. The enclosed mode is more accurate than the top mode when cells in the same design are placed in a contiguous region during layout.

segmented

Causes the sum of the cell areas of the designs containing the net to be used. Nets crossing hierarchical boundaries are divided into segments (part of the net is outside the module). Each net segment is estimated from the wire load model of the design containing the segment. If the design enclosing a segment has no wire load model, the design hierarchy is traversed upward until a wire load model is found.

If you want to set the lower blocks in a design to a mode other than top mode, you must set the TOP design to enclosed or segmented mode.

Examples

To set the model my_model on a design, enter

```
set_wire_load_model my_model
```

If the current library does not define a default mode, Design Compiler searches in the libraries in the `link_library`.

To set the mode for the current design to top so that all nets use the same model (wire load models set on subdesigns are ignored), enter

```
set_wire_load_mode top
```

In a hierarchical design, which is the current design, to cause each net to use the model of the lowest-level block that completely contains that net, enter

```
set_wire_load_mode enclosed
```

In a hierarchical design, which is the current design, to cause nets to be divided into segments (as the nets cross block boundaries) and to estimate each net segment from the wire load model of the block containing the segment, enter

```
set_wire_load_mode segmented
```

Note:

The default value for the `set_wire_load_mode` command is top. The `wire_load_mode` attribute cannot be removed. It can only be changed by using the `set_wire_load_mode` command.

Setting a Wire Load Model

The `set_wire_load_model` command can be used to define a wire load model for designs, hierarchical cells, and ports. The wire load model you define with `set_wire_load_model` overrides defaults.

You can use the `set_wire_load_model` command to define an external wire load model for a port. The wire load model of a port can affect the calculated wire load of the net connected to the port (depending on the wire load model for the design).

In the presence of physical hierarchy, wire loads set on subdesigns are ignored. In general, wire loads associated with clusters obey the same mode rules that govern wire loads in the presence of logical hierarchy.

If you set a wire load model on a design, automatic wire load selection is disabled for that design.

The set_wire_load_model command does not assume a default wire load mode. You use the set_wire_load_mode command to explicitly set the wire load mode before you use the set_wire_load_model command. (For additional information about this command, see the man page.)

The syntax is

```
set_wire_load_model    -name  model_name
                       [-cluster  cluster_name ]
                       [-library  library_name ]
                       [-min] [-max] [ object_list ]
```

Note:

For more information, see the set_wire_load_model man page.

Examples

```
dc_shell> set_wire_load_model -name "60x60"
Using wire_load model '60x60' found in library 'my_lib'.
```

To remove the wire_load_model attribute, use the remove_wire_load_model command. Additional information, refer to the man pages.

Setting a Wire Load Mode

The set_wire_load_mode command sets the wire load mode for the current design. For this command, Design Compiler performs additional checking to ensure that the minimum block size is not set with wire load modes other than enclosed.

The syntax is

```
set_wire_load_mode  mode_name
```

Note:

For more information, see the `set_wire_load_mode` man page.

Example

```
dc_shell> set_wire_load_mode top
```

Setting a Wire Load Minimum Block Size

The `set_wire_load_min_block_size` command sets the minimum block size for a wire. The minimum block size is supported only with the enclosed wire load mode. For this command, Design Compiler performs additional checking to ensure that minimum block size is not set with other wire load modes.

The syntax is

```
set_wire_load_min_block_size size
```

size

Minimum block size. Specify a value that is greater than or equal to zero (>=0).

To remove the `wire_load_min_block_size` attribute, use the `remove_wire_load_min_block_size` command.
Additional information, refer to the man pages.

Setting a Wire Load Selection Group

The `set_wire_load_selection_group` command sets the wire load selection group used to determine the wire load model when `auto_wire_load_selection` is set to true.

The `set_wire_load_selection_group` command does not assume a default wire load mode. You can use the `set_wire_load_mode` command to explicitly set the wire load mode to enclosed. For detailed information, see the man page for the command.

The syntax is

```
set_wire_load_selection_group
                  [-min] [-max]
                  [-library  lib_name ]
                  [-cluster  cluster_name ]  group_name
```

Note:

For more information, see the `set_wire_load_selection_group` man page.

To remove the `wire_load_selection_group` attribute, use the `remove_wire_load_selection_group` command. For additional information, refer to the man pages.

Defining Wire Load Models or Selection Groups for Hierarchical Cells

To define distinct wire load models for hierarchical cells in the top-level design, define wire load models at the designs and at the instance level. When defining your wire load models remember the following:

- Instance level wire load settings take precedence over design level wire load settings
- Design Compiler selects a wire load model based on the total cell area of the design if you designate top as the wire load mode for a design that has no wire load model defined (for the top most level) and if the first library in the link path has a wire load selection table

You can use the `set_wire_load_min_block_size` command to set a minimum block size for the design.

Selecting the Wire Load Model Automatically

If no wire load model is defined for a design, Design Compiler automatically selects a default wire load model based on area. The cell area used to select a wire load model for a net is determined by the wire load mode. Design Compiler searches the first library on the link path (or the first library in the design's local_link_library) for a default wire load.

- If the library contains a `wire_load_selection` group definition, Design Compiler uses the cell area of the `current_design` to select the wire load model.

- If the library does not contain a `wire_load_selection` group definition, Design Compiler uses the `default_wire_load` for the library.
- If the library contains neither a `wire_load_selection` group nor a `default_wire_load`, Design Compiler does not select a wire load model and no wire load model is used.

If you set a wire load model on a design, automatic wire load selection is disabled for that design.

Note:

On large designs, using automatic wire load selection based on area is not recommended, because excessive runtimes might result.

To disable automatic wire load model selection, set the variable

```
auto_wire_load_selection = false
```

Defining the Smallest Block Size

Use the `set_wire_load_min_block_size` command to define the smallest block size to be laid out contiguously in a design. Use the `set_wire_load_min_block_size` command when the logical hierarchy is decomposed into subblocks that have no physical meaning (and should not be considered in selection of a wire load model). Design Compiler selects a wire load for each design on the basis of a cell area that is no less than the value defined with the `set_wire_load_min_block_size` command.

Example

```
wire_load("05x05") {
    resistance: 0;
    capacitance: 1;
    area: 0;
    slope: 0.186;
    fanout_length(1,0.39);
wire_load("10x10") {
    resistance: 0;
    capacitance: 1;
    area: 0;
    slope: 0.311;
    fanout_length(1,0.53);
default_wire_load: "15x15";
wire_load_selection() {
    wire_load_from_area (0,100,"05x05");
    wire_load_from_area (150,200,"10x10");
}
```

A design with an area larger than the range defined in the `wire_load_from_area` definition is assigned the wire load of the next area range. For example, a design with area 120 lies between the defined intervals 0–100 and 150–200, so the design is assigned the higher of the two wire loads, 10x10.

A design must be mapped to determine the total area. For an unmapped design, if the `default_wire_load` attribute is not defined in the library, no wire load is used during mapping.

The wire load is automatically selected before and during a timing operation and design modification (such as with `update_timing`, `report_timing`, `compile`, `translate`, and `reoptimize_design`).

Design Compiler selects a wire load model before `compile` or `translate`. If the design is not mapped, Design Compiler uses either the default or no wire load model.

Design Compiler resets the wire load model at the following phases:

- After structuring or mapping and before gate-level optimization
- Before buffering
- At the end of optimization

The last update of the wire load model can create a new violation. In this case, you must rerun `compile -incremental_mapping`.

When the wire load model changes, an informational message is issued in the compile log. For example,

```
Information: Changed wire load model for 'U1' from '(none)' to '50x50'. (OPT-170)
```

Reporting Wire Load Models

You can get information about wire load models for designs, clusters, ports, or libraries, using the commands listed in Table 3-4.

Table 3-4 Commands for Reporting Wire Load Information

To report this	Use this command
The wire load models associated with either a design or a cluster	`report_wire_load`
The wire load model and mode for the current design and the libraries linked to the design	`report_design`
The wire load associated with a cluster	`report_clusters`
The wire load model of the output ports of a design	`report_port`
The wire load models defined in a specified library and wire load selection tables defined in the library	`report_lib`

Example

To display the selected wire load model in the current design, enter

```
dc_shell> report_design
****************************************
Report : design
Design : TOP
Version: v3.4
Date   : Thu 1995
****************************************

Library(s) Used:
    class (File: library/class.db)
Local Link Library:
    {library/class.db}
Flip-Flop Types:
    No flip-flop types specified.
Latch Types:
    No latch types specified.

Operating Conditions:

Name       Library       Process     Temp    Volt   Interconnect Model
-------------------------------------------------------------------
WCCOM     class            1.50    70.00    4.75     worst_case_tree

Wire Loading Model:
        Selected automatically from the total cell area.

Name   Library      Res     Cap       Area   Slope  Fanout   Length
      ----------------------------------------------------------
10x10   class     0.0000  1.0000  0.0000  0.3110        1    0.5300

Wire Loading Model Mode: top.
```

Depending on the `wire_load` commands that are set, you see various messages. For example,

```
Selected from the default (area not fully known).
```

appears when design area is not known.

```
Selected manually by the user.
```

appears when you use the `set_wire_load_model` command to set the value.

```
Selected from the default.
```

appears when you do not define a wire load model, using `set_wire_load_model`, and a default is defined in the library.

```
Inherited from parent design via characterize.
```

appears when the wire load for the design is assigned by `characterize` and the mode is top.

```
Inherited during characterize, only used if no wire-load
default exists.
```

appears when the wire load for the design is assigned by `characterize` and the library does not define a default (from area or simple default).

Example

To display the wire load models defined in the library my_lib, enter

```
dc_shell> report_lib my_lib
****************************************
Report : library
Library: my_lib
Version: v3.4
Date   : Fri 1995
****************************************
. . .
Wire Loading Model:
Name   Library   Res      Cap    Area   Slope  Fanout Length
-----------------------------------------------------------
05x05  my_lib   100.000  1.000  0.000  0.186  1     0.390
10x10  my_lib   100.000  1.000  0.000  0.311  1     0.530
20x20  my_lib   100.000  1.000  0.000  0.547  1     0.860
30x30  my_lib   100.000  1.000  0.000  0.782  1     1.400
40x40  my_lib   100.000  1.000  0.000  1.007  1     1.900
50x50  my_lib   100.000  1.000  0.000  1.218  1     1.800
60x60  my_lib   100.000  1.000  0.000  1.391  1     1.700
70x70  my_lib   100.000  1.000  0.000  1.517  1     1.800
80x80  my_lib   100.000  1.000  0.000  1.590  1     1.800
90x90  my_lib   100.000  1.000  0.000  1.640  1     1.900
Wire Loading Model Mode: enclosed.
. . .
Wire loading model selection:
    Selection    Name
min area    max area
-----------------------------------------------------------
      0    100    "05x05"
    100    200    "10x10"
```

4

Specifying Clocks and Clock Networks

Clocks and clock latencies are necessary to constrain a design. Most delays, especially for synchronous designs, depend on the clock.

You can set attributes that store information according to object types, such as nets, pins, cells, clocks, ports, and entire designs. You can also specify internal design timing, logical and electrical connections of ports, and subdesign interfaces.

This chapter includes the following sections:

- Specifying Transition Times of Clock Networks
- Performing Clock-Gating Signal Timing Checks
- Creating Internally Generated Clocks
- Clock-Related Commands Summary

Design Compiler considers clock network delays to be ideal; by default, the timing analyzer does not consider the delay through the gates leading to the clock. The timing analyzer does not report timing delays to the clock pin of a sequential element. You can override this behavior and obtain nonzero clock network delay by using the `set_propagated_clock` or `set_clock_latency` command.

Timing for sequential paths is considered relative to clock edges. Design Compiler automatically finds the relevant setup and hold relations by expanding the clock waveforms. If you want nonsingle-cycle behavior, use the `set_multicycle_path` command.

Design Compiler reports timing delays from clock to clock: from synchronous logic to synchronous logic or the logic between synchronous cells.

Clock Types

Design Compiler uses real clocks and virtual clocks. It does not automatically create clocks when it times the design. You must explicitly create clocks for the clock sources in your design. (See “Creating Clocks” on page 4-5.)

Real Clocks

Real clocks have sources and can be ideal or propagated. You create a real clock by using a `create_clock` command and including a source list of ports or pins.

Ideal Clocks

An ideal clock incurs no delay through the clock network. The ideal clock type is the default for Design Compiler. An ideal clock can represent expected clock network delay and uncertainty. It is the opposite of a propagated clock.

Propagated Clocks

A propagated clock is the opposite of an ideal clock. Registers clocked by a propagated clock have edge times skewed by the path delay from the clock source to the register clock pin. You define a propagated clock by using the `set_propagated_clock` command.

Virtual Clocks

A virtual clock has no sources. It exists in memory but is not part of a design. Use a virtual clock as a reference for specifying input and output delays relative to a clock. In some cases, creating a clock that

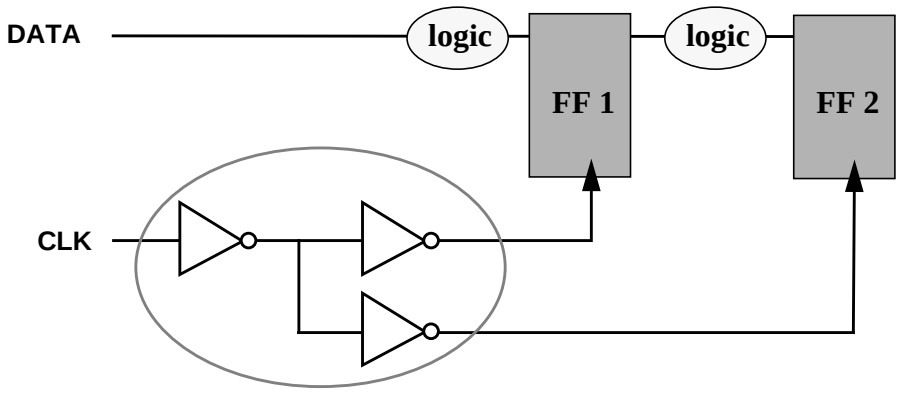
exists in the system but not within the current design is useful. Create a virtual clock by using the `create_clock` command with no source list.

Clock Networks

A clock network (or tree) is the combinational logic between a clock source and the registers in the transitive fanout of that source. A clock network is inverting if an odd number of inversions exist between a clock source and the destination clock pin.

Figure 4-24 shows a simple clock network consisting of inverters leading to two registers.

Figure 4-24 Simple Clock Network



Clock trees usually are synthesized by the vendor, based on the physical placement data. Design Compiler does not effectively synthesize clock trees.

Modeling Clock Trees

Usually, the vendor is responsible for clock tree synthesis and agrees to a guaranteed maximum uncertainty (skew) between the branches in the clock tree. During synthesis, you model the effects of clock skew, using the `set_clock_uncertainty` command, which you apply to a clock object.

By default, clock skew is considered ideal and set equal to zero.

If you have a manually instantiated clock tree, you must use the `dont_touch` attribute on the clock network to ensure that the entire clock network inherits a `dont_touch` attribute.

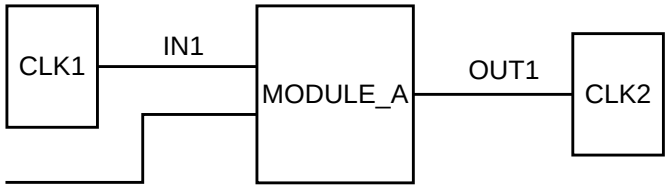
Creating Clocks

Design Compiler can handle single phase, multiphase, and multifrequency clocking schemes. The tool does not automatically create clocks when it times the design. You must explicitly create clocks for the clock sources in your design, using the `create_clock` command.

- A combinational module is not fed by a clock. You must create virtual clocks in the design before setting input or output delays.
- Sequential circuits always have clocks; to constrain these designs, use input and output delays relative to a clock.

Figure 4-25 shows that input delay (IN1) is defined relative to a source clock (CLK1) and output delay (OUT1) is defined relative to a destination clock (CLK2).

Figure 4-25 Input and Output Delay Clock Relationships



These terms describe clocks:

clock source

The pins or ports where the clock waveform is applied to the design. A clock can have multiple sources. Note that the internal pins of an extracted timing model or complex cell can serve as source pins. The clock signal reaches the registers in the transitive fanout of all its sources. You can identify clock sources by using the `derive_clocks` command.

clock period (cycle time)

The smallest time at which the clock waveform repeats. For a simple waveform with one rising and one falling edge, the period is the difference between successive rising edges. The clock period constrains the register-to-register timing of the current design.

waveform

The description of clock edges for one period. A simple waveform has two edges — for example, {0.0 5.0}. Complex waveforms are allowed and can be useful when (certain) clock pulses are blanked out. An example of a complex waveform is {3 5 11 13}.

Edges in the waveform are assumed to be rise and fall.

rising edge

The transition from logic 0 to logic 1.

falling edge

The transition from logic 1 to logic 0.

pulse

Two successive edges of the clock waveform.

In addition to creating clocks, use the `create_clock` command to

- Set the clock period
- Change existing clock information such as period, waveform, and sources
- Constrain clocks with synchronous paths

The syntax is

```
create_clock      [-period  period_value ]
                  [-name  clock_name ]
                  [-waveform  edge_list ] [ source_list ]
```

Note:

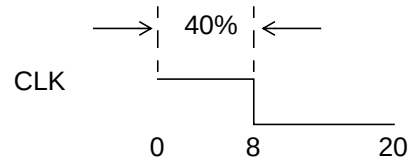
For more information, see the `create_clock` man page.

When you create a clock on an output pin, Design Compiler sets the `dont_touch` attribute on the cell driving the clock source to ensure that the cell is not removed or exchanged for another cell during optimization.

Example 1

To create a clock with a 40 percent duty cycle, enter

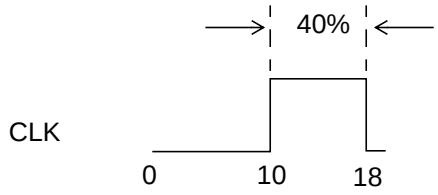
```
dc_shell> create_clock -period 20 -waveform {0 8} CLK
```



Example 2

To create a non-overlapping clock with a 40 percent duty cycle, enter

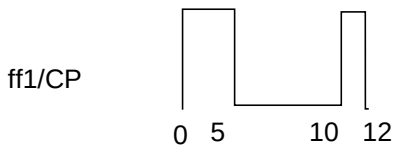
```
dc_shell> create_clock -period 20 waveform {10 18}
```



Example 3

To create a complex clock for pulse blanking, enter

```
dc_shell> create_clock -period 20 -waveform {0 5 10 12} {ff1/CP ff2/CP}
```

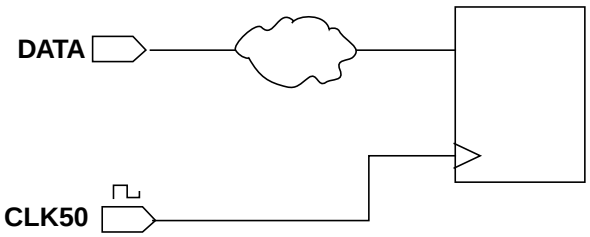


No active edge is assumed. The created clock signal can, in general, drive both positive and negative edge-triggered logic. Use the `-clock_fall` option with the `set_input_delay` and `set_output_delay` commands to specify delays relative to the falling clock edge.

Example 4

Design Compiler does not automatically infer clock signals. Even though input port CLK50 drives a flip-flop clock pin, no clock waveform is assumed. Use `create_clock` to specify a 20-ns waveform for the CLK50 port. The small square-wave icon above the port appears in Design Analyzer. For example, enter

```
dc_shell> create_clock -period 20 CLK50
```



Removing Clocks

Use the `remove_clock` command to remove clocks. You can remove all clocks or define a list of clock names to remove.

The syntax is

```
remove_clock  clock_list  | -all
```

Note:

For more information, see the `remove_clock` man page.

Example

```
dc_shell> remove_clock {CLK ff1/CP}
```

In this example a specific clock is deleted. In the next example all clocks are deleted.

```
dc_shell> remove_clock -all
```

Using Internal Pins as Clock Sources

The internal pins of extracted timing models and complex cells can serve as clock source pins. You can obtain a list or collection of these pins by applying the `find pin` *`object_list`* or the `get_pins -of_objects` *`object_list`* command. Note that you must first set the global variable `access_internal_pins` to true to enable pin access (the default is false).

