

# 9

## Analyzing and Debugging Your Design

---

Use the reports generated by Design Compiler to analyze and debug your design. You can generate reports both before and after you compile your design. Generate reports before compiling to check that you have set attributes, constraints, and design rules properly. Generate reports after compiling to analyze the results and debug your design.

This chapter contains the following sections:

- [Checking for Design Consistency](#)
- [Analyzing Your Design During Optimization](#)
- [Analyzing Design Problems](#)
- [Analyzing Timing Problems](#)
- [Debugging Specific Problems](#)

## Checking for Design Consistency

A design is consistent when it does not contain errors such as unconnected ports, constant-valued ports, cells with no input or output pins, mismatches between a cell and its reference, multiple driver nets, connection class violations, or recursive hierarchy definitions.

Use the `check_design` command to verify the design consistency. The `check_design` command reports a list of warning and error messages. It reports

- An error if it finds a problem that Design Compiler cannot resolve. You cannot compile a design that has `check_design` errors.

The `check_design` command always reports error messages.

- A warning if it finds a problem that indicates a corrupted design or a design mistake not severe enough to cause the `compile` command to fail.

By default, the `check_design` command reports all warning messages. You can reduce the output by summarizing the warnings (by using the `-summary` option) or by disabling the warnings (by using the `-no_warnings` option).

By default, the `check_design` command validates the entire design hierarchy. To limit the validation to the current design, specify the `-one-level` option.

---

## Analyzing Your Design During Optimization

Design Compiler provides the following capabilities for analyzing your design during optimization:

- Customizing the compile log
- Saving intermediate design databases

The following sections describe these capabilities.

---

### Customizing the Compile Log

The compile log records the status of the compile run. Each optimization task has an introductory heading, followed by the actions taken while performing that task. There are four tasks in which Design Compiler works to reduce the compile cost function:

- Delay optimization
- Design rule fixing, phase 1
- Design rule fixing, phase 2
- Area optimization

While completing these tasks, Design Compiler performs many trials to determine how to reduce the cost function. For this reason, these tasks are collectively known as the trials phase of optimization.

By default, Design Compiler logs each action in the trials phase by providing the following information:

- Elapsed time

- Design area
- Worst negative slack
- Total negative slack
- Design rule cost
- Endpoint being worked on

You can customize the trials phase output by setting the `compile_log_format` variable. Table 9-1 lists the available data items and the keywords used to select them. For more information about customizing the compile log, see the man page for the `compile_log_format` variable.

*Table 9-1 Compile Log Format Keywords*

Column	Column Header	Key Word	Column Description
Area	AREA	area	Shows the area of the design.
CPU seconds	CPU SEC	cpu	Shows the process CPU time used (in seconds).
Design rule cost	DESIGN RULE COST	drc	Measures the difference between the actual results and user-specified design rule constraints.
Elapsed time	ELAPSED TIME	elap_time	Tracks the elapsed time since the beginning of the current compile or <code>reoptimize_design</code> .
Endpoint	ENDPOINT	endpoint	Shows the endpoint being worked on. When delay violations are being fixed, the endpoint is a cell or a port. When design rule violations are being fixed, the endpoint is a net. When area violations are being fixed, no endpoint is printed.

*Table 9-1 Compile Log Format Keywords (continued)*

<b>Column</b>	<b>Column Header</b>	<b>Key Word</b>	<b>Column Description</b>
Maximum delay cost	MAX DELAY COST	max_delay	Shows the maximum delay cost of the design.
Megabytes of memory	MBYTES	mem	Shows the process memory used (in MB).
Minimum delay cost	MIN DELAY COST	min_delay	Shows the minimum delay cost of the design.
Path group	PATH GROUP	group_path	Shows the path group of an endpoint.
Time of day	TIME OF DAY	time	Shows the current time.
Total negative slack	TOTAL NEG SLACK	tns	Shows the total negative slack of the design.
Trials	TRIALS	trials	Tracks the number of transformations that the optimizer tried before making the current selection.
Worst negative slack	WORST NEG SLACK	wns	Shows the worst negative slack of the current path group.

---

## **Saving Intermediate Design Databases**

Design Compiler provides the capability to output an intermediate design database during the trials phase of the optimization process. This capability is called checkpointing. Checkpointing saves the entire hierarchy of the intermediate design. You can use this intermediate design to debug design problems, as described in “Analyzing Design Problems” on page 9-8.

Design Compiler supports both manual checkpointing and automatic checkpointing. The following sections describe these options.