The following clock-related commands support extracted model internal pins as source pins:

- `create_clock`
- `report_clock`
- `create_generated_clock`
- `set_clock_latency`
- `remove_clock_latency`
- `set_clock_uncertainty`
- `remove_clock_uncertainty`
- `set_propagated_clock`
- `remove_propagated_clock`

These commands are discussed in the following sections of this chapter. You can find additional information in the man pages.

The `report_timing` command supports internal pins, and the `read`, `read_file`, and `write` commands preserve these pins. The `write_script` command outputs any constraints placed on internal pins.

Creating Clock Objects for Network Source Pins or Ports

The `derive_clocks` command creates clock objects for network source pins or ports in the current design.

Use `derive_clocks` to identify the sources of clock networks in the design. The `derive_clocks` command finds clock sources by following the clock network back to input ports or register outputs. New clocks are not assigned periods. To derive the minimum clock periods for clocks, use the `derive_timing_constraints` command.

The syntax is

```
derive_clocks
```

Example

This example shows the design counter before and after clocks are derived. The `report_clock` command, in this example, displays the clock status. Once the `derive_clocks` command is used, the clock status changes.

```
dc_shell> report_clock
****************************************
Report : clocks
```

```
Design : counter
Version: v3.1
Date   : Wed  Jan 08 1997
****************************************

No clocks in this design.

dc_shell> derive_clocks
create_clock find(port,"CLK")

dc_shell> report_clock
****************************************
Report : clocks
Design : counter
Version: v3.1
Date   : Wed Jan 08 12:21:14 1997
****************************************

Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
Clock         Period   Waveform         Attrs     Sources
-------------------------------------------------------------
CLK           -        -                          {CLK}
-------------------------------------------------------------
```

Applying create_clock to Internally Derived or Gated Clocks

The following script enables you to apply the `create_clock` command to internally derived or gated clocks. The script allows you to apply the `create_clock` command after the first compile and then find the instance name of the internal flip-flop, multiplexer, or gate driving the output port of the module you select.You can specify the hierarchical path so that finding the instance name and applying the clock is a repeatable script.

The script makes the following assumptions:

- You have an output port associated with the clock pin.

- You have only one flip-flop driving this clock or output port, so that when you filter the pins connected on the net, the script finds only one output driver pin.
- You have specified the hierarchical path down to the level of hier (current design) where the clock pin exists.

Here is the script. Each line is preceded with an explanation delineated by the /* and */ characters.

```
/* Specify hier. path to current design where clock output port is located. */
internal_clock_path = "core_level/second_level/next_level/"

/* Tell current design where clock output port is located. */
current_design lower_module

/* Find clock port. */
all_connected lower_module_port

/* Extract net from this dc_shell_status */
all_connected find (net, dc_shell_status)

/* Find flip-flop output pin instance name and pin */
filter find (pin,dc_shell_status) "@pin_direction == out"

/* Pull out string from dc_shell_status */
foreach (derived_clock_pin, dc_shell_status) {}

/* Go to top level to apply create_clock command. */
current_design top_level

/* Use create_clock at top level with full hier path */
create_clock -period 30 find(pin, internal_clock_path + derived_clock_pin)
set_clock_uncertainty 1 find(pin, internal_clock_path + derived_clock_pin)
```

Specifying Clock Network Timing

Design Compiler treats clock networks as ideal (having no delay) by default. The timing analyzer does not report timing delays to the clock pin of a sequential element. If the design has a gated clock, the timing analyzer does not consider the delay through the gates

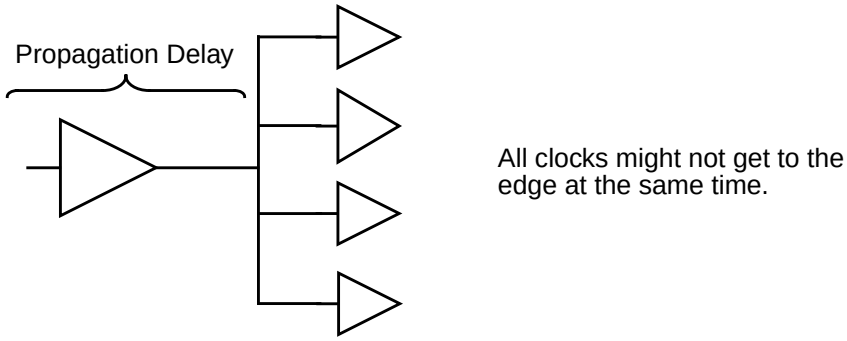
leading to the clock. You can override the default behavior (using the `set_clock_latency` and `set_propagated_clock` commands) to obtain nonzero clock network delay and specify information about the clock network delays, whether the skews are predicted or actual. You can specify this information on clock objects or on flip-flop clock pins.

Timing for sequential paths is considered relative to the clock edges.

The difference in rise or fall edges at two register clock pins is called skew. You can model the potential for skew before layout by using setup uncertainty and hold uncertainty values. After layout, actual skew can be modeled by use of different delays for individual clock pins or by use of the propagated clock attribute in combination with back-annotation of cell and net delays. Clock network delay and uncertainty are specified with the `set_clock_latency` and `set_clock_uncertainty` commands. Cell and net delays are specified with the `set_annotated_delay` command.

Specify rise and fall values in pairs, in increasing value order and less than one period in duration—for example, {rise fall} or {rise fall rise fall} or {rise fall rise fall rise fall} and so on. Figure 4-26 shows clock skew.

Figure 4-26 Clock Skew



How Design Compiler Calculates Clock Skew During Minimum and Maximum Timing

Design Compiler supports simultaneous optimization of minimum as well as maximum delay paths. You can specify the minimum as well as maximum operating conditions to be used during optimization and timing analysis. The `dc_shell compile` and `reoptimize_design` commands support this feature.

You can use the `set_clock_latency` command to approximate the expected delay through the clock network. For example, as part of the pre-layout process, you might specify the following commands:

```
set_clock_latency -max 0.8 clk
set_clock_latency -min 0.3 clk
```

Alternatively, you might use back-annotation to approximate the expected delay, in which case you would back-annotate Standard Delay Format (SDF) values. For example, as part of the pre-layout process, you might specify the following commands:

```
set_propagated_clock clk
read_sdf -max_type max -min_type min design.sdf
```

The `set_propagated_clock` command specifies propagated clock latency for the stipulated clock, list of clocks. ports, pins, and cells. For information about the `read_sdf` command, see the *PrimeTime User Guide: Advanced Timing Analysis*, Chapter 6, “Reading and Writing SDF Delay.” Refer to the man pages for the complete description of the `set_propagated_clock` and `read_sdf` commands.

The clock skew to clock pins of the source and destination registers can be calculated in multiple ways. By default, Design Compiler uses the specifications shown in Table 4-5 to calculate the clock skew with minimum and maximum values.

Table 4-5 indicates that for a setup calculation, Design Compiler calculates the clock skew of the source register, using the longest path to the clock in the worst-case operating condition. Design Compiler calculates the clock skew of the destination register by looking at the shortest path to the clock pin in the worst-case operating condition.

Table 4-5 Default Values Used to Calculate Setup and Hold

Register	Operating condition	Path
setup source	worst case	longest
setup destination	worst case	shortest
hold source	best case	shortest
hold destination	best case	longest

Setting Clock Network Timing

The `set_clock_uncertainty`, `set_clock_latency`, and `set_propagated_clock` commands place propagated clock, rise and fall delay, or uncertainty attributes on specified objects. These attributes propagate to clock pins in their transitive fanout during compilation, when they influence timing calculations on these devices.

The `set_clock_uncertainty` command sets clock skew attributes on clock objects or flip-flop clock pins. This command sets the clock skew values for all flip-flops and latches in the transitive fanout of the specified clocks, ports, or pins.

Use the `set_clock_latency` command to capture actual or predicted delay information about clock networks. The silicon vendor provides the values.

To list clock skew values that are set, use the `report_clock -skew` command.

Example

This example shows how to enable propagated clocks and set uncertainty to 1 for all flip-flops clocked by CLK.

```
dc_shell> set_propagated_clock CLK
dc_shell> set_clock_uncertainty 1 CLK
```

Use the `remove_clock_uncertainty`, `remove_clock_latency`, and `remove_propagated_clock` commands to remove propagated clock, rise and fall delay, or uncertainty attributes from the specified objects. Refer to the man pages for the syntax of these commands.

Setting Clock Latency

Two types of clock latency can be specified for a design: clock network latency is the time it takes for a clock signal on the design to propagate from the clock definition point to a register clock pin. The rise latency is the latency for a rise transition on the register clock pin. The fall latency is the latency for a fall transition on the register clock pin. Clock source latency (also called insertion delay) is the time it takes for a clock signal to propagate from its actual ideal waveform origin point to the clock definition point in the design. It can

be used to model off-chip clock latency when clock generation circuit is not part of the current design. For generated clocks, clock source latency can be used to model the delay from the master-clock to the generated clock definition point.

The syntax for setting clock latency is

```
set_clock_latency      value   [-rise] [-fall]
                       [-min] [-max] [-source]
                       [-early] [-late] [object_list]
```

For more information, see the `set_clock_latency` man page.

Example

```
dc_shell> set_clock_latency 1.5 -rise CLK1
dc_shell> set_clock_latency 2.0 -source CLK2
```

Removing Clock Latency

To undo `set_clock_latency`, use the `remove_clock_latency` command.

The syntax is

```
 remove_clock_latency  [-rise] [-fall]
                       [-min] [-max]
                       [-source] object_list
```

Note:

For more information, see the `remove_clock_latency` man page.

Example

```
dc_shell> remove_clock_latency -rise CLK1
dc_shell> remove_clock_latency -source CLK2
```

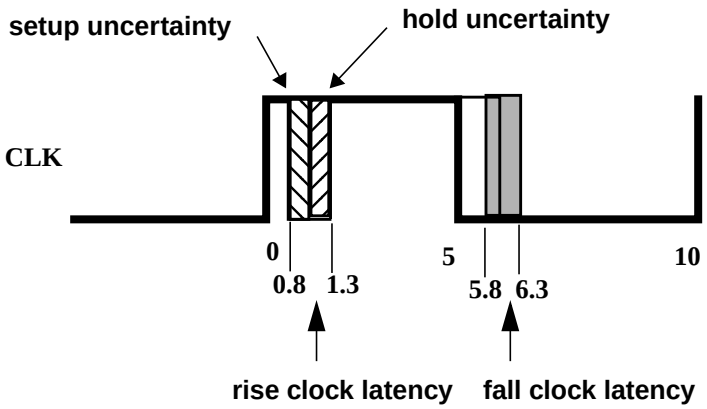
Setting Clock Uncertainty

Setting clock uncertainty is a way to incorporate a margin of error in the design to account for possible variances in the clock propagation times because of layout. Setup uncertainty and hold uncertainty can be the same value or different values. You can define both setup uncertainty and hold uncertainty, creating a window surrounding each clock edge.

- Use setup uncertainty when setup is checked at a register that is a path endpoint. This effectively tightens the maximum delay requirement.
- Use hold uncertainty when hold is checked at a path endpoint.

Figure 4-27 shows an ideal clock with a period of 10 and a waveform of {0 5}. A rise clock latency of 1.0, fall clock latency of 1.0, hold uncertainty of 0.3, and setup uncertainty of 0.2 are defined.

Figure 4-27 Ideal Clock Showing Setup Uncertainty and Hold Uncertainty



Use the `set_clock_uncertainty` command to specify the skew characteristics of one or more clock networks. Specify clock uncertainty as simple uncertainty or interclock uncertainty. Simple

uncertainty means that the setup uncertainty and hold uncertainty applies to all paths to the endpoint. Interclock uncertainty allows you to specify different skew between various clock domains.

The syntax for the `set_clock_uncertainty` command is

```
set_clock_uncertainty  value
                       [-from object_list -to object_list]
                       [-rise] [-fall]
                       [-setup] [-hold] object_list
```

Note:

For more information, see the `set_clock_uncertainty` man page.

Example

```
dc_shell> set_clock_uncertainty 1.5 -setup CLK1
dc_shell > set_clock_uncertainty 2.0 -from CLK1 -to CLK2
```

To undo `set_clock_uncertainty`, use the `remove_clock_uncertainty` command. The syntax is

```
remove_clock_uncertainty
              [-from  object_list  -to object_list ]
              [-rise] [-fall]
              [-setup] [-hold] object_list
```

Note:

For more information, see the `remove_clock_uncertainity` man page.

Example

```
dc_shell> remove_clock_uncertainty -setup CLK1
dc_shell> remove_clock_uncertainty -from CLK1 -to CLK2
```

Setting a Propagated Clock

Set a propagated clock to specify that delays be propagated through the clock network to determine latency at register clock pins. If not specified, ideal clocking is assumed. Ideal clocking means clock networks have a specified latency (from the `set_clock_latency` command) or zero latency by default. Propagated clock latency is used after layout, after final clock tree generation. Ideal clock latency provides an estimate of the clock tree before layout.

The syntax is

```
set_propagated_clock  object_list
```

Note:

For more information, see the `set_propagated_clock` man page.

Example

```
dc_shell> set_propagated_clock CLK1
```

To undo `set_propagated_clock`, use the `remove_propagated_clock` command. The syntax is

```
remove_propagated_clock  object_list
```

Note:

For more information, see the `remove_propagated_clock` man page.

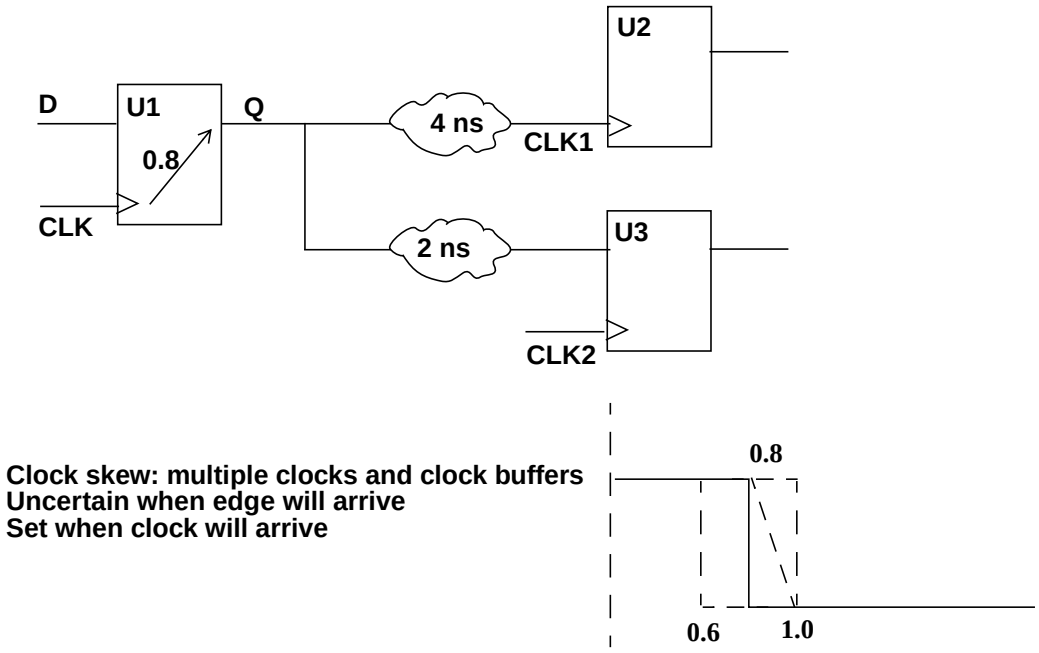
Example

```
dc_shell> remove_propagated_clock CLK1
```

Specifying Timing Goals for a Synchronous Design Example

This example shows the flow and commands for setting a synchronous design. Figure 4-28 on page 4-22 shows the flow.

Figure 4-28 Flow of a Synchronous Design



Specify timing goals by defining the clock and other related timing requirements for the design.

1. Create the clock. Enter

```
create_clock U1/Q -name CLK1 -period 10
```

2. Set the clock skew. Enter

```
set_clock_latency 0.8 CLK1
set_clock_uncertainty 0.2 CLK1
set_clock_uncertainty -from CLK -to CLK2 1
```

To undo values set with `set_clock_latency` and `set_clock_uncertainty`, use `remove_clock_latency` and `remove_clock_uncertainty`.

Clock Timing for Ideal Clocks

For setup checks at a path endpoint, use the `-setup_uncertainty` option.

Clock Edge Computation, Setup, Noninverting Clock Network.

```
rise_edge = ideal_clock_rise_edge + min_rise_delay - setup_uncertainty
fall_edge = ideal_clock_fall_edge + min_fall_delay - setup_uncertainty
```

Clock Edge Computation, Setup, Inverting Clock Network.

```
rise_edge = ideal_clock_fall_edge + min_rise_delay - setup_uncertainty
fall_edge = ideal_clock_rise_edge + min_fall_delay - setup_uncertainty
```

The clock edge computation for hold checks with plus uncertainty is at the path endpoint.

Clock Edge Computation, Hold, Noninverting Clock Network.

```
rise_edge = ideal_clock_rise_edge + max_rise_delay + hold_uncertainty
fall_edge = ideal_clock_fall_edge + max_fall_delay + hold_uncertainty
```

Clock Edge Computation, Hold, Inverting Clock Network.

```
rise_edge = ideal_clock_fall_edge + max_rise_delay + hold_uncertainty
fall_edge = ideal_clock_rise_edge + max_fall_delay + hold_uncertainty
```

Clock Timing for Propagated Clocks

Use the actual delay through the clock network for propagated clocks.

Clock Edges for Setup

```
rise_edge = actual_min_rise_time - setup_uncertainty
fall_edge = actual_min_fall_time - setup_uncertainty
```

Clock Edges for Hold

```
rise_edge = actual_max_rise_time + hold_uncertainty
fall_edge = actual_max_fall_time + hold_uncertainty
```

Note:

Uncertainty does not affect the clock edges at the startpoint.

Listing Clock Information

You can display clock information by using the following commands:

- `all_clocks()`
- `report_clock`
- `get_clocks` (dctcl mode only)
- `report_transitive_fanout`
- `report_transitive_fanin`

Listing All Clocks in the Design

The `all_clocks()` command displays a list of clocks in the design. You can use the output of `all_clocks()` as input for other clock commands.

The syntax is

```
all_clocks()
```

Example

To direct `compile` to fix hold violations for all clocks in the design by adding delay to the paths, enter

```
dc_shell> set_fix_hold all_clocks()
```

Reporting Clock Information

The `report_clock` command displays information about the clocks in a design.

The syntax is

```
report_clock [-attributes] [-skew] [-nosplit]
```

Note:

For more information, see the `report_clock` man page.

Examples

Example 4-1 Clock Attributes Report

```
dc_shell> report_clock  -attributes
****************************************
Report : clock
Design : counter
Version: v1997.01
Date   : Tues Jan 14 1997
****************************************
Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
Clock          Period   Waveform   Attrs     Sources
-------------------------------------------------------------
CLK            10.00   {0 5}                    {CLK}
ffa/CP                 12.00      {2 4 10 12}             {ffa/CP ffb/CP}
off_chip       20.00   {5 15}                   {}
-------------------------------------------------------------
```

Example 4-2 Clock Skew Report

```
dc_shell> report_clock -skew

****************************************
Report : clock_skew
Design : counter
Version: v1997.01
```

```
Date   : Tues Jan 14 1997
*************************************
            Rise      Fall     Hold         Setup
Object         Delay        Delay      Uncertainty   Uncertainty
-------------------------------------------------------------------------
CLK          1.00     1.20       0.30      0.20
ffa/CP       1.10     1.10       -         -
ffb/CP       0.80     0.80       -         -
```

Preserving the Clock Network

Design Compiler preserves clock networks and specified nets when you use the `set_dont_touch_network` or `set_auto_disable_drc_nets` command.

Note:

The `set_auto_disable_drc_nets` command has the same functionality as the `set_auto_ideal_nets` command of previous releases and replaces that command.

The `set_dont_touch_network` command preserves the nets attached to one or more clock sources during optimization. You should use this command after clock tree synthesis.

The `set_dont_touch_network` command sets a `dont_touch_network` attribute on the clock objects and propagates the `dont_touch` attribute throughout the hierarchy for the clock network, marking the cells in each path. The `dont_touch` attribute stops at output ports and at sequential components if setup and hold relationships exist.

The `set_auto_disable_drc_nets` command disables design rule fixing (DRC fixing) on clock trees and constant nets. You should use the `set_auto_disable_drc_nets` command when you do

not have an accurate representation of the clock tree and you want Design Compiler to treat the clock net as ideal. This is typically done when a clock tree has not yet been inserted for your design.

The `set_auto_disable_drc_nets` automatically sets the `auto_disable_drc_net` attribute on clock nets and nets driven by constants within the current design. The `auto_disable_drc_net` attribute disables DRC fixing, that is, the `max_capacitance`, `max_fanout`, and `max_transistion` constraints are ignored for these nets.

Preserving a Clock Network After Clock Tree Synthesis

The `set_dont_touch_network` command does not apply if the network has unmapped logic.

To list the clock networks in the design, use the `report_transitive_fanout -clock_tree` command.

The syntax for `set_dont_touch_network` is

```
set_dont_touch_network  clock_list
```

Note:

For more information, see the `set_dont_touch_network` man page.

Example

The following example shows how you can you use the `set_dont_touch_network` command.

```
dc_shell> set_dont_touch_network CLOCK
Performing set_dont_touch_network on clock 'CLOCK'.
```

Preserving a Network Before Clock Tree Synthesis

The `set_auto_disable_drc_nets` command should be used before your clock tree has been created. If a net has the `auto_disable_drc_net` attribute, redundant buffers or inverters of the net might be removed during optimization because only the net is preserved.

Disabling DRC fixing on high fanout nets such as clocks speeds up optimization. In addition, it allows Design Compiler to focus on only the violating paths in the design.

The syntax for `set_auto_disable_drc_nets` is

```
set_auto_disable_drc_nets  [-default] [-none] [-all]
                           [-clock true | false]
                           [-constant true | false]
                           [-scan true| false ]
```

For additional information on the `set_auto_disable_drc_nets` command, refer to the man pages.

Correcting Hold Violations

Design Compiler fixes hold time requirements only when directed by the `set_fix_hold` command.

The `set_fix_hold` command directs Design Compiler to insert delay to correct hold (minimum path) violations for timing paths that end at registers fed by this clock. The `set_fix_hold` command sets the `fix_hold` attribute. Use `set_fix_hold` after synthesis when setup requirements are met.

Hold time problems generally occur in shift register structures or scan chains. Because Design Compiler treats the clock as ideal with no path delays, you need to account for the network by issuing the `set_propagated_clock` command.

The syntax is

```
set_fix_hold  clock_list
```

Note:

For more information, see the `set_fix_hold` man page.

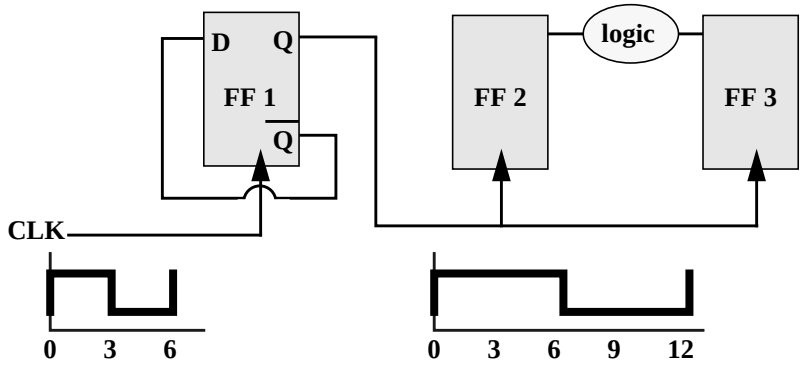
Example

```
dc_shell> set_fix_hold CLOCK
```

Defining Divided Clocks

A clock divider circuit generates a new clock signal with a lower frequency than the original clock. Describe a clock divider by using the `create_clock` command to create a new clock at the output of the divider circuit. Figure 4-29 shows a clocked circuit with a divided clock.

Figure 4-29 Clocked Circuit With Divided Clock



Design Compiler does not automatically determine that FF1/Q is a divided clock source. You must create both the clock and the divided clock. For example, enter

```
dc_shell> create_clock -period 6 -waveform {0 3} CLK
dc_shell> create_clock -period 12 -waveform {0 6} FF1/Q
```

Specifying Transition Times of Clock Networks

The `set_clock_transition` command specifies the transition time of a clock network. Use `set_clock_transition` on a clock object to override the calculated transition time for that clock network. The specified value is then used for computing and indexing into delay tables. To see existing clock transition values, use `report_clock -skew`.

Use `set_clock_transition` to specify the transition time of the clock network when

- The calculated value from the library is inaccurate
- You need to perform timing analysis under different conditions

The `set_clock_transition` command is especially useful when the pre-layout netlist might not have an accurate representation of the clock tree and when calculated transition times at register clock pins might be highly pessimistic. It places the `clock_rise_transition` and `clock_fall_transition` attributes on specified ideal clocks.

Using fixed times for register clock transition reduces overall runtime, because of time savings during the sequential mapping phase of compilation.

Use `set_clock_transition` with ideal clocks. For propagated skews on register clock pins, the values are ignored.

The syntax is

```
set_clock_transition  transition_value  [-rise | -fall]
     [-min] [-max]  clock_list
```

Note:

For more information, see the `set_clock_transition` man page.

To undo a `set_clock_transition` command, use `remove_clock_transition`, `remove_attribute`, or `reset_design`.

Examples

To set both the rise and the fall transition times to 0.75 on the clock pins of all sequential elements clocked by CLK, enter

```
dc_shell> set_clock_transition 0.75 CLK
```

To set only the fall transition time to 0.64 on the clock pins of all sequential elements clocked by CLK, enter

```
dc_shell> set_clock_transition 0.64 -fall CLK
```

To set both the rise and the fall transition times to 0.0 on the clock pins of all sequential elements clocked by some clock, enter

```
dc_shell> set_clock_transition 0.0 all_clocks()
```

Removing Clock Transition Attributes

The `remove_clock_transition` command removes the clock transition attributes on specified clocks, resetting the original calculated clock transition. To see existing clock transition values, use `report_clock -skew`.

The syntax is

```
remove_clock_transition  clock_list
```

Note:

For more information, see the `remove_clock_transition` man page.

Examples

To remove transition attributes on clock CLK, enter

```
dc_shell> remove_clock_transition CLK
```

To remove transition attributes on all the clocks in the current design, enter

```
dc_shell> remove_clock_transition all_clocks()
```

Performing Clock-Gating Signal Timing Checks

A gated clock signal occurs when a clock network contains logic other than inverters or buffers. For example, if a clock signal acts as one input to a logical AND function and a control signal acts as the other input, the output is a gated clock signal.

Design Compiler does *not* automatically check setup and hold violations on the gating signals of clock-gated cells. Without appropriate constraints, it is possible for these signals to undergo transitions while clock pulses are passing through the gating cells. This can lead to both clipped and spurious clock pulses.

Using the `set_clock_gating_check` command, you can specify setup and hold margins to control gating signal transitions. The setup check (`-setup` option) ensures that the clock-gating input is stable for a given time interval before the clock input of the gating cell changes to a noncontrolling value. Similarly, the hold check (`-hold` option) ensures that the clock-gating signal remains stable for a given time interval after the clock input returns to a controlling value. Used together, the two checks ensure that the clock-gating signal is stable for the entire period of time during which the gated clock input has a noncontrolling value.

If you want only the rising or only the falling delays constrained by the `set_clock_gating_check` command, use the `-rise` or the `-fall` option, respectively. If you do not specify either of these options, both types of delays are constrained.

You can list the design objects on which clock-gating setup and hold margins are to be checked. Design objects include designs, cells, pins, and clock objects. Note that multiplexer cells are supported. Also, when you specify a clock object, all clock-gating cells in the given clock's network are checked. If you do not specify any design objects, the clock-gating check is applied to the current design only.

In the case of multiplexers or certain other cells that Design Compiler cannot analyze, you must designate the noncontrolling value of the clock. Use the `-high` (`-low`) option to define the noncontrolling clock value as high (low). You use these options only with cells or pins.

Design Compiler handles clock-gating checks like constraints and tries to adjust the delays of the logic driving the gating inputs to avoid setup and hold violations. During optimization, Design Compiler can change only the size of cells with clock-gating checks. The logic functions of such cells are preserved, that is, no other logic transformations are allowed.

Note:

Clock-gating checks cannot be performed between two clock signals or between two nonclock signals.

The command syntax is

```
set_clock_gating_check      [-setup setup_margin]
                            [-hold hold_margin]
                            [-rise][-fall] [-high|-low]
                            [object_list]
```

If you decide that clock-gating checks are no longer needed, you can remove them with the `remove_clock_gating_check` command. This command has the same syntax as the `set_clock_gating_check` command.

Note:

For more information, see the `set_clock_gating_check` and the `remove_clock_gating_check` man pages.

In Figure 4-30 on page 4-35, the setup and hold margins define AND and NAND clock-gating cells when the `set_clock_gating_check` command is used. The setup margin is measured relative to the rising transition of the gating cell's clock input; the hold margin is measured from the falling transition of the clock input.

Figure 4-30 Setup and Hold Margins for AND and NAND Gates

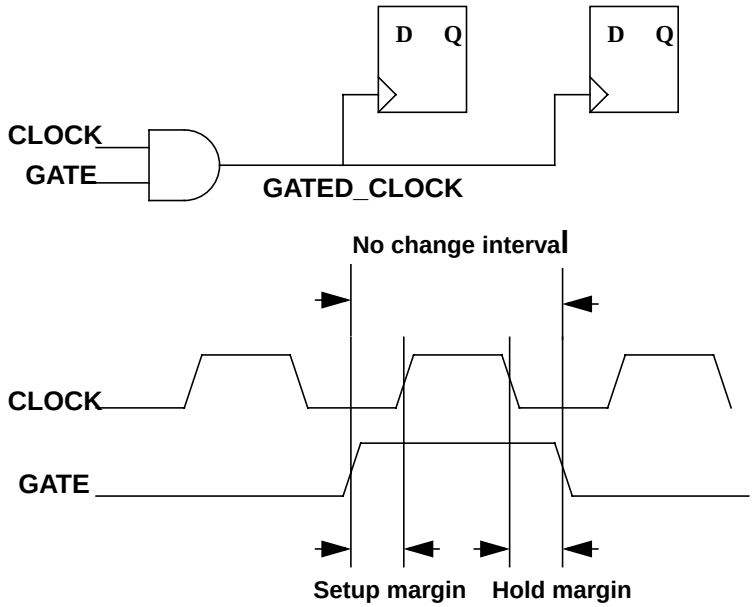


Figure 4-31 on page 4-36 shows setup and hold margins for OR and NOR clock-gating cells. The setup margin is measured relative to the falling transition of the gating cell clock input. The hold margin is measured relative to the rising transition of the clock input.

Figure 4-31 Setup and Hold Margins for OR and NOR Gates

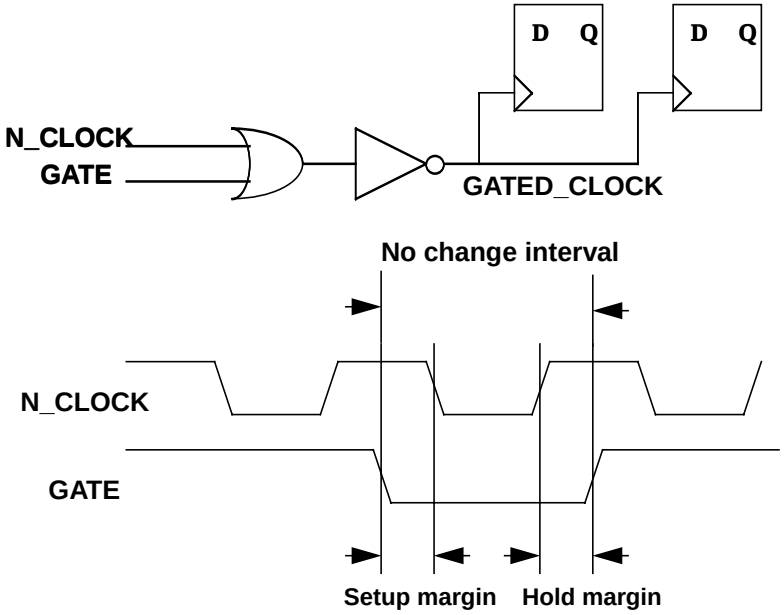
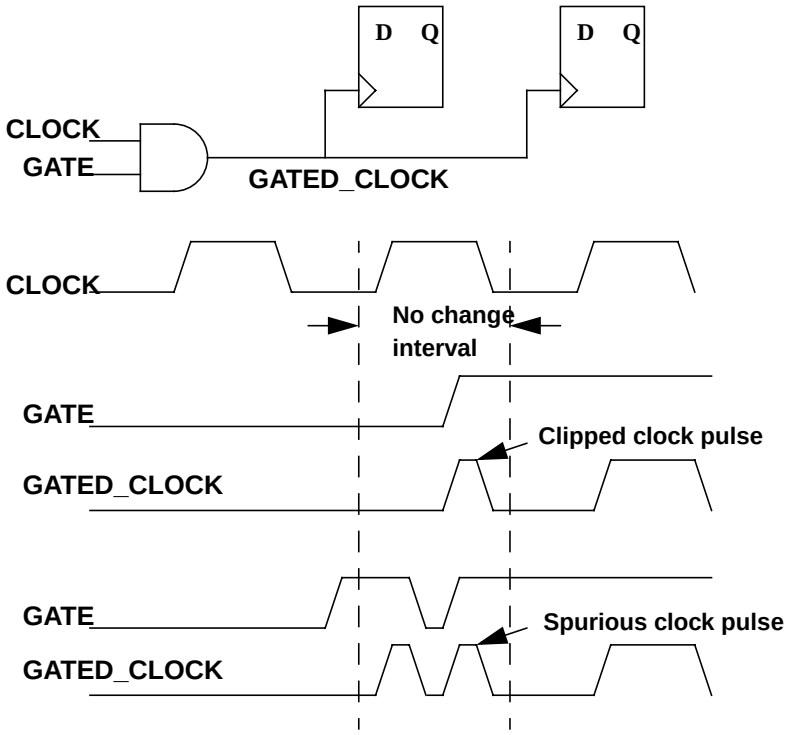


Figure 4-32 on page 4-37 shows examples of distorted clock waveforms that might be caused by invalid changes on the clock-gating inputs if `set_clock_gating_check` is *not* used.

Figure 4-32 Distorted Clock Waveforms



Example 1

To specify a setup requirement of 0.2 and a hold requirement of 0.4 on all gates in the clock network of CLK1, enter

```
dc_shell> set_clock_gating_check -setup 0.2 \
? -hold 0.4 CLK1
```

Example 2

To specify a setup requirement of 0.5 on gate and1, enter

```
dc_shell> set_clock_gating_check -setup 0.5 and1
```

Disabling and Enabling Clock-Gating Checks

You can disable clock-gating checks on specific cells and pins. Use the `set_disable_clock_gating_check` command to list the cells and pins you want the clock-gating check to ignore. To reenable the clock-gating check on disabled cells or pins, use the `remove_disable_clock_gating_check` command.

Note:

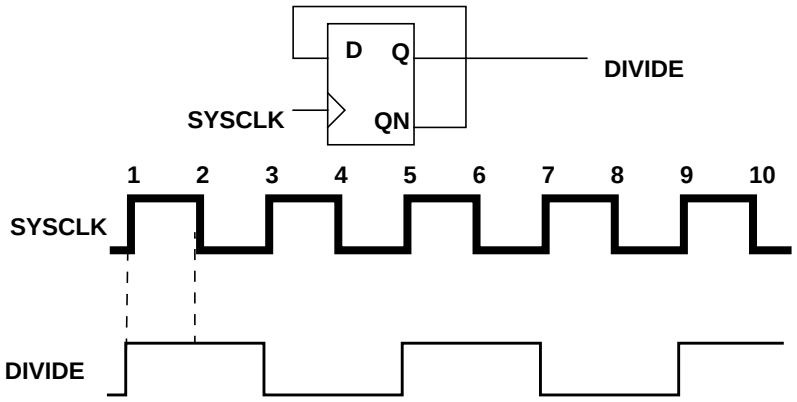
For more information, see the `set_disable_clock_gating_check` and `remove_disable_clock_gating_check` man pages.

Creating Internally Generated Clocks

A design might include clock dividers or other structures that produce a new clock from a source clock. Design Compiler can now represent these structures with a generated clock object.

Figure 4-33 shows the waveforms of the master and generated clock for a divide-by-2 clock generator (assuming that the generated clock is ideal and that there is no clock-to-Q delay).

Figure 4-33 Divide-by-2 Clock Generator



With the `create_generated_clock` command, you can define the relationship between the master clock and the generated clock so that you do not need to derive the waveform manually if the master clock changes.

The `create_generated_clock` command creates an internally generated clock. When Design Compiler times your design, it expands generated clocks and generates new clocks. Used on an existing generated clock object, `create_generated_clock` overwrites the attributes of that clock. Generated clock objects are expanded to real clocks at the time of analysis.

You cannot use the `create_generated_clock` command relative to a virtual clock because the source is not known and the source delay cannot be calculated.

The syntax is

```
create_generated_clock
                  [-name clock_name]
                  -source pin_or_port
                  [-divide_by freq_div_factor |
                  -multiply_by freq_mult_factor]
                  [-duty_cycle duty_cycle] [-invert]
```

```
-edges {edge1, edge2, edge3}
[-edge_shift {shift1, shift2, shift3}]
object_list
```

Note that an internal pin of an extracted timing model or complex cell can serve as a source pin for a generated clock. For more information, see the `create_generated_clock` man page.

Creating a Divide-by-2 Generated Clock

When the frequency division factor for a divided clock is a power of 2, use the `-divide_by` option of the `create_generated_clock` command and specify the frequency division factor.

To create the divide-by-2 generated clock in Figure 4-33 on page 4-39, specify 2 as the frequency division factor. For example, enter

```
dc_shell> create_generated_clock -name CLK1 \
? -source CLK -divide_by 2 INFF1/Q
```

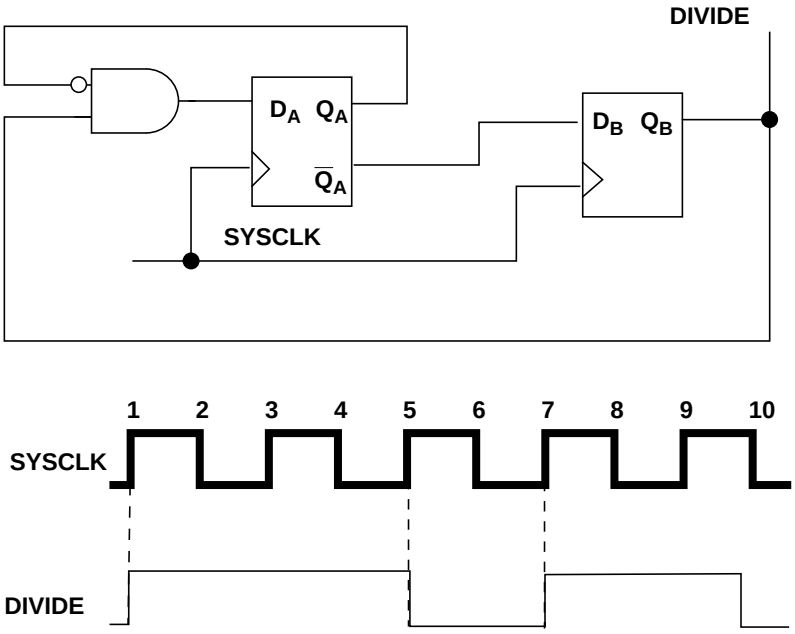
Creating a Divide-by-3 Generated Clock

When the frequency division factor for a divided clock is not a power of 2, use the `-edges` option of the `create_generated_clock` command and specify the edges.