## Manual Checkpointing

You can manually checkpoint the design at any time after optimization has entered the trials phase by using the Ctrl-c interrupt. You can checkpoint the design multiple times throughout the optimization process; however, each checkpoint overwrites the previous checkpoint file.

When running Design Compiler interactively, pressing Ctrl-c once causes the following menu to appear (after a short delay):

```
Please type in one of the following options:
  1 to Write out the current state of the design
  2 to Abort optimization
  3 to Kill the process
  4 to Continue optimization
Please enter a number:
```

Select option 1 to checkpoint the design. By default, Design Compiler writes the intermediate design database to `./CHECKPOINT.db`. You can specify the file name by using the `compile_checkpoint_filename` variable. The directory you specify must exist and be writable. After you checkpoint the design, Design Compiler displays the menu again. Select option 2, 3, or 4, depending on how you want to proceed.

When Design Compiler is running in the background, the behavior of the Ctrl-c interrupt depends on the optimization phase and the number of times you press Ctrl-c.

Before the trials phase,

- Pressing Ctrl-c once stops the optimization process
- Pressing Ctrl-c three times stops the `dc_shell` process

After the trials phase,

- Pressing Ctrl-c once checkpoints the design after a minimum delay of five seconds
- Pressing Ctrl-c three times stops the optimization process
- Pressing Ctrl-c five times stops the `dc_shell` process

If you use the UNIX `tee` command to print the results of a background run to both the screen and a log file, you must specify the `-i` option of the `tee` command to use checkpointing. For example,

```
% dc_shell -f run.scr | tee -i run.out
```

If you do not use the `-i` option, pressing Ctrl-c interrupts the `tee` process and the `dc_shell` process never sees the interrupt signal. The `-i` option forces the `tee` process to ignore interrupt signals.

---

## Automatic Checkpointing

You can automatically checkpoint the design based on CPU time intervals, optimization phase, or both.

To checkpoint based on elapsed CPU time, set the `compile_checkpoint_cpu_interval` variable to the desired time interval (in minutes). Each checkpoint overwrites the previous checkpoint file.

To checkpoint based on optimization phase, set the `compile_checkpoint_phases` variable to `true`. This creates a checkpoint file at the following points: before starting delay optimization (pre-delay), before starting the first phase of design rule fixing (pre-DRC1), before starting the second phase of design rule fixing (pre-DRC2), and before starting area optimization (pre-area).

Design Compiler saves each checkpoint in a separate file. Table 9-2 lists the default file name for each phase and the variable used to control each file name. You can turn off checkpointing for any phase by setting the corresponding variable to “none”.

*Table 9-2 Phased-Based Checkpoint Files*

Phase	Default File Name	Variable
Pre-delay	./CHECKPOINT_PRE_DELAY.db	compile_checkpoint_pre_delay_filename
Pre-DRC1	./CHECKPOINT_PRE_DRC1.db	compile_checkpoint_pre_drc1_filename
Pre-DRC2	./CHECKPOINT_PRE_DRC2.db	compile_checkpoint_pre_drc2_filename
Pre-area	./CHECKPOINT_PRE_AREA.db	compile_checkpoint_pre_area_filename

## Analyzing Design Problems

Table 9-3 shows the design analysis commands provided by Design Compiler. For additional information about these commands, see the man pages.

*Table 9-3 Commands to Analyze Design Objects*

Object	Command	Description
Design	report_design report_area report_hierarchy report_resources	Reports design characteristics. Reports design size and object counts. Reports design hierarchy. Reports resource implementations.
Instances	report_cell	Displays information about instances.
References	report_reference	Displays information about references.
Pins	report_transitive_fanin report_transitive_fanout	Reports fanin logic. Reports fanout logic.

*Table 9-3 Commands to Analyze Design Objects (continued)*

<b>Object</b>	<b>Command</b>	<b>Description</b>
Ports	report_port	Displays information about ports.
	report_bus	Displays information about bused ports.
	report_transitive_fanin	Reports fanin logic.
	report_transitive_fanout	Reports fanout logic.
Nets	report_net	Reports net characteristics.
	report_bus	Reports bused net characteristics.
	report_transitive_fanin	Reports fanin logic.
	report_transitive_fanout	Reports fanout logic.
Clocks	report_clock	Displays information about clocks.