To create a divide-by-3 generated clock, specify the edges in terms of the edge number from the master clock that form the edges of the generated clock. For example, to generate the clock shown in Figure 4-34, enter

```
dc_shell> create_generated_clock -edges { 1 5 7 } \
? -source SYSCLK [get_pins DIVIDE]
```

Figure 4-34 Divide-by-3 Clock Generator

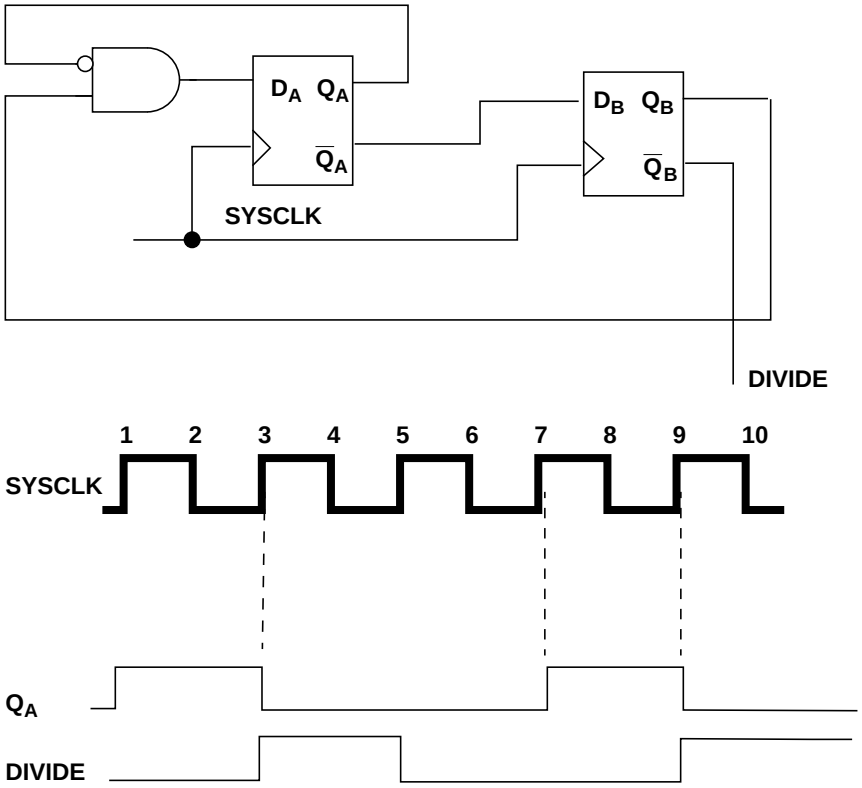


Shifting the Edges of a Generated Clock

You can shift the edges of a generated clock. This shift is not considered to be clock latency. The number of shifts must equal the number of edges. To create a divide-by-3 generated clock and shift the edges by two time units as shown in Figure 4-35, enter

```
dc_shell> create_generated_clock -edges { 1 3 7 } \
? -edge_shift { 2 2 2 } -source SYSCLK [get_pins DIVIDE]
```

Figure 4-35 Divide-by-3 Generated Clock With Shifted Edges



Q_A is divided by 3.
DIVIDE is Q_A shifted by 2.

Selecting Generated Clock Objects

The `get_generated_clocks` command selects a generated clock object.

The syntax is

```
get_generated_clocks [-filter expression] pattern
```

Note:

This command can be used only in dctcl mode. For more information, see the `get_generated_clocks` man page.

Example

The `get_generated_clocks` command returns a token that represents a collection of generated clocks whose names match the pattern and whose attributes pass a filter expression:

```
dc_shell> set_false_path -from [get_generated_clocks CLK_DIV*] \
? -to [get_clocks CLKB]
```

Reporting Clock Information

The `report_clock` command shows information about clocks and generated clocks in the current design. It can also show clock network information.

The `report_transitive_fanout -clock_tree` command shows the clock networks in your design.

Removing Generated Clock Objects

The `remove_generated_clock` command removes generated clocks.

The syntax is

```
remove_generated_clock -all | generated_clock_list
```

Note:

For more information, see the `remove_generated_clock` man page.

Example

To remove the generated clock named CLK_DIV2 (and its corresponding clock object), enter

```
dc_shell> remove_generated_clock CLK_DIV2
```

Clock-Related Commands Summary

Table 4-6 summarizes the clock-related commands.

Table 4-6 Clock-Related Commands Summary

Command	Description
`all_clocks()`	Lists the clocks in the design
`create_clock`	Creates a clock, sets the clock period, changes existing clock information, and constrains clocks with synchronous paths
`create__generated_clock`	Creates generated clock objects from specified source pins.
`derive_clocks`	Creates clock objects for network source pins or ports
`get_clocks`	Displays collection of clocks in the design (dctcl mode only)
`remove_clock`	Removes a clock or clocks
`report_clock`	Displays information about the clocks in a design
`report_transitive_fanin`	Lists fanin logic
`report_transitive_fanout`	Lists fanout logic
`set_clock_gating_check`	Specifies setup time and hold time margins for clock-gating signals

Table 4-6 Clock-Related Commands Summary (Continued)

Command	Description
`set_clock_uncertainty`	Sets expected skew value between two clocks or a clock and all other paths
`set_clock_latency`	Sets an estimated clock latency.
`set_propagated_clock`	Calculates the actual clock latency along the clock network
`set_clock_transition`	Controls the input transition time of register clock pins of specified ideal clock sources
`set_dont_touch_network`	Preserves the nets attached to one or more clock sources during optimization
`set_fix_hold`	Corrects hold violations for timing paths

5

Describing Logic Functions and Signal Interfaces

Design Compiler uses logical information about ports during optimization. For example, if an input port is always logic 1 or if the output port is not used, the surrounding logic function might be simplified.

Because each design or module you time is part of a larger system, describe the interface of each module in the context of the larger system for accurate timing analysis.

You can define the electrical interface environment of your design in terms of input signal drive strength and port loading. Additionally, you can define signal relationships in terms of clocks and input and output delays.

This chapter includes the following sections:

Setting Ports to Improve Optimization Quality

Design Compiler provides commands for setting ports to improve optimization quality. Table 5-7 lists the commands that eliminate redundant ports or inverters.

Table 5-7 Commands Eliminating Redundant Ports or Inverters

Command	Description
`set_equal`	Defines ports as logically equivalent
`set_opposite`	Defines ports as logically opposite
`set_logic_dc`	Specifies one or more ports driven by don't care
`set_logic_one`	Specifies one or more ports tied to logic 1
`set_logic_zero`	Specifies one or more ports tied to logic 0
`set_unconnected`	Lists output ports to be unconnected

The following sections describe these commands in detail.

Defining Ports as Logically Equivalent

Some input ports are driven by logically related signals. For example, the signals driving a pair of input ports might always be the same (logically equal) or always be different (logically opposite).

The `set_equal` command specifies that two input ports are logically equivalent.

The syntax is

```
set_equal  port1 port2
```

Note:

For more information, see the `set_equal` man page.

Example

To specify that ports IN_X and IN_Y are equal, enter

```
dc_shell> set_equal IN_X IN_Y
Performing set_equal on port 'IN_X'.
Performing set_equal on port 'IN_Y'.
```

To remove this attribute, use the `reset_design` command. The `reset_design` command removes from the current design all user-specified objects and attributes, except those defined with the `set_attribute` command.

Defining Logically Opposite Input Ports

The `set_opposite` command specifies that two input ports in the current designs are logically opposite.

Use `set_opposite` to eliminate redundant inverters and to improve the quality of optimization.

The syntax is

```
set_opposite  port1 port2
```

Note:

For more information, see the `set_opposite` man page.

To remove this attribute, use the `reset_design` command, which removes from the current design all user-specified objects and attributes, except those defined with the `set_attribute` command. To remove those defined with `set_attribute`, use the `remove_attribute` command.

Allowing Assignment of Any Signal to an Input

The `set_logic_dc` command specifies an input port driven by don't care.

This information is used to create smaller designs during compile. After optimization, a port connected to don't care usually does not drive anything inside the optimized design.

Use `set_logic_dc` to allow assignment of any signal to that input (including but not limited to 0 and 1 during compilation). The outputs of the design are significant only when the inputs that are not don't care completely determine all the outputs, independent of the don't care inputs.

Use `set_logic_dc` on input ports. To specify output ports as unused, use the `set_unconnected` command.

The syntax is

```
set_logic_dc  port_list
```

Note:

For more information, see the `set_logic_dc` man page.

Example 1

```
dc_shell> set_logic_dc { A B }
```

For a 2:1 multiplexer design with inputs S, A, B, and output Z, the function is computed as

```
Z = S*A + S'*B
```

The command `set_logic_dc` B implies that the value of Z is significant when S = 1 and is a don't care when S = 0. The resulting simplification, done during compilation, gives the reduced logic as a wire and the function is

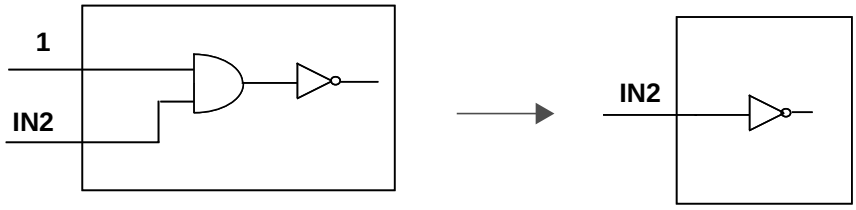
```
Z = A
```

To undo this command, use the `remove_attribute` command.

Specifying Input Ports Always One or Zero

If an input port is always logic-high or -low, Design Compiler might be able to simplify the surrounding logic function during optimization and create a smaller design. Figure 5-36 shows simplified input port logic.

Figure 5-36 Simplified Input Port Logic



You can specify that input ports are connected to logic 1 or logic 0.

Tying Input Ports to Logic 1

The `set_logic_one` command lists the input ports tied to logic 1.

After optimization, a port connected to logic 1 usually does not drive anything inside the optimized design.

Use `set_logic_one` on input ports. To specify output ports as unused, use the `set_unconnected` command.

The syntax is

```
set_logic_one port_list
```

Note:

For more information, see the `set_logic_one` man page.

To undo this command, use the `remove_attribute` command.

Tying Input Ports to Logic 0

The `set_logic_zero` command lists input ports tied to logic 0.

After optimization, a port connected to logic 0 usually does not drive anything inside the optimized design.

Use `set_logic_zero` on input ports. To specify output ports as unused, use the `set_unconnected` command.

The syntax is

```
set_logic_zero  port_list
```

Note:

For more information, see the `set_logic_zero` man page.

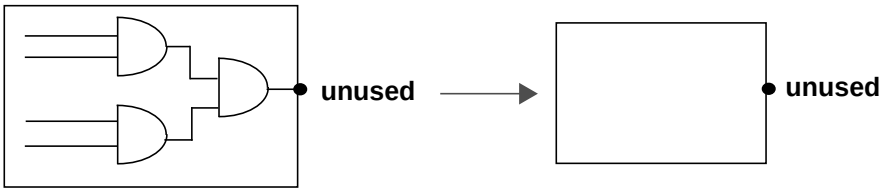
Example

```
dc_shell> set_logic_one IN
Performing set_logic_one on port 'IN'.
```

Specifying Unconnected Output Ports

If an output port is not used (if it is unconnected), the logic driving the port can be minimized or eliminated during optimization. Figure 5-37 shows an example.

Figure 5-37 Minimizing Logic Driving an Unconnected Output Port



The `set_unconnected` command specifies output ports to be unconnected to outside logic.

The syntax is

```
set_unconnected  port_list
```

Note:

For more information, see the `set_unconnected` man page.

Example

```
dc_shell> set_unconnected OUT
Performing set_unconnected on port 'OUT'.
```

To undo this command, use the `remove_attribute` command.

Defining Drive Characteristics for Input Ports

The `set_driving_cell` command associates a library pin with input or inout ports of the current design so that delay calculators can accurately model the drive capability of an external driver.

Use the `set_driving_cell` command instead of the `set_drive` command to describe input drive capabilities. The `set_driving_cell` command works with all delay models. The `set_driving_cell` command removes any corresponding rise or fall drive resistance attributes (from `set_drive`) on the specified ports.

The `set_driving_cell` command takes into account the design rule constraints associated with the specified driving cell in addition to the drive information used to compute the transition time of the input net.

When you specify a driving cell, the design rule constraints are annotated on the input ports of the block being synthesized. The following warning message appears when you run the command:

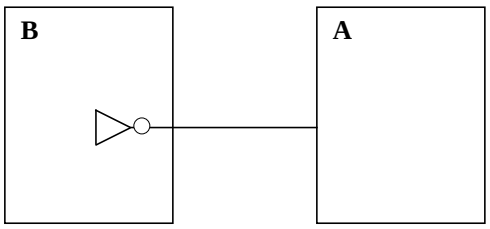
```
Warning: Design rule attributes from the driving cell will be set on the port.
(UID-401)
```

To use the driving cell only to compute input transition times, use the `-no_design_rule` option to `set_driving_cell`. By default, the `characterize` command annotates driving cell information on designs being characterized with the `-no_design_rule` option.

To maintain backward compatibility when you use the `update_script` command to update existing scripts, Design Compiler appends the `-no_design_rule` option to `set_driving_cell` commands.

Example

Setting `set_driving_cell` on A identifies the inverter in B as the driving cell.



- For nonlinear delay models, the drive capability of the port is the same as if the specified driving cell were connected in the same context to allow accurate modeling of the port drive capability.
- For the CMOS2 delay model, the edge rate of pins driven by the port is the same as if the driving cell were substituted for the port. Unless `-dont_scale` is specified, the drive capability of the port is scaled according to the current operating conditions.

To view drive information on ports, use `report_port -drive`.

The `characterize` command automatically sets driving cell attributes on subdesign ports based on their context in the entire design.

Input ports in a subdesign require the same transition delay as the external driving cell in the top design.

The syntax is

```
set_driving_cell  [-lib_cell  lib_cell_name ]
                  [-rise] [-fall] [-library  lib_name ]
                  [-pin  pin_name ]
                  [-from_pin  from_pin_name ]
                  [-multiply_by  factor ][-dont_scale]
                  [-no_design_rule]
                  [-input_transition_rise rtrans]
                  [-input_transition_fall ftrans]
                  port_list
```

Note:

For more information, see the `set_driving_cell` man page.

Attributes Set by set_driving_cell Summary

Table 5-8 summarizes the attributes set by the `set_driving_cell` command.

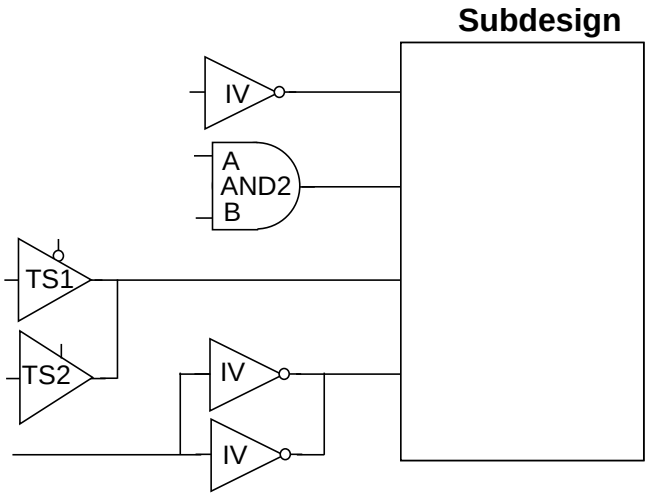
Table 5-8 Attributes Set by set_driving_cell

Attribute name	Attribute type	Value example
`driving_cell_rise` `driving_cell_fall`	string	“AN2”
`driving_cell_library_rise` `driving_cell_library_fall`	string	“tech_lib”
`driving_cell_pin_rise` `driving_cell_pin_fall`	string	“Z”
`driving_cell_from_pin_rise` `driving_cell_from_pin_fall`	string	“B”
`driving_cell_dont_scale`	Boolean	TRUE
`driving_cell_multiply_by`	float	0.5
`driving_cell_max_rise_itrans_rise`	float	1.3
`driving_cell_max_rise_itrans_fall`	float	6.7
`driving_cell_max_fall_itrans_rise`	float	8.343
`driving_cell_max_fall_itrans_fall`	float	63.21

Example

Figure 5-38 shows the circuit on which the example `set_driving_cell` commands are set.

Figure 5-38 Using set_driving_cell



In Figure 5-38, Port I1 is driven by a single IV cell. Because IV has only one output and one input, define the drive as follows:

```
dc_shell> set_driving_cell -lib_cell IV {I1}
```

Port I2 is driven by an AND2 cell. If the different arcs of this cell have different transition times, define the drive as follows:

```
dc_shell> set_driving_cell -lib_cell AND2 -pin Z -from_pin B {I2}
```

Port I3 has two drivers. Assuming that TS1 has the worst rise drive and TS2 has the worst fall drive, define the drive as follows:

```
dc_shell> set_driving_cell -lib_cell TS1 -rise {I3}
dc_shell> set_driving_cell -lib_cell TS2 -fall {I3}
```

Port I4 has two parallel drivers. In some technologies, this configuration is valid and reduces transition times by 50 percent. Use the multiplier option, as follows:

```
dc_shell> set_driving_cell -lib_cell IV -multiply_by 0.5 {I4}
```

Setting Transition Time on Input and Bidirectional Ports

To set the input transition time on a port (the output transition time of the driving cell), use the `set_input_transition` command. You can also use this command to override the transition time calculated using the timing characteristics set by the `set_drive` and `set_driving_cell` commands.

Use the `set_input_transition` command to describe the surrounding environment of a subdesign when you are performing bottom-up compilation of a hierarchical design.

The syntax is

```
set_input_transition   [-rise | -fall] [-min | -max]
                       transition_value port_list
```

Note:

For more information, see the `set_input_transition` man page.

Removing the Drive Strength on Input Ports

The `remove_driving_cell` command removes the drive strength on the specified input ports.

The syntax is

```
remove_driving_cell port_list [-rise | -fall]
```

Note:

For more information, see the `remove_driving_cell` man page.

Specifying Drive Strength for Input and Inout Ports

The `set_drive` command defines the external drive strength (resistance) for input and inout ports.

Note:

If you are describing input drive capabilities, use the `set_driving_cell` command instead of `set_drive`. The `set_driving_cell` command works with all delay models.

The `set_drive` command removes the corresponding rise or fall driving cell attributes on specified ports.

During optimization, the drive of an input port is used to calculate the timing delay to gates driven by that port. For the generic CMOS delay model, the delay to these gates is computed as

```
Time = arrival_time + [drive * load (net)] + connect_delay
```

To view drive information on ports, use the `report_port -drive` command.

The syntax is

```
set_drive          drive_strength  [-rise | -fall]
                   [-min] [-max]  port_list
```

Note:

For more information, see the `set_drive` man page.

Example

This example sets the rise and fall drives of ports A, B, and C to 2.0

```
dc_shell> set_drive 2.0 {A,B,C}
```

Setting Values for All Input Ports

To set values for all input ports, use the `all_inputs()` command. For example, enter

```
dc_shell> set_drive 12.3 all_inputs()
Performing set_drive on port 'IN_0'.
Performing set_drive on port 'IN_1'.
...
Performing set_drive on port 'IN_7'.
```

To remove drive attributes from ports, use either of these commands:

- `remove_attribute`
- `reset_design`

Defining External Loads and Resistance on Ports

External load and resistance describes the current design interface to the environment outside the design.

You can specify loads on input and output ports, and fanout loads for output ports, using these commands:

- `set_load`

  Sets capacitance on nets or ports.

- `set_fanout_load`

  Sets the fanout load value on output ports.

You can specify resistance on input ports, using this command:

- `set_input_parasitics`

Sets resistance value for the external net connected to the input ports.

Defining Input and Output Port Loads

The `set_load` command sets the capacitance on nets and input and output ports.

The syntax is

```
set_load      load_value object_list
              [-min][-max]
              [-subtract_pin_load]
              [-pin_load][-wire_load]
```

Note:

For more information, see the `set_load` man page.

Examples

```
dc_shell> set_load 6.53 IN_PORT
Performing set_load on port 'IN_PORT'.
dc_shell> set_load 5.36 {OUT1, OUT2}
Performing set_load on port 'OUT1'.
Performing set_load on port 'OUT2'.
```

Defining Input Port Resistance

Use the `set_input_parasitics` command to insert a resistive delay between the driving cell of the port and the port's fanout. The resistance value specified is appended in series with the port's fanout net. The delay of the net is then calculated based on the newly formed net.

The net delay is increased on the net connected to the given port. To report the net delays, use the `report_timing -input_pins` command.

The syntax for the `set_input_parasitics` command is:

```
set_input_parasitics   -resistance revalue
                       [-min] [-max] input_ports
```

Note:

For more information, see the `set_input_parasitics` man page.

Defining External Wire Loads on Ports

Use the `set_load` command `-wire_load` option to define the external wire load. Use the `set_load` command `-pin_load` option to define pin loads of input and output ports.

The `-wire_load` and `-pin_load` options are useful when you know the total value of the loads contributed by the external nets. For example, use these options with back-annotated nets or with segmented wire loads where the wire load model is insignificant and you only want the total capacitive load caused by the external net.

If you use both the `-wire_load` and `-pin_load` options, the load value you define is used for both the pin load and wire load of the port. The `-pin_load` option is the default. However, with both the options, you can distinguish between the pin load and wire load portions of the total port load.

With the `set_wire_load_model` command `-object_list` option, you can specify the `wire_load_model` to use for the external net connected to the output port.

Defining the Number of External Fanout Pins

The `set_port_fanout_number` command defines the number of external fanout points.

The syntax is:

```
set_port_fanout_number fanout_number port_list
```

Note:

For more information, see the `set_port_fanout_number` man page.

Defining Fanout Loads on Ports

Design Compiler adds the fanout value to all other loads on the pin driving each port and tries to make the total fanout load less than the pin's maximum fanout load.

The `set_fanout_load` command sets expected fanout load values for output ports. The syntax is

```
set_fanout_load  fanout_load_value port_list
```

Note:

For more information, see the `set_fanout_load` man page.

Example

```
dc_shell> set_fanout_load 6.53 {OUT1, OUT2}
Performing set_fanout_load on port 'OUT1'.
Performing set_fanout_load on port 'OUT2'.
```

For more information about fanout values and constraints, see Chapter 2, "Constraining Designs."

Defining Timing for Ports

Because the current design is usually part of a larger system, you need to model the timing context of the design within that system. You can do this by using the following two commands:

`set_input_delay`

Sets the input delay on pins or input ports relative to a specified clock signal.

`set_output_delay`

Sets the output delay on pins or output ports relative to a specified clock signal.

When using these commands, you can accept the default settings, or you can make the following specifications:

- Identify the rising or falling edge of the specified clock as the reference time for the given delay value.
- Apply the delay to the longest or shortest timing paths.
- Specify a level-sensitive latch instead of a flip-flop as the source of the delay.
- Add delay values at a port or pin when there are multiple paths leading to the port or pin.
- Specify whether the delay value includes the clock source latency or the clock network latency.

Along with clock, drive, and load information, input and output delay information is usually sufficient to capture the timing environment of a design.

Alternatively, for combinational designs or single-phase clocking, you can define the timing as arrival times on inputs and maximum or minimum path delay requirements to outputs.

For paths that originate or go off chip, include in the `set_input_delay` and `set_output_delay` commands an estimate of the delay through the I/O pad. I/O pad delay affects only peripheral data timing paths. Internal logic paths are not affected.

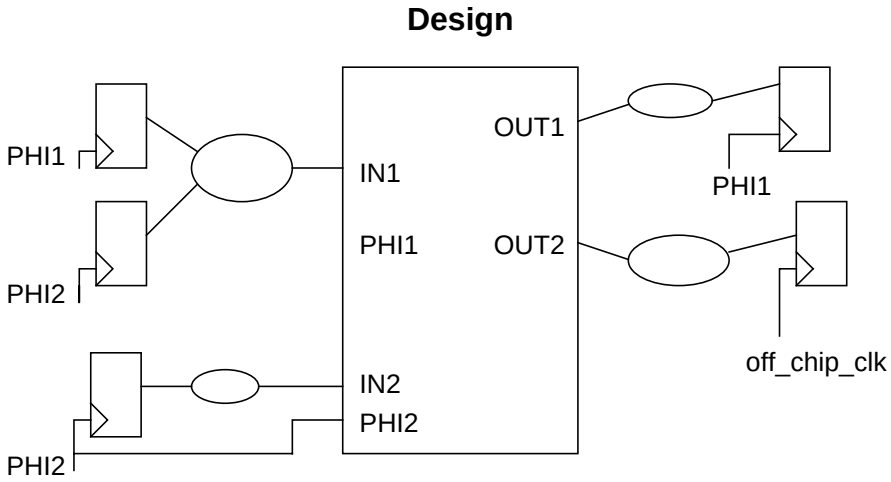
Note:

In most cases, you do not have to specify input or output delay values on internal pins.

Two-Phase Clocking Design Example

Figure 5-39 shows a design that uses two-phase clocking. This design is used to show how you apply the input and output delay commands (see the following two sections).

Figure 5-39 Two-Phase Clocking



The ovals represent combinational timing paths outside the design to be optimized. To properly optimize the design, you need to capture input and output delay information relative to clock edges.

Note:

In designs with multiple clocks or phases, the clock and the edge must be defined with the path delay.

Setting Input Delays

The `set_input_delay` command sets input path delays and input ports relative to a path edge.

Input delays are used to model the external delays arriving at the input ports of the constrained module. These delays are defined relative to a real or virtual clock and are specified to the right of the active clock edge.

When you use `set_input_delay`, Design Compiler tries to check for both setup and hold.

The syntax is

```
set_input_delay        delay_value  [-clock  clock_name ]
                       [-clock_fall] [-level_sensitive]
                       [-max | -min] [-rise | -fall]
                       [-add_delay]
                       [-source_latency_included]
                       [-network_latency_included]
                       port_or_pin_list
```

Note:

For more information, see the `set_input_delay` man page.

Example 1

In the earlier Figure 5-39, consider the path to input IN2. Assume that a rising signal on IN2 can occur 4.3 time units after the rising edge of clock PHI2 and a falling signal has a delay of 3.5 units to reach IN2 (the maximum and minimum times are the same in this case). Input delay is defined as

```
dc_shell> set_input_delay 4.3 -rise -clock PHI2 { IN2 }
dc_shell> set_input_delay 3.5 -fall -clock PHI2 { IN2 }
```

Now consider the delays leading to input port IN1. Paths from registers are clocked by two different clocks. To fully describe this situation, use the `-add_delay` option to capture input delay information relative to both clocks. Enter

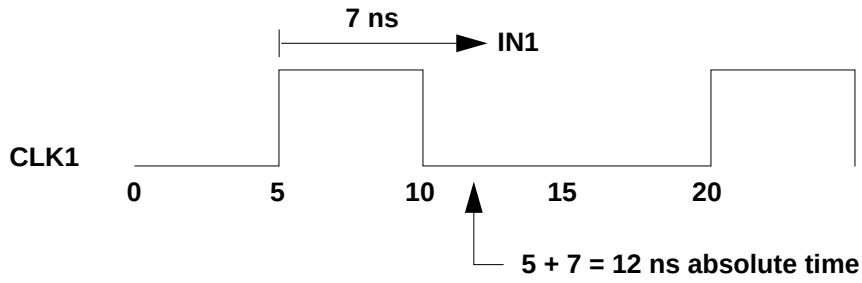
```
dc_shell> set_input_delay 2.7 -clock PHI1 -add_delay { IN1 }
dc_shell> set_input_delay 4.2 -clock PHI2 -add_delay { IN1 }
```

Example 2

This example shows a 7-ns input delay defined relative to clock CLK1. The command for defining the delay is

```
dc_shell> set_input_delay 7 IN1 -clock CLK1
```

Figure 5-40 Input Delay on CLK1

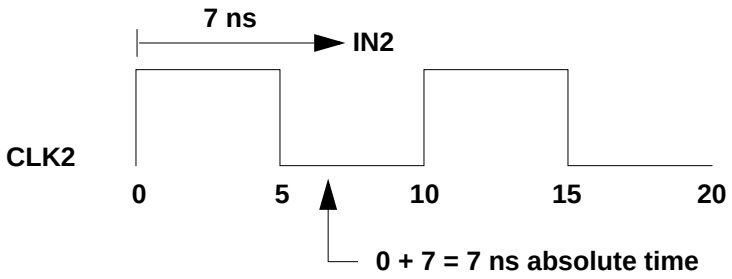


Example 3

This example shows a 7-ns input delay defined relative to clock CLK2. The command for defining the delay is

```
dc_shell> set_input_delay 7 IN2 -clock CLK2
```

Figure 5-41 Input Delay on CLK2



Removing Input Delay and Arrival Times from Ports

To remove input delay and arrival times from ports, use the `remove_input_delay` command. For example, enter

```
dc_shell> remove_input_delay all_inputs()
dc_shell> remove_input_delay -rise {IN1}
```

Setting Output Delays

The `set_output_delay` command specifies path delays and the relative clock and edge for an output port or inout port. Use `set_output_delay` to capture the timing environment of output ports.

Output delays are used to model the external delays leaving the output ports of the constrained module.

Output delays must be defined relative to a real or virtual clock to be considered a path constraint. Output delays are defined to the left of the active clock edge; this delay corresponds to the time before the next rising edge.

Note:

In most cases, you do not have to specify input or output delay values on internal pins.

The syntax is

```
set_output_delay       delay_value  [-clock  clock_name ]
                       [-clock_fall] [-level_sensitive]
                       [-max | -min] [-rise | -fall]
                       [-add_delay]
                       [-source_latency_included]
                       [-network_latency_included]
                       [-group_path  group_name]
                       port_or_pin_list
```

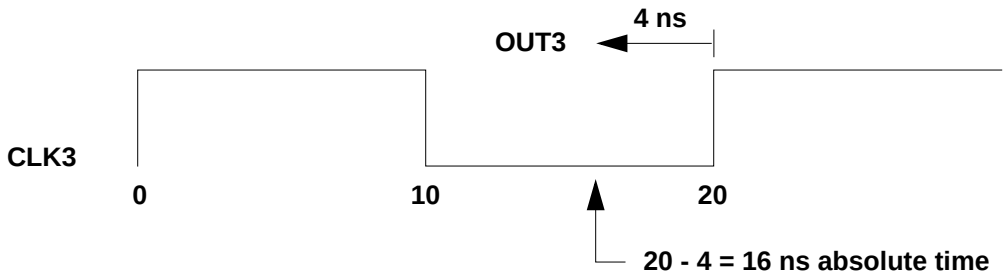
Note:

For more information, see the `set_output_delay` man page.

Example

This example shows a 4-ns output delay defined relative to clock CLK3. The command for defining it is

```
set_output_delay 4 OUT3 -clock CLK3
```



Removing Delay Values

To remove output delay values, use `remove_output_delay` or `reset_design`.

Modifying Path Groups

To modify paths grouped with the `-group_path` option, use the `group_path` command to place the paths in another group or the default group.

Calculating and Setting Maximum Output Delay

If the combinational logic delay from a rising signal on port OUT1 reaches the register in 3.5 time units and there is a 0.3-unit library setup time for that register, the total output delay is 3.8 relative to the clock PHI1 rising edge. The maximum output delay is defined as

```
dc_shell> set_output_delay 3.8 -rise -clock PHI1 {OUT1}
```

Calculating and Setting Minimum Output Delay

If the library hold time is 0.7, the minimum output delay is the minimum path length minus the library hold time (3.5 – 0.7). For example, enter

```
dc_shell> set_output_delay 2.8 -min -rise -clock PHI1 { OUT1 }
```

Creating a Virtual Clock and Setting Output Delay

In some cases, the relative clock for the output delay does not exist within the subdesign. You can use the `create_clock` command to create a virtual clock that can be used in a `set_output_delay` command. For example, enter

```
dc_shell> create_clock -period 10 -waveform {3 8} -name off_chip_clk
dc_shell> set_output_delay 2.5 -clock off_chip_clk {OUT2}
```

Including Instance-Specific Clock Skew in the Output Delay

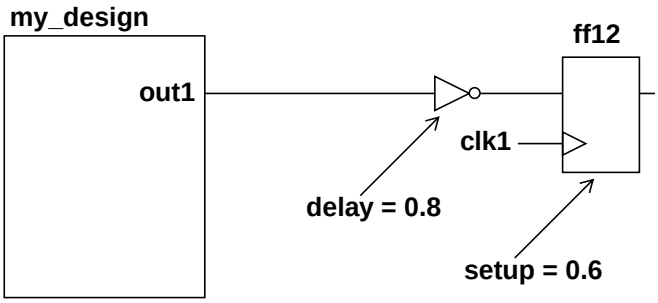
You can include instance-specific clock skew in the output delay calculation. For example, enter

```
dc_shell> create_clock -period 10 -waveform {0 5} clk1
dc_shell> set_clock_latency 0.4 clk1
dc_shell> set_clock_uncertainty 0.1 clk1
```

```
output_delay = path_delay + uncertainty + setup  -clock_delay
     = 0.8 + 0.1 + 0.6 - 0.4
     = 1.1
```

For flip-flop ff12, you implicitly defined the following:

```
dc_shell> set_output_delay 1.1 -clock clk1 {out1}
```



Reporting Port Values

The `report_port` command lists load, drive, external fanout, input delay, output delay, and other information for the ports in the current design or current instance.

The syntax is

```
report_port [-drive] [-verbose] [-nosplit] [ port_list ]
```

Note:

For more information, see the `report_port` man page.

The `report_port` command is instance-specific. If you define `current_instance`, the design of that instance is used in the report, because the attributes on that design do not take effect unless you change `current_design`. In this case, Design Compiler prints a warning.

```
dc_shell> current_instance u1
{"u1"}
dc_shell> report_port
Warning: Port information will be listed for the design of the current instance.
This information is not used by the tool unless current_design is set to 'BOTTOM'.
(UID-363)
****************************************
Report : port
-verbose
Design : DESIGN
Version: v1997.01
Date   : Tues Jan 14 1997
****************************************
                        Pin      Wire     Max     Connection
Port           Dir      Load     Load     Trans   Class        Attrs
--------------------------------------------------------------------------
clk             in      0.0000   0.0000   --      --
in1             in      6.5300   0.0000   --      --
reset           in      0.0000   0.0000   --      --
out1            out     5.3600   0.0000   --      --
                     Input Delay
                   Min             Max      Related  Pin Drive     Wire Drive
Input Port    Rise    Fall    Rise    Fall  Clock   Rise   Fall   Rise   Fall
--------------------------------------------------------------------------------
clk           --      --      --      --      --    0.00   0.00   0.00   0.00
in1           4.30    3.50    4.30    3.50  PHI2   12.30  11.70  12.30  11.70
reset         --      --      --      --      --    0.00   0.00   0.00   0.00
                     Output Delay                          External Fanout
                   Min             Max      Related  Fanout  Number  Wireload
Output Port   Rise    Fall    Rise    Fall  Clock     Load  Points  Model
----------------------------------------------------------------------------
out1          3.80    --      3.80    --    PHI1      6.53      0 --
```

Removing Port Values

Use the `remove_attribute` command to remove load or drive attributes. For example, enter

```
dc_shell> remove_attribute X load
Performing remove_attribute on port 'X'.
{X}
dc_shell> remove_attribute all_inputs() load
Performing remove_attribute on port 'IN_0'.
Performing remove_attribute on port 'IN_1'.
...
```

Describing Internal Timing

You can control internal timing in the following ways:

- Set input delay or output delay on pins.
- Disable timing arcs on cells in a design.
- Back-annotate in Design Compiler cell delay, net delay, resistance, or capacitance calculated by your place and route software.

Breaking Feedback Loops

You can disable timing on parts of a design, such as by breaking a combinational feedback loop.

To report combinational feedback loops, use the `report_timing` command.

To inspect a circuit for timing loops and break them, use the `set_disable_timing` command.

Disabled arcs remain disabled until you remove the `disable_timing` attribute or use the `reset_design` command.

Disabling Timing Arcs

The two types of timing arcs are cell arcs and net arcs.

cell arcs

Describe the cell's internal pin-to-pin timing and are defined in a technology library cell (component) description. Disabling a cell arc is the same as removing a cell description from the technology library.

net arcs

Are implicitly defined by connections between cells. Each driver pin has a net arc to each load pin of a net. Disabling a net arc is the same as breaking the connection between a net driver pin and a load pin.

Use the `set_disable_timing` command to break a timing path at the defined cell or pin arcs.

To mark paths as false (as opposed to breaking a feedback loop), use the `set_false_path` command. For more information, see Chapter 2, "Constraining Designs."

The syntax is

```
set_disable_timing     cell_or_pin_list
                       [-from pin_name  -to  pin_name ]
                       [-restore]
```

Note:

For more information, see the `set_disable_timing` man page.

When you use `set_disable_timing` for pins, the pins and their nets are saved during optimization. This might mean that Design Compiler cannot perform some transformations and optimizations and can lower the quality of results.

Example

```
dc_shell> set_disable_timing CELL31
dc_shell> set_disable_timing {U33, U37/A, U71/Z}
dc_shell> set_disable_timing {U17} -from A2 -to ZN
```

Restoring Disabled Timing Arcs. To restore previously disabled timing arcs, use the `set_disable_timing -restore` command or the `reset_design` command. For example, enter

```
dc_shell> set_disable_timing -restore {U33, U71/Z}
```

Reporting Timing-Disabled Cells and Pins

The `report_design` command lists the cells and pins whose timing arcs are disabled.

```
dc_shell> report_design
****************************************
Report : design
Design : TOP
Version: v1997.01
Date   :Wed Jan 14 1997
****************************************
...
Disabled Timing Arcs:
    Object         Name              From Pin      To Pin
    ------------------------------------------------------
    Cell           U33
    Pin            U71/Z
    Cell           U17                 A2           ZN
```

Breaking Timing Loops

If a timing loop is discovered during compilation, Design Compiler displays the message

```
Information: Breaking a timing loop by disabling timing arcs between pin 'name1'
and pin 'name2'.
```

After optimization, automatically disabled arcs are restored.

Back-Annotating Post-Layout Net Loads

The Design Compiler timing analyzer uses estimates of interconnect network (wire) capacitance and resistance when calculating timing paths and required cell and pin power. Although the wire load is a significant part of the total load, the actual load of each net is not known until after the place and route phase.

For more information, see the *Design Compiler Reference Manual: Optimization and Timing Analysis*.

Selecting a Scan Style

If you are using test-ready compile to insert scan structures in your design, you must select a scan style before you perform logic synthesis. You must use the same scan style for all modules of your design.

Select a scan style based on your design style and on the types of scan cells available in your target technology library. See the *DFT Compiler Scan Synthesis User Guide* for more information on considerations for selecting a scan style.

Specify the scan style by setting the `test_default_scan_style` variable.

```
dc_shell> test_default_scan_style = style
```

The scan style defined by the `test_default_scan_style` variable applies to all the designs in the current session. You can also use the `set_scan_configuration -style` command to specify the scan style. However, this command applies only to the current design. If your selected scan style differs from the default scan style, you must execute this command for each module. See the *Design Compiler Reference Manual: Optimization and Timing Analysis* for details on the `set_scan_configuration` command.

Table 5-9 shows the scan style keywords to use when specifying the scan style. You can use these keywords with either the `test_default_scan_style` variable or the `set_scan_configuration -style` command.

Table 5-9 Scan Style Keywords

Scan style	Keyword
Multiplexed flip-flop	multiplexed_flip_flop
Clocked scan	clocked_scan
Level-sensitive scan design (LSSD)	lssd
Auxiliary-clock LSSD	aux_clock_lssd

Table 5-10 shows the scan cells supported by DFT Compiler. The columns represent circuit clocking in functional mode; the rows represent circuit clocking in scan mode. Use this table to determine scan cell support and the corresponding scan style for your design. For example, if your design has one level-sensitive clock (C) in functional mode and two nonoverlapping clocks (A and B) for scan shift, you need to set the scan style to lssd. If the table does not have an entry, DFT Compiler does not support the configuration.

Note:

Although this is not shown in the table, DFT Compiler also supports multiplexed JK flip-flop scan cells using the multiplexed_flip_flop scan style argument.

Table 5-10 Supported Scan Cells

Functional mode **Scan mode**	**Edge-triggered clock C**	**Level-sensitive clock C**	**Level-sensitive clocks C and B**
Edge-Triggered Clock C	Multiplexed D flip-flop (multiplexed_flip_flop)	Multiplexed D latch (multiplexed_flip_flop)	
Level-Sensitive Clocks C and B			Multiplexed master-slave latch (multiplexed_flip_flop)
Edge-Triggered Clock SC	Clocked-scan flip-flop (clocked_scan)	Clocked-scan D latch (clocked_scan)	
Level-Sensitive Clocks A and B	LSSD D flip-flop (lssd)	LSSD D latch (lssd)	LSSD master-slave latch (lssd)
Level-Sensitive Clocks A and B;Edge-Triggered Clock TC	Auxiliary-clock LSSD (aux_clock_lssd)		

Note: The *scan_style* argument is shown in parentheses.