## Analyzing Timing Problems

Before you begin debugging timing problems, verify that your design meets the following requirements:

- You have defined the operating conditions.
- You have specified realistic constraints.
- You have appropriately budgeted the timing constraints.
- You have properly constrained the paths.
- You have described the clock skew.

If your design does not meet these requirements, make sure it does before you proceed.

After producing the initial mapped netlist, use the `report_constraint` command to check your design's performance.

Table 9-4 lists the timing analysis commands.

*Table 9-4 Timing Analysis Commands*

<b>Analysis Task</b>	<b>Command</b>
Show operating conditions, wire load model and mode, timing ranges, internal input and output, and disabled timing arcs.	report_design
Check for unconstrained timing paths and clock-gating logic.	check_timing
Show unconstrained input and output ports and port loading.	report_port
Show all timing exceptions set on the design.	report_timing_requirements
Check the clock definition and clock skew information.	report_clock
Check internal clock and unused registers.	derive_clocks
Show all timing path groups in the design.	report_path_group
Check the timing of the design.	report_timing
Check the design constraints.	report_constraint
Report the details of a delay arc calculation.	report_delay_calculation

---

## Debugging Specific Problems

This section provides examples for debugging design problems you might encounter and the workarounds for them.

---

## Analyzing Cell Delays

Some cell delays shown in the full path timing report might seem too large. Use the `report_delay_calculation` command to determine how Design Compiler calculated a particular delay value.

For example, Example 9-1 shows a full path timing report with a large cell delay value.

### Example 9-1 Full Path Timing Report

```
*****
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : Adder8
Version: 1999.05
Date   : Mon Jan 4 10:56:49 1999
*****
```

```
Operating Conditions:
Wire Loading Model Mode: top
```

```
Startpoint: cin (input port)
Endpoint: cout (output port)
Path Group: (none)
Path Type: max
```

Point	Incr	Path
input external delay	0.00	0.00 f
cin (in)	0.00	0.00 f
U19/Z (AN2)	0.87	0.87 f
U18/Z (EO)	1.13	2.00 f
add_8/U1_1/CO (FA1A)	2.27	4.27 f
add_8/U1_2/CO (FA1A)	1.17	5.45 f
add_8/U1_3/CO (FA1A)	1.17	6.62 f
add_8/U1_4/CO (FA1A)	1.17	7.80 f
add_8/U1_5/CO (FA1A)	1.17	8.97 f
add_8/U1_6/CO (FA1A)	1.17	10.14 f
add_8/U1_7/CO (FA1A)	1.17	11.32 f
U2/Z (EO)	1.06	12.38 f

```

cout (out)                0.00      12.38 f
data arrival time                12.38 f
-----

```

(Path is unconstrained)

The delay from port cin through cell FA1A seems large (2.27 ns). Enter the following command to determine how Design Compiler calculated this delay:

```

dc_shell> report_delay_calculation \
          -from add_8/U1_1/A -to add_8/U1_1/CO

```

Example 9-2 shows the results of this command.

### *Example 9-2 Delay Calculation Report*

```

*****
Report : delay_calculation
Design : Adder8
Version: 1997.01
Date   : Mon Apr  7 13:23:12 1997
*****

From pin:                add_8/U1_1/A
To pin:                  add_8/U1_1/CO

arc sense:               unate
arc type:                 cell
Input net transition times: Dt_rise = 0.1458, Dt_fall = 0.0653

Rise Delay computation:
rise_intrinsic           1.89 +
rise_slope * Dt_rise     0 * 0.1458 +
rise_resistance * (pin_cap + wire_cap) / driver_count
0.1458 * (2 + 0) / 1
-----
Total                    2.1816

Fall Delay computation:
fall_intrinsic           2.14 +
fall_slope * Dt_fall     0 * 0.0653 +
fall_resistance * (pin_cap + wire_cap) / driver_count
0.0669 * (2 + 0) / 1
-----
Total                    2.2738

```

---

## Finding Unmapped Cells

All unmapped cells have the `is_unmapped` attribute. You can use the `filter` and `find` commands to find all unmapped components:

```
dc_shell> filter find(-hier, cell, "**") "@is_unmapped==true"
```

---

## Finding Black Box Cells

All black box cells have the `is_black_box` attribute. You can use the `filter` and `find` commands to find all black box cells:

```
dc_shell> filter find(-hier, cell, "**") \  
          "@is_black_box==true"
```

---

## Finding Hierarchical Cells

All hierarchical cells have the `is_hierarchical` attribute. You can use the `filter` and `find` commands to find all hierarchical cells:

```
dc_shell