The following sections describe each of the supported scan cells, including diagrams and truth tables of the default model for the scan cell. DFT Compiler uses this default model if your library does not contain a valid functional model for the cell. See the *Library Compiler User Guide* for information on modeling scan cells in your technology library. These truth tables do not include the invalid conditions (output goes to X) when multiple clocks are active.

Multiplexed Flip-Flop Scan Style

The multiplexed flip-flop scan style uses a multiplexed data input to provide serial shift capability. In functional mode, the scan-enable signal, acting as the multiplexer select line, selects the functional data input. During scan shift, the scan-enable signal selects the scan data input. The scan data input comes from either the scan input port or the scan output pin of the previous cell in the scan chain.

A multiplexed flip-flop scan cell must have the following test pins:

- Scan input
- Scan enable
- Scan output (can be shared with functional output pin)

Multiplexed flip-flop is the scan style most commonly supported in technology libraries. Most libraries provide multiplexed flip-flip equivalents for D flip-flops, JK flip-flops, and master-slave latches. Some technology libraries also provide a multiplexed flip-flop equivalent for D latches.

Flip-Flop Equivalents

Figure 5-42 on page 5-36 shows the logic diagram for the default multiplexed flip-flop scan equivalent of a D flip-flop. Figure 5-42 also shows, in parentheses, the `signal_type` attribute that identifies the test pins in the test_cell group in the technology library.

Table 5-11 on page 5-36 shows the truth table for the cell shown in Figure 5-42.

Figure 5-42 Default Multiplexed D Flip-Flop Scan Cell

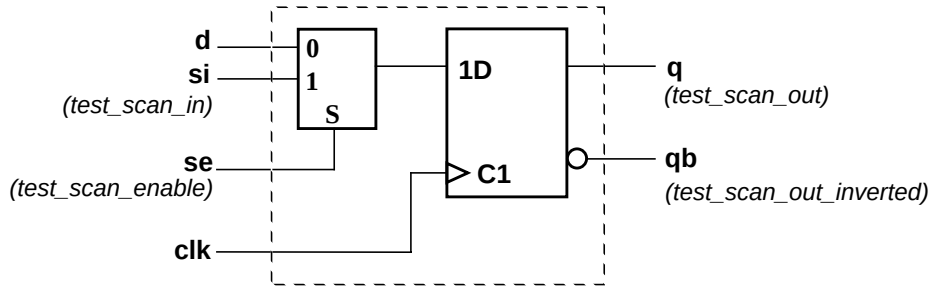


Table 5-11 Truth Table for Default Multiplexed Flip-Flop Scan Cell

Input				Output		Mode
d	**si**	**se**	**clk**	**q**	**qb**	
0	X	0	↑	0	1	Functional
1	X	0	↑	1	0	Functional
X	0	1	↑	0	1	Scan
X	1	1	↑	1	0	Scan
X	X	X	0/1	q	qb	Either

↑= Rising Edge of Clock
X = Don’t Care

Master-Slave Latch Equivalents

Figure 5-43 on page 5-37 shows the logic diagram for the default multiplexed flip-flop scan equivalent of a master-slave latch (technology library `signal_type` attributes for the test pins appear in parentheses). The master clock (mclk) and slave clock (sclk) are non-overlapping, as shown in Figure 5-44 on page 5-37. Table 5-12 on page 5-37 shows the truth table for the cell shown in Figure 5-43.

Figure 5-43 Default Multiplexed Master-Slave Latch Scan Cell

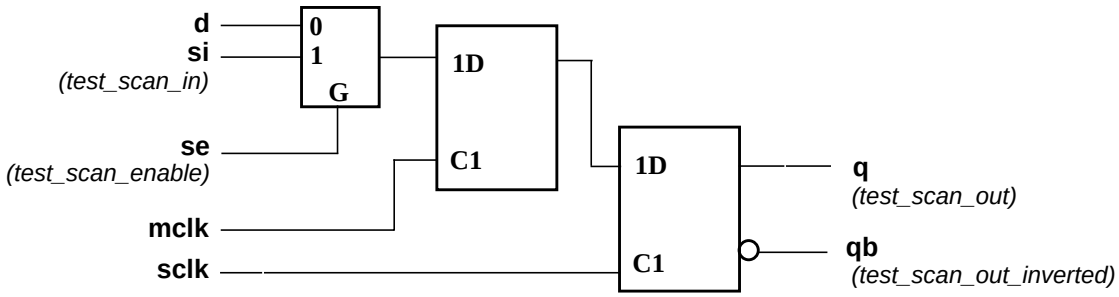


Figure 5-44 Non-overlapping Test Clocks

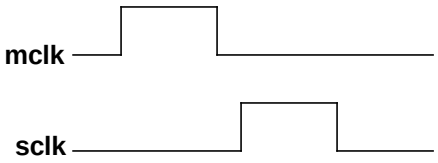


Table 5-12 Truth Table for Default Multiplexed Master-Slave Latch Scan Cell

Input					Output		Mode
d	**si**	**se**	**mclk**	**sclk**	**q**	**qb**	
0	X	0	1	1	0	1	Functional
1	X	0	1	1	1	0	Functional

Table 5-12 Truth Table for Default Multiplexed Master-Slave Latch Scan Cell (Continued)

Input					Output		Mode
d	**si**	**se**	**mclk**	**sclk**	**q**	**qb**	
X	0	1	1	1	0	1	Scan
X	1	1	1	1	1	0	Scan
X	X	X	0	0	q	qb	Either

X = Don’t Care

D Latch Equivalents

Multiplexed flip-flop equivalents for D latches are level-sensitive in functional mode but edge-triggered during scan shift. As with the flip-flop equivalent cell, DFT Compiler supports a single clock input; therefore, the scan-enable input controls whether the cell is operating in functional mode or scan mode.

Figure 5-45 shows the logic diagram for the default model of the multiplexed flip-flop scan equivalent of a D latch (technology library `signal_type` attributes for the test pins appear in parentheses). The default model is negative-edge-triggered during scan shift. Table 5-13 shows the truth table for this model.

Figure 5-45 Default Multiplexed D Latch Scan Cell

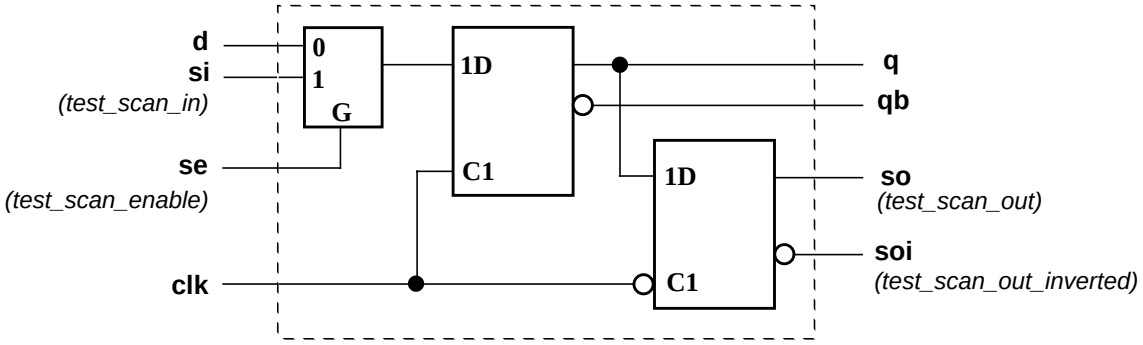


Table 5-13 Truth Table for Default Multiplexed D Latch Scan Cell

Input				Output				Mode
d	**si**	**se**	**clk**	**q**	**qb**	**so**	**soi**	
0	X	0	1	0	1	NA	NA	Functional
1	X	0	1	1	0	NA	NA	Functional
X	0	1	↓	NA	NA	0	1	Scan
X	1	1	↓	NA	NA	1	0	Scan
X	X	X	0	q	qb	q	qb	Either

↓ = Falling Edge of Clock
X = Don’t Care
NA = Not Used in This Mode

Clocked-Scan Scan Style

The clocked-scan style uses a dedicated, edge-triggered test clock to provide serial shift capability. In functional mode, the system clock is active and functional data is clocked into the cell. In scan shift mode, the test clock is active and scan data is clocked into the cell.

A clocked-scan cell must have the following test pins:

- Scan input
- Test clock
- Scan output (can be shared with functional output pin)

DFT Compiler supports clocked-scan equivalents for D flip-flops, JK flip-flops, and D latches.

Flip-Flop Equivalents

Figure 5-46 on page 5-41 shows the logic diagram for the default model of the clocked-scan equivalent of a D flip-flop (technology library `signal_type` attributes for the test pins appear in parentheses). Table 5-14 on page 5-41 shows the truth table for this model.

Figure 5-46 Default Clocked-Scan Flip-Flop Cell

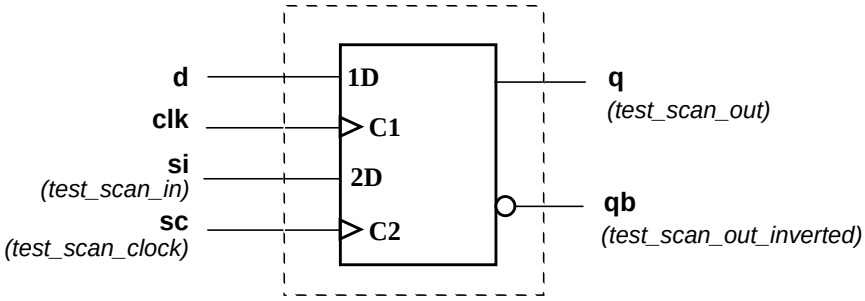


Table 5-14 Truth Table for Default Clocked-Scan Flip-Flop Cell

Input				Output		Mode
d	**si**	**sc**	**clk**	**q**	**qb**	
0	X	0	↑	0	1	Functional
1	X	0	↑	1	0	Functional
X	0	↑	0	0	1	Scan
X	1	↑	0	1	0	Scan
X	X	0/1	0/1	q	qb	Either

↑ = Rising Edge of Clock
X = Don’t Care

D Latch Equivalents

The clocked-scan equivalent for a D latch is level-sensitive in functional mode and edge-sensitive during scan shift. Figure 5-47 shows the logic diagram for the default model of the clocked-scan equivalent of a D latch (technology library `signal_type` attributes for the test pins appear in parentheses). Table 5-15 shows the truth table for this model.

Figure 5-47 Default Clocked-Scan D Latch Cell

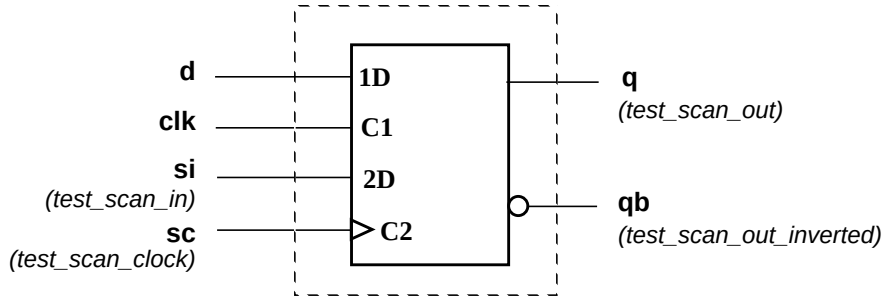


Table 5-15 Truth Table for Default Clocked-Scan D Latch Cell

Input				Output		Mode
d	**si**	**sc**	**clk**	**q**	**qb**	
0	X	0	1	0	1	Functional
1	X	0	1	1	0	Functional
X	0	↑	0	0	1	Scan
X	1	↑	0	1	0	Scan
X	X	0/1	0	q	qb	Either

↑ = Rising Edge of Clock
X = Don’t Care

Level-Sensitive Scan Design Scan Style

The level-sensitive scan design (LSSD) scan style uses a dedicated, level-sensitive test master clock and a level-sensitive slave clock (which might be dedicated, depending on the type of cell) to provide serial shift capability. The classical LSSD scan cell consists of two latches acting as a master-slave pair. The master latch has dual input ports and can latch either functional data or scan data. In functional mode, the system master clock controls the transfer of

data from the data input to the master latch. In scan mode, the test master clock controls the transfer of data from the data input to the master latch. The slave clock input controls the transfer of data from the master latch to the slave latch.

An LSSD cell must have the following test pins:

- Scan input
- Master clock
- Slave clock
- Scan output (can be shared with functional output pin)

DFT Compiler supports LSSD equivalents for D latches, master-slave latches, and D flip-flops.

D Latch Equivalents

In functional mode, the master latch of the LSSD cell functions like the original latch in the design. The level-sensitive clock (c) enables the system data (d) to the output of the master latch (mq). In scan shift mode, the two non-overlapping test clocks (a and b) shift data from the scan input through both the master and the slave stages to the slave output (sq/scan_out). In the LSSD scan style, D latch equivalents are also single-latch LSSD cells.

Figure 5-48 on page 5-44 shows the logic diagram for the default model of the LSSD equivalent of a D latch (technology library `signal_type` attributes for the test pins appear in parentheses). Figure 5-44 shows the non-overlapping test clocks. Table 5-16 on page 5-44 shows the truth table for the default master latch; Table 5-17 on page 5-44 shows the truth table for the default slave latch.

Figure 5-48 Default LSSD D Latch Cell

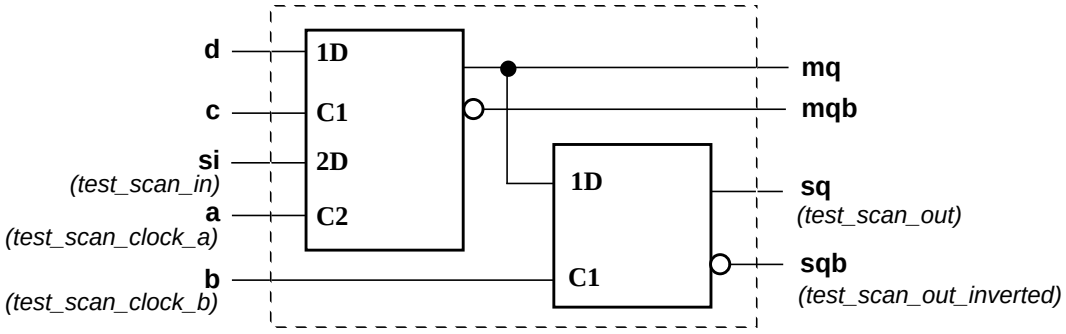


Table 5-16 Truth Table for Default LSSD Latch Cell (Master Latch)

Input				Output		Mode
d	**si**	**a**	**c**	**mq**	**mqb**	
0	X	0	1	0	1	Functional
1	X	0	1	1	0	Functional
X	0	1	0	0	1	Scan
X	1	1	0	1	0	Scan
X	X	0	0	mq	mqb	Either

X = Don’t Care

Table 5-17 Truth Table for Default LSSD D (Slave Latch)

Input		Output		Mode
mq	**b**	**sq**	**sqb**	
0	1	0	1	Either
1	1	1	0	Either
X	0	sq	sqb	Either

X = Don’t Care

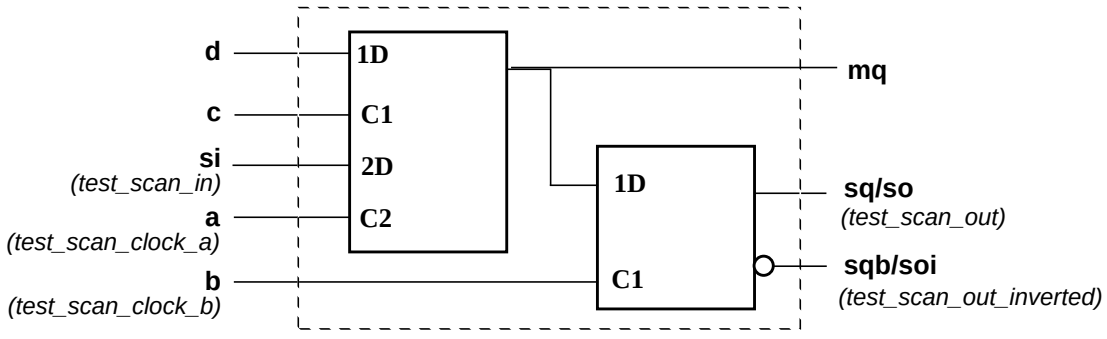
Master-Slave Latch Equivalents

In functional mode, the master and slave latches in the LSSD cell function like the master and slave latches in the original master-slave latch. System data (d) is clocked through the latch pair, with master-slave clocking on the master clock (c) and slave clock (b) inputs. Data is clocked to the slave output (sq). The master output (mq) cannot be used. In scan shift mode, two-phase, non-overlapping master-slave test clocks (a and b) are applied to the clock inputs to shift data from the scan input through both master and slave stages to the slave output (sq). In the LSSD scan style, master-slave latch equivalents are also double-latch LSSD cells.

When an LSSD cell is used as a scan equivalent for a master-slave latch, the slave clock (b) input is used in both functional mode and scan shift mode. When an LSSD cell is used as a scan equivalent for a D latch, the slave clock (b) input is used only in scan shift mode.

Figure 5-49 shows the logic diagram for the default model of the LSSD equivalent of a master-slave latch (technology library `signal_type` attributes for the test pins appear in parentheses). Figure 5-44 on page 5-37 shows the non-overlapping test clocks. The truth tables for this model are the same as for the D latch equivalent (see Table 5-16 and Table 5-17).

Figure 5-49 Default LSSD Master-Slave Latch Cell



Flip-Flop Equivalents

The LSSD equivalent for a D flip-flop is edge-triggered in functional mode and level-sensitive during scan shift. In functional mode, the LSSD cell functions as an edge-triggered cell with the system clock active and functional data clocked into the cell. In scan shift mode, two-phase, non-overlapping master-slave test clocks are applied to the master test and slave test clock inputs to shift data from the scan input to the scan output. In the LSSD scan style, D flip-flop equivalents are also referred to as clocked LSSD cells.

The actual implementation of this type of cell varies from vendor to vendor. The default model might not provide an accurate representation of your cell functionality.

Figure 5-50 on page 5-47 shows the logic diagram for the default model of the LSSD equivalent of a D flip-flop (technology library `signal_type` attributes for the test pins appear in parentheses). Figure 5-44 on page 5-37 shows the non-overlapping test clocks. Table 5-18 on page 5-47 shows the truth table for this model.

Figure 5-50 Default LSSD D Flip-Flop Cell

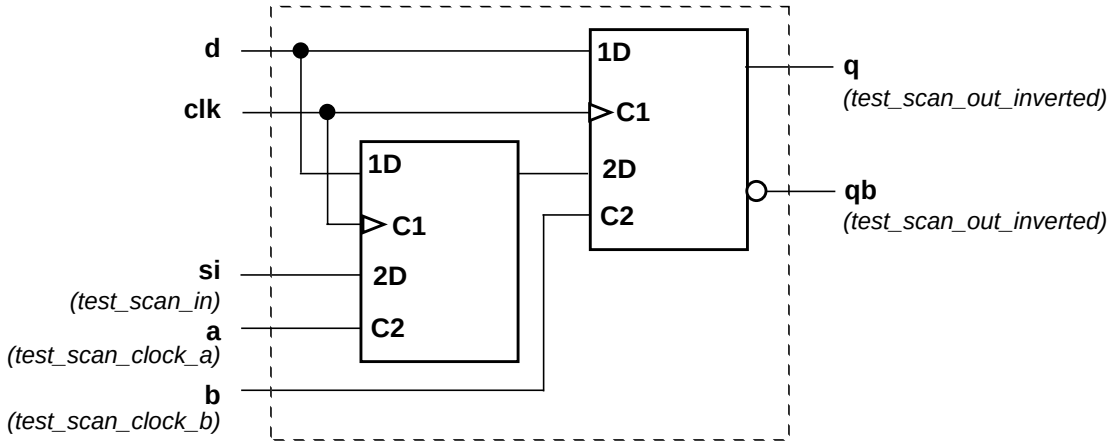


Table 5-18 Truth Table for Default LSSD D Flip-Flop Cell

Input					Output		Mode
d	**clk**	**si**	**a**	**b**	**q**	**qb**	
0	↑	X	0	0	0	1	Functional
1	↑	X	0	0	1	0	Functional
X	0	0	1	1	0	1	Scan
X	0	1	1	1	1	0	Scan
X	0/1	X	0	0	q	qb	Either

↑ = Rising Edge of Clock
X = Don’t Care

Auxiliary-Clock LSSD Scan Style

The auxiliary-clock LSSD scan style uses the standard LSSD master-slave test clocks to provide serial shift capability. In full-scan designs, the system clock is held inactive and the cell is edge-triggered by a dedicated auxiliary test clock during the capture cycle. In partial-scan designs, the dedicated auxiliary test clock is held inactive and the cell is edge-triggered by the system clock during the capture cycle.

An auxiliary-clock LSSD cell must have the following test pins:

- Scan input
- Test master clock
- Test slave clock
- Test clock
- Scan output (can be shared with functional output pin)

DFT Compiler supports auxiliary-clock LSSD equivalents for D flip-flops. The functionality of an auxiliary-clock LSSD cell can be modeled in the technology library only, using state tables. See the *Library Compiler User Guide.* for information on modeling scan cells in your technology library.

In functional mode, an auxiliary-clock LSSD cell functions as an edge-triggered cell, with system data (d) clocked into the cell by either the system clock (clk) or the auxiliary test clock (tclk). During scan shift, the auxiliary test clock (tclk) is held at logic 1 and two-phase, overlapping master-slave test clocks (a and b) are applied to the clock inputs to shift data from the scan input (si) to the scan output (q). The master test clock (a) and the slave test clock (b) are overlapping because the cell uses an inverted slave clock.

Figure 5-51 on page 5-49 shows the logic diagram for the default model of the auxiliary-clock LSSD equivalent of a D flip-flop (technology library `signal_type` attributes for the test pins appear in parentheses). Figure 5-52 on page 5-49 shows the overlapping test clocks. Table 5-19 on page 5-50 shows the truth table for this model.

Figure 5-51 Default Auxiliary-Clock LSSD Cell

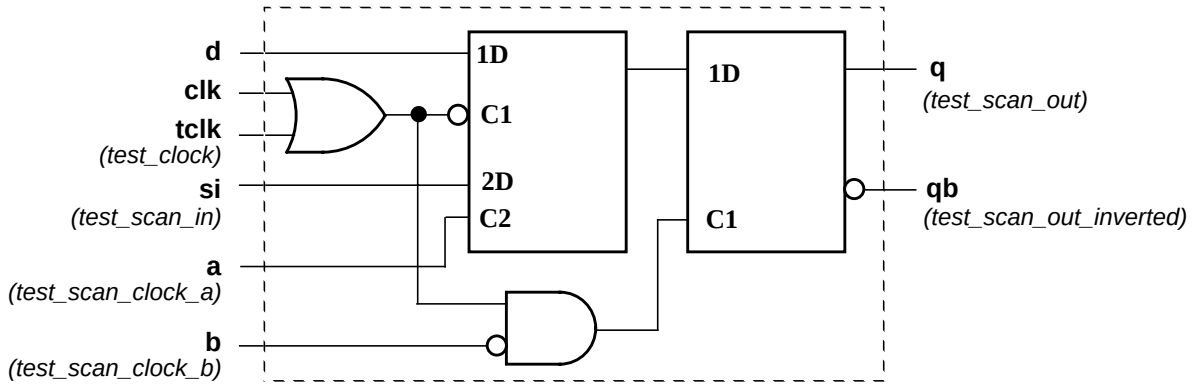


Figure 5-52 Auxiliary-Clock LSSD Test Clocks

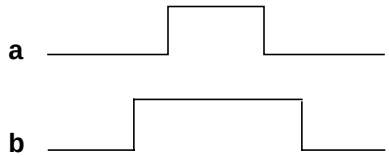


Table 5-19 Truth Table for Full-Scan Auxiliary-Clock LSSD Scan Cell

Input						Output		Mode
d	**clk**	**tclk**	**si**	**a**	**b**	**q**	**qb**	
0	↑	0	X	0	0	0	1	Functional
1	↑	0	X	0	0	1	0	Functional
X	0	1	0	1	0	0	1	Scan (shift)
X	0	1	1	1	0	1	0	Scan (shift)
0	0	↑	X	0	0	0	1	Scan (parallel)
1	0	↑	X	0	0	1	0	Scan (parallel)
X	0/1	0/1	X	0	1	q	qb	Either

↑ = Rising Edge of Clock
X = Don’t Care

6

Propagating Constraints in Hierarchical Designs

Hierarchical designs are composed of subdesigns. Using Design Compiler, you can propagate constraints up or down the hierarchy. This chapter introduces methods of propagating constraints to or from subdesigns independently of the parent design. Sections include

Hierarchical designs are composed of subdesigns. You can propagate constraints up or down the hierarchy in the following ways:

characterizing

: Captures information about the environment of specific cell instances and assigns the information as attributes on the design to which the cells are linked.

modeling

: Creates a characterized design as a library cell.

design budgeting

: Allocates timing constraints and environment constraints among the blocks in a design.

propagating constraints up the hierarchy

: Propagates clocks, timing exceptions, and disabled timing arcs from lower level subdesigns to the current design.

Characterizing Subdesigns

When you compile subdesigns separately, boundary conditions such as the input drive strengths, input signal delays (arrival times), and output loads can be derived from the parent design and set on each subdesign. You can do this manually or automatically.

manually

Use the `set_drive`, `set_driving_cell`, `set_input_delay`, `set_output_delay`, and `set_load` commands. (See Chapter 5, “Describing Logic Functions and Signal Interfaces.”)

automatically

Use the `characterize` command or the design budgeting tool.

Using the characterize Command

The `characterize` command places on a design the information and attributes that characterize its environment in the context of a specified instantiation in the top level design.

The primary purpose of `characterize` is to capture the timing environment of the subdesign. This occurs when you use `characterize` with no arguments or when you use its `-constraints`, `-connections`, or `-power` options.

The `characterize` command derives and asserts the following information and attributes on the design to which the instance is linked:

- Unless `-no_timing` is specified, `characterize` places on the subdesigns any timing characteristics previously set by the following commands:

create_clock	set_load
group_path	set_max_delay
read_timing	set_max_time_borrow
read_clusters	set_min_delay
set_annotated_check	set_multicycle_path
set_annotated_delay	set_operating_conditions
set_auto_disable_drc_nets	set_output_delay
set_drive	set_resistance
set_driving_cell	set_timing_ranges
set_false_path	set_wire_load_model
set_ideal_net	set_wire_load_mode
set_ideal_network	set_wire_load_model_selection_group
set_input_delay	set_wire_load_min_block_size

- If you specify `-constraint`, `characterize` places on the subdesigns any area, power, connection class, and design rule constraints previously set by the following commands:

set_cell_degradation	set_max_fanout
set_connection_class	set_max_power
set_dont_touch_network	set_max_transition
set_fanout_load	set_min_capacitance
set_max_area	set_min_porosity
set_max_capacitance	

- If you specify `-connection`, `characterize` places on the subdesigns the connection attributes set by the following commands: (Connection class information is applied only when you use `-constraint`.)

set_equal	set_logic_zero
set_logic_dc	set_opposite
set_logic_one	set_unconnected

- If you specify `-power`, `characterize` places on the subdesigns the switching activity information, toggle rates, and static probability previously set, calculated, or saved by the following commands:

report_power	set_switching_activity

Removing Previous Annotations

In most cases, characterizing a design removes the effects of a previous characterization and replaces the relevant information. However, in the case of back-annotation (`set_load`, `set_resistance`, `read_timing`, `set_annotated_delay`, `set_annotated_check`), the characterize step removes the annotations and cannot overwrite existing annotations made on the subdesign. In this case, you must explicitly remove annotations from the subdesign (using `reset_design`) before you run the `characterize` command again.

Optimizing Bottom Up Versus Optimizing Top Down

During optimization, you can use `characterize` with `set_dont_touch` to maintain hierarchy. This is known as bottom-up optimization, which you can apply by using a golden instance or a uniquify approach. An alternative to bottom-up optimization is top-down optimization, also called hierarchical compile. During top-down optimization, the tool automatically performs characterization and optimization for subdesigns. For more information about these optimization strategies, see the *Design Compiler User Guide*.

Deriving the Boundary Conditions

The `characterize` command automatically derives the boundary conditions of a subdesign based on its context in a parent design. It examines an instance's surroundings to obtain actual drive, load, and timing parameters and computes three types of boundary conditions:

timing conditions

: Expected signal delays at input ports.

constraints

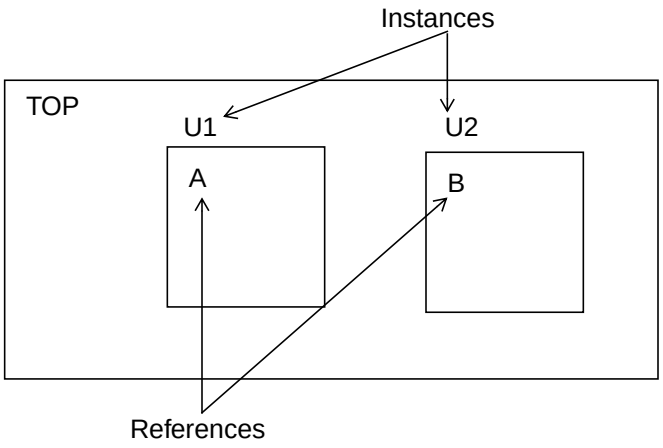
: Inherited requirements from the parent design, such as maximum delay.

connection relations

: Logical relationships between ports or between ports and power or ground, such as always logic 0, logical opposite of another port, or unconnected.

The `characterize` command summarizes the boundary conditions for one instance of a subdesign in one invocation. The result is applied to the reference. Figure 6-53 shows instances and references.

Figure 6-53 Instances and References



If a subdesign is used in more than one place, you must either characterize it manually or create a copy of the design for each instantiation and characterize each. See “Characterizing Multiple Instances” on page 6-20.

Limitations of the characterize Command

The `characterize` command provides many useful features, but do not always rely on this command to derive constraints for the subdesigns in a design hierarchy. Before you characterize a design, keep in mind the following limitations:

The `characterize` command

- Does not derive timing budgets. (It reflects the current state of the design.)
- Ignores `clock_skew` and `max_time_borrow` attributes placed on a hierarchical boundary (generally not an issue, because these attributes are usually placed on clocks and cells).

The syntax is

```
characterize      cell_list  [-no_timing] [-constraints]
                  [-connections] [-power] [-verbose]
```

Note:

For more information, see the `characterize` man page.

With no options, the `characterize` command replaces a subdesign's port signal information (clocks, port drive, input and output delays, maximum and minimum path delays, and port load) with information derived from the parent design.

The `characterize` command recognizes when the top-level design has back-annotated information (load, resistance, or delay) and to move this data down to the subdesign in preparation for subsequent optimization.

Saving Attributes and Constraints

The `characterize` command captures the timing environment of the subdesign. Use the `write_script` command with `characterize` to save the attributes and constraints for the current design. By default, the `write_script` command writes the dc_shell commands to standard output. You can redirect the output of this command to a disk file by using the redirection operator (>).

Annotation of Physical Hierarchy From characterize

If your design contains physical hierarchy annotation, you can use the `characterize` command to annotate this information to your subdesign.

For more details on how to annotate using the `characterize` command, refer to the "Reoptimizing With Floorplan Manager" chapter in the *Floorplan Manager User Guide*.

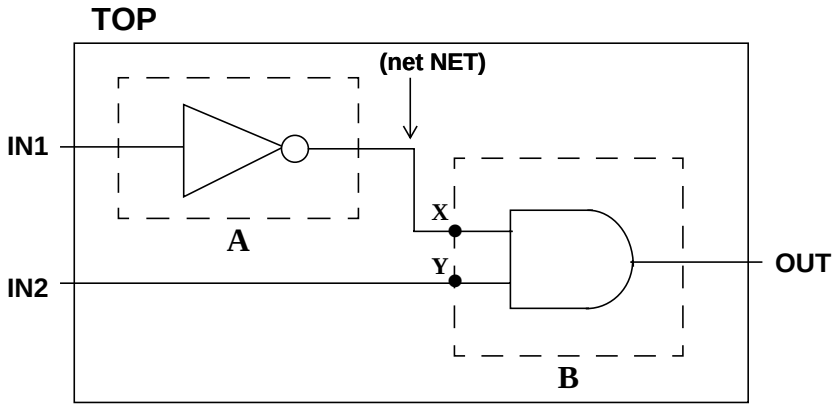
characterize Command Calculations

This section describes how the `characterize` command derives specific load and timing values for ports.

The `characterize` command uses values that allow the timing numbers on the output ports of a characterized design to be the same as if the design were flattened and then timed.

The "Load Calculations" and "Input Delay Calculations" sections that follow use the hierarchical design example in Figure 6-54 to describe the load and input delay calculations used by the `characterize` command.

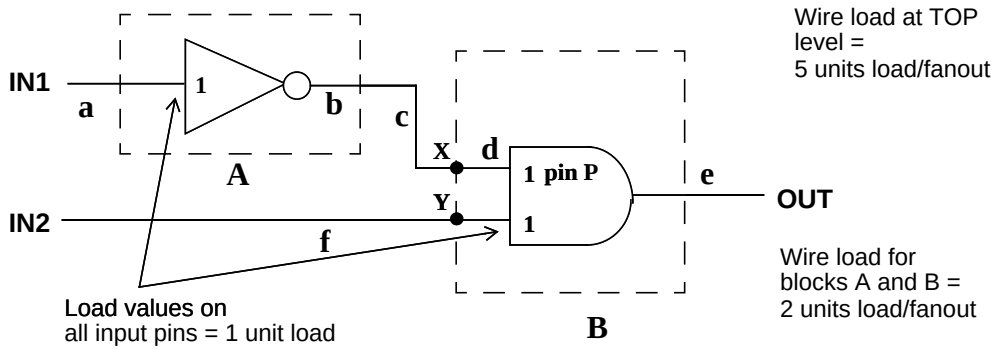
Figure 6-54 Hierarchical Design Example



Load Calculations

Figure 6-55 provides values for wire loads and input pin capacitances for the example shown in Figure 6-54.

Figure 6-55 Hierarchical Design With Annotated Loads



In Figure 6-55, a through f are wire segments used in the calculations. This example assumes a segmented wire loading model, which takes the interconnection net loads on the blocks into account and uses a linear function for the wire loads.

- The calculation for the outside load of pin P of hierarchical block B is

```
outside load = sum of the loads of all pins on the net
     loading P that are not in B
     plus the sum of the loads of all segments of net
     driving or loading P that are not in B
```

- The calculation for each segment's load is

```
segment load = number of fanouts * wire load
```

  - The calculation for the outside load on input IN1 to block A uses 0 driving pins, a fanout count of 1 for segment a, and the TOP wire load of 5 loads per fanout.

    The calculation is

```
load pins on driving net + load of segment a
     = 0 + (1 * 5)
     = 5
```

- The calculation for the outside load on the output of block A is

```
load pins on net
         + load of segment c
         + load of segment d
     = 1 (for load pin P)
     + (1 * 5)
     + (1 * 2)
     = 8
```

For each segment in the calculation, the local wire load model is used to calculate the load. That is, the calculation for block A's output pin uses TOP's wire load of 5 loads per fanout for segment c and block B's wire load of 2 loads per fanout for segment d.

- The calculation for the outside load on the output of block B is

```
load pins on net + load of segment e
     = 0 + (1 * 5)
     = 5
```

- The calculation for the outside load on the input pin X of block B is

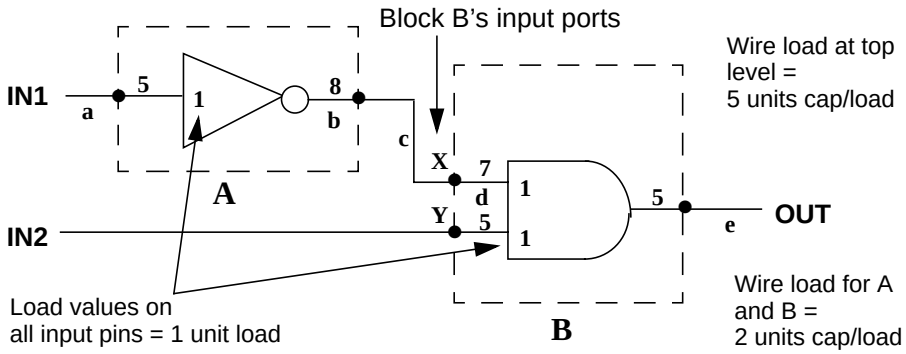
```
load pins on net
         + load of segment b
         + load of segment c
         = 0 + (1 * 2) + (1 * 5)
         = 7
```

- The calculation for the outside load on the input pin Y is

```
pin loads on driving net + load of segment f
     = 0 + 1 * 5
     = 5
```

Figure 6-56 shows the modified version of Figure 6-54 with the outside loads annotated.

Figure 6-56 Loads After Characterization



Input Delay Calculations

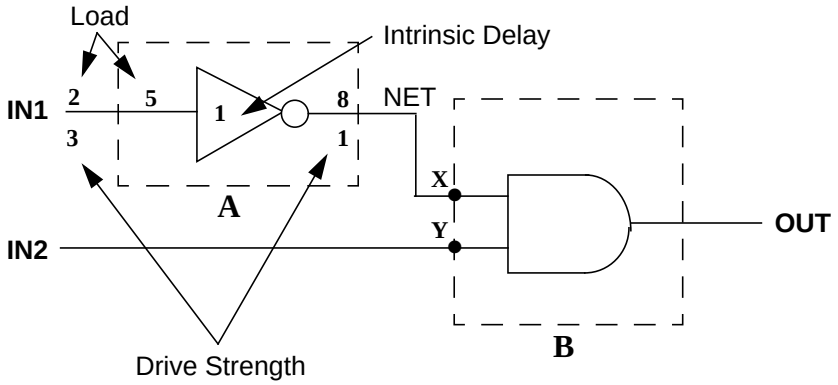
Because characterizing provides accurate details of outside loads, the path delays of input signals reflect only the delay through the intrinsic delay of the last gate driving the port. The path delays of input signals do not include the gate's load delay or the connect delay on the net.

For example, the characterized input delay on the input pins of block B is calculated from the delay to the pin that drives the port being characterized, without the gate's load delay or the connect delay on the net.

The timing calculations for characterizing block B follow Figure 6-57.

Figure 6-57 on page 6-14 shows the default drive strengths and intrinsic delays of block A and signal IN1.

Figure 6-57 Design With Annotations for Timing Calculations



The delay calculation for input pin X is

```
drive strength at IN1 * (wire load + pin load)
     + intrinsic delay of A's cell
     = 3 * (5 + 2 + 1)
     + 1
     = 25
```

Characterizing Subdesign Port Signal Interfaces

The `characterize` command with no options replaces a subdesign's port signal information (clocks, port drive, input and output delays, maximum and minimum path delays, and port load) with information derived from the parent design. The subdesign also inherits operating conditions and timing ranges from the top-level design.

You can manually set port signal information by using the commands `create_clock`, `set_clock_latency`, `set_clock_uncertainty`, `set_propagated_clock`, `set_drive`, `set_driving_cell`, `set_input_delay`, `set_output_delay`, `set_max_delay`, `set_min_delay`, and `set_load`. For information about these commands, see Chapter 2,

“Constraining Designs,” Chapter 4, “Specifying Clocks and Clock Networks,” and Chapter 5, “Describing Logic Functions and Signal Interfaces.”

The `characterize` command sets the wire loading model selection group and model for subdesigns. The subdesigns inherit the top-level wire loading mode. The wire loading model for subdesigns is determined as follows:

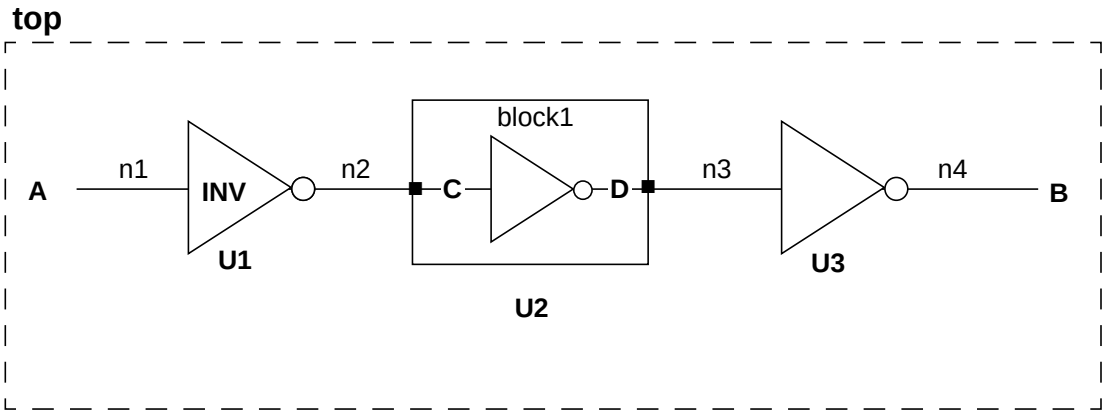
- If the top-level mode is top, the subdesigns inherit the top-level wire loading model. Unless an instance specific model or selection group has be specified. Instance specific model and selection group settings take precedence over design level settings.
- If the top-level mode is enclosed or segmented, the wire loading model is based on the following:
  - If no wire loading model is defined for the lower block and the wire load cannot be determine by the area, the wire loading model of the top-level design is used.
  - If no wire loading model is defined for the lower block but the wire load can be determine by the area, the wire loading model is reselected at compile time, based on the cell area.

Combinational Design Example

Figure 6-58 shows subdesign block in the combinational design top. Set the port interface attributes either manually or automatically.

- Script 1 uses characterize to set the attributes automatically.
- Script 2 sets the attributes manually.

Figure 6-58 Characterizing Drive, Timing, and Load Values—Combinational Design




```
current_design = top
set_input_delay 0 A
set_max_delay 10 -to B
```

Script 1

```
current_design = top
characterize U2
```

Script 2

```
current_design = block1
set_driving_cell -cell INV {C}
set_input_delay 3.3 C
set_load    1.3 D
set_max_delay 9.2 -to D
current_design = top
```

In Script 2,

Line 2 captures driving cell information.

Line 3 sets the arrival time of net n2 as 3.3.

Line 4 sets the load of U3 as 1.3.

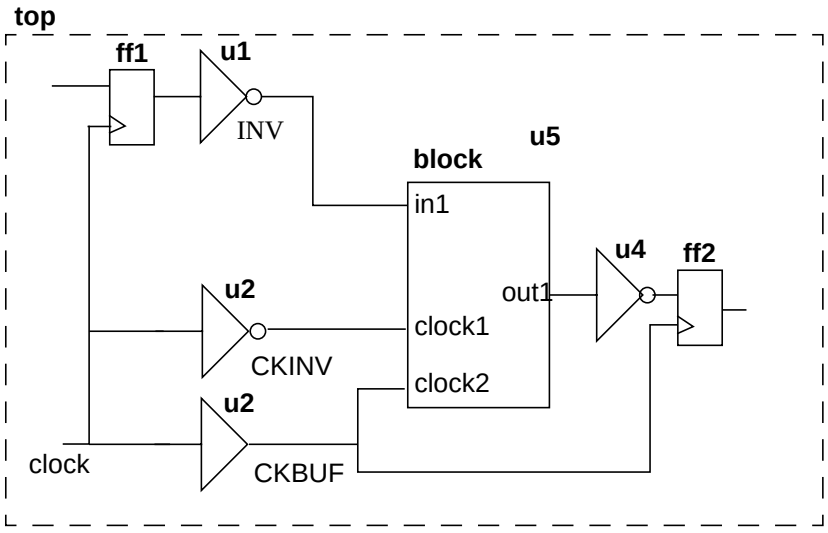
Line 5 sets the inherited `set_max_delay`, which is 10 – .8 (.4 for each inverter).

Sequential Design Example

Figure 6-59 shows subdesign block in the sequential design top. Set the port interface information either manually or automatically.

- Script 1 sets the information manually.
- Script 2 uses `characterize` to set the information automatically.

Figure 6-59 Characterizing Sequential Design Drive, Timing, and Load Values



Script 1

```
current_design = top
create_clock -period 10 -waveform {0 5} clock
current_design = block
create_clock -name clock -period 10 -waveform {0 5} clock1
create_clock -name clock_bar -period 10 \
   -waveform {5 10} clock2
```

```
/* delay from ff1/CP to u5/in1 is 1.8 */
set_input_delay -clock clock 1.8 in1
/* delay of u4 plus ff2 setup time is 1.2 */
set_output_delay -clock clock 1.2 out1
set_driving_cell -cell INV -input_transition_rise 1 in1
set_driving_cell -cell CKINV clock1
set_driving_cell -cell CKBUF clock2
set_load 0.85 out1 /* outside load on out1 net */
current_design = top
```

Script 2

```
current_design = top
create_clock -period 10 -waveform {0 5} clock
characterize u5
```

Characterizing Subdesign Constraints

The `characterize -constraints` command uses values derived from the parent design to replace the `max_area`, `max_fanout`, `fanout_load`, `max_capacitance`, and `max_transition` attributes of a subdesign. For information about these commands, see Chapter 2, “Constraining Designs.”

Characterizing Subdesign Logical Port Connections

The `characterize -connections` command uses connection attributes derived from the parent design to replace the port connection attributes of a subdesign.

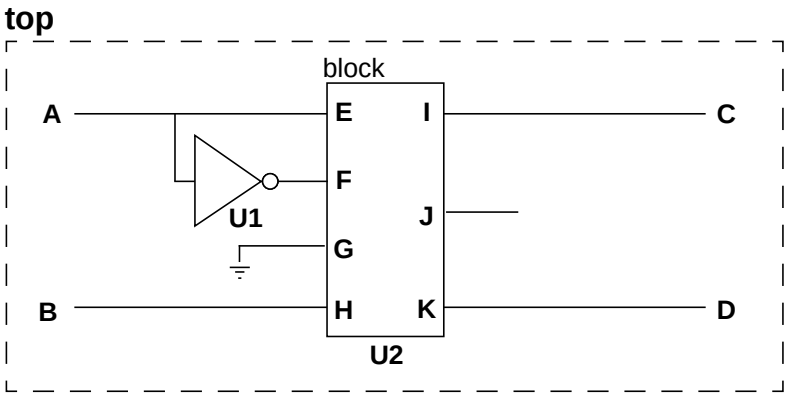
The connection attributes are those set by the commands `set_equal`, `set_opposite`, `set_logic_one`, `set_logic_zero`, and `set_unconnected`. For information about these commands, see Chapter 5, “Describing Logic Functions and Signal Interfaces.”

Example

Figure 6-60 shows subdesign block in design top. Set logical port connections either manually or automatically.

- Script 1 sets the attributes manually.
- Script 2 uses `characterize -no_timing -connections` to set the attributes automatically. The `-no_timing` option inhibits computation of port timing information.

Figure 6-60 Characterizing Port Connection Attributes



Script 1

```
current_design = block
set_opposite     E F
set_logic_zero   G
set_unconnected  J
```

Script 2

```
current_design = top
characterize U2 -no_timing -connections
```

Characterizing Multiple Instances

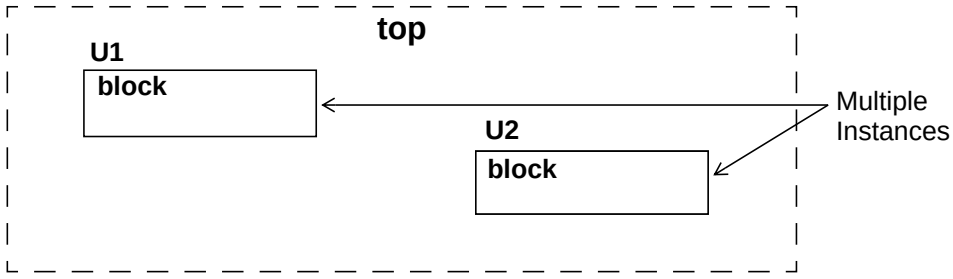
The `characterize` command summarizes the boundary conditions for one instance of a subdesign in each invocation. If a subdesign is used more than once, use the `uniquify` command to make each instance distinctive before using `characterize` for each instance.

The `uniquify` command creates copies of subdesigns that are referenced more than once. It then renames the copies and updates the corresponding cell references. For information about the `uniquify` command, see the *Design Compiler User Guide*.

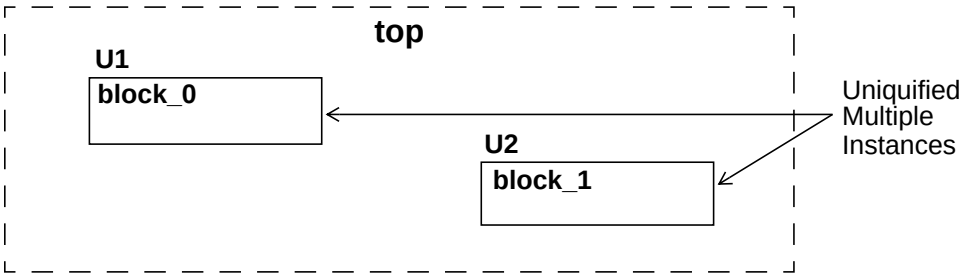
Example

Figure 6-61 shows how to use `uniquify` and `characterize` for the subdesign block, which is referenced in cells U1 and U2.

Figure 6-61 Characterizing a Subdesign Referenced Multiple Times




```
dc_shell> current_design = top
dc_shell> uniquify -reference block
/*uniquify all copies of 'block'*/
```




```
dc_shell> characterize U1
dc_shell> characterize U2
```

Characterizing Designs With Timing Exceptions

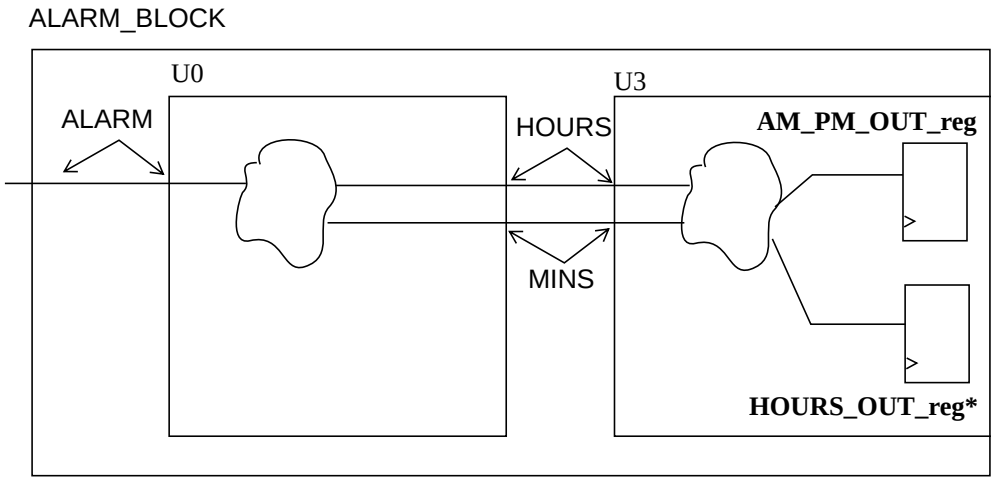
When paths crossing design hierarchies contain different timing exceptions, `characterize` creates timing constraints with virtual clocks to capture this information.

Possible timing exceptions include `set_multicycle_path`, `set_false_path`, `set_max_delay`, and `set_min_delay`. The virtual clock scheme can also handle multiple clocks.

Example

Figure 6-62 shows the uncompressed `characterize -verbose` result of U0 block. In the example, the path from ALARM to HOURS_OUT_reg* in U3 is constrained as a two-cycle path.

Figure 6-62 characterize -verbose Result of U0 Block




```
create_clock -period 10 -waveform {0 5} find(port,"CLK")
create_clock -name "CLK_virtual1" -period 10 -waveform {0 5}
create_clock -name "CLK_virtual2" -period 10 -waveform {0 5}
create_clock -name "CLK_virtual3" -period 10 -waveform {0 5}
create_clock -name "CLK_virtual4" -period 10 -waveform {0 5}
create_clock -name "CLK_virtual5" -period 10 -waveform {0 5}
set_input_delay 2 -clock "CLK" find(port,"MINUTES_BUTTON")
set_input_delay 2 -clock "CLK" find(port,"HOURS_BUTTON")
set_input_delay 2 -clock "CLK_virtual1" find(port,"ALARM_BUTTON")
set_output_delay 7.62796 -max -rise -clock "CLK" find(port,"MINS")
set_output_delay 6.93955 -max -fall -clock "CLK" find(port,"MINS")
set_output_delay 2.20324 -min -rise -clock "CLK" find(port,"MINS")
set_output_delay 2.40013 -min -fall -clock "CLK" find(port,"MINS")
set_output_delay 8.05575 -add_delay -max -rise -clock "CLK_virtual2" \
find(port,"MINS")
set_output_delay 6.83933 -add_delay -max -fall -clock "CLK_virtual2" \
find(port,"MINS")
set_output_delay 2.89679 -add_delay -min -rise -clock "CLK_virtual2" \
find(port,"MINS")
set_output_delay 2.80452 -add_delay -min -fall -clock "CLK_virtual2" \
find(port,"MINS")
set_output_delay 8.96438 -add_delay -max -rise -clock "CLK_virtual3" \
find(port,"MINS")
set_output_delay 8.53005 -add_delay -max -fall -clock "CLK_virtual3" \
find(port,"MINS")
set_output_delay 2.94413 -add_delay -min -rise -clock "CLK_virtual3" \
find(port,"MINS")
set_output_delay 3.1334 -add_delay -min -fall -clock "CLK_virtual3" \
find(port,"MINS")
set_output_delay 9.59218 -add_delay -max -rise -clock "CLK_virtual4" \
```

```
find(port,"MINS")
set_output_delay 8.77876 -add_delay -max -fall -clock "CLK_virtual4" \
find(port,"MINS")
set_output_delay 3.01398 -add_delay -min -rise -clock "CLK_virtual4" \
find(port,"MINS")
set_output_delay 2.93837 -add_delay -min -fall -clock "CLK_virtual4" \
find(port,"MINS")
set_output_delay 10.232 -add_delay -max -rise -clock "CLK_virtual5" \
find(port,"MINS")
set_output_delay 9.4791 -add_delay -max -fall -clock "CLK_virtual5" \
find(port,"MINS")
set_output_delay 3.90222 -add_delay -min -rise -clock "CLK_virtual5" \
find(port,"MINS")
set_output_delay 3.57818 -add_delay -min -fall -clock "CLK_virtual5" \
find(port,"MINS")
set_output_delay 7.34128 -max -rise -clock "CLK" find(port,"HOURS")
set_output_delay 8.05292 -max -fall -clock "CLK" find(port,"HOURS")
set_output_delay 2.80185 -min -rise -clock "CLK" find(port,"HOURS")
set_output_delay 2.62819 -min -fall -clock "CLK" find(port,"HOURS")
set_output_delay 7.24106 -add_delay -max -rise -clock "CLK_virtual2" \
find(port,"HOURS")
set_output_delay 8.48071 -add_delay -max -fall -clock "CLK_virtual2" \
find(port,"HOURS")
set_output_delay 3.20624 -add_delay -min -rise -clock "CLK_virtual2" \
find(port,"HOURS")
set_output_delay 3.32175 -add_delay -min -fall -clock "CLK_virtual2" \
find(port,"HOURS")
set_output_delay 8.93178 -add_delay -max -rise -clock "CLK_virtual3" \
find(port,"HOURS")
set_output_delay 9.38934 -add_delay -max -fall -clock "CLK_virtual3" \
find(port,"HOURS")
set_output_delay 3.53513 -add_delay -min -rise -clock "CLK_virtual3" \
find(port,"HOURS")
set_output_delay 3.36908 -add_delay -min -fall -clock "CLK_virtual3" \
find(port,"HOURS")
set_output_delay 9.18049 -add_delay -max -rise -clock "CLK_virtual4" \
find(port,"HOURS")
set_output_delay 10.0171 -add_delay -max -fall -clock "CLK_virtual4" \
find(port,"HOURS")
set_output_delay 3.3401 -add_delay -min -rise -clock "CLK_virtual4" \
find(port,"HOURS")
set_output_delay 3.43894 -add_delay -min -fall -clock "CLK_virtual4" \
find(port,"HOURS")
set_output_delay 9.88083 -add_delay -max -rise -clock "CLK_virtual5" \
find(port,"HOURS")
set_output_delay 10.6569 -add_delay -max -fall -clock "CLK_virtual5" \
find(port,"HOURS")
set_output_delay 3.9799 -add_delay -min -rise -clock "CLK_virtual5" \
```

```
find(port,"HOURS")
set_output_delay 4.32717 -add_delay -min -fall -clock "CLK_virtual5" \
find(port,"HOURS")
set_multicycle_path 2 -from find(clock,"CLK_virtual1") -to \
find(clock,"CLK_virtual2")
set_multicycle_path 2 -from find(clock,"CLK_virtual1") -to \
find(clock,"CLK_virtual3")
set_multicycle_path 2 -from find(clock,"CLK_virtual1") -to \
find(clock,"CLK_virtual4")
set_multicycle_path 2 -from find(clock,"CLK_virtual1") -to \
find(clock,"CLK_virtual5")
```

Design Budgeting

The design budgeting tool implements automatic, iterative design budgeting. Design budgeting, or constraint allocation, is the process of allocating timing constraints and environment constraints among the blocks in a design. The blocks can be processed separately by synthesis and other module-level tools. Design budgeting provides a complete specification of timing and environmental budgets at subblock boundaries.

Design budgeting differs from using the `characterize` command, in that it does not use a snapshot of the external environment of a subblock. Design budgeting allocates the timing and environmental constraints for a block by taking into account the portion of the total constraints that block requires.

The design budgeting tool is described in a separate book, the *Design Budgeting User Guide*.

Propagating Constraints up the Hierarchy

If you have hierarchical designs and compile the subdesigns, then move up to the higher-level blocks (bottom-up compilation), you can propagate clocks, timing exceptions, and disabled timing arcs from lower-level .db files to the current design, using the `propagate_constraints` command.

The syntax is

```
propagate_constraints
                  [-design design_list]
                  [-clocks] [-false_path]
                  [-multicycle_path]
                  [-max_delay] [-min_delay]
                  [-disable_timing] [-gate_clock -all]
                  [-ignore_through_port_exceptions]
                  [-ignore_from_or_to_port_exceptions]
                  [-verbose] [-dont_apply]
                  [-output file_name]
```

Note:

For more information, see the `propagate_constraints` man page.

Note:

All the options show conflicts.

Methodology for Propagating Constraints Upward

Use the `propagate_constraints` command to propagate constraints from lower levels of hierarchy to the current design. If you do not use the command, you can propagate constraints from a higher-level design to the `current_instance`, but you cannot propagate constraints set on a lower-level block to the higher-level blocks in which it is instantiated.

Note:

Using the `propagate_constraints` command might cause memory usage to increase.

The following shows an example methodology. Assume that A is the top-level and B is the lower-level design.

```
current_design B
include constraints.scr
compile
current_design A
propagate_constraints -design B     /*or use -verbose
                                    -dont_apply -output
                                    file.cons*/
report_timing_requirements
compile
report_timing                       /*include the output*/
```

To generate a report of all the constraints that were propagated up, use the `-verbose` and `-dont_apply` options and redirect the output to a file:

```
propagate_constraints -design name -verbose -dont_apply -output report.cons
```

Use the write `-format db` command to save the .db file with the propagated constraints so there is no need to go through the propagation again when restarting a new dc_shell session.

Handling of Conflicts Between Designs

The following shows what happens if there are conflicts between the lower-level and top-level designs.

Clock name conflict

The lower-level clock has the same name as the clock of the current design (or another block).

The clock is not propagated. A warning is issued.

Clock source conflict

A clock source of a lower-level block is already defined as a clock source of a higher-level block.

The lower-level clock is not propagated. A warning is issued.

Exceptions from or to an unpropagated clock

This can be either a virtual clock, or a clock that was not propagated from that block due to a conflict.

Exceptions

A lower-level exception overrides a higher-level exception that is defined on the exact same path.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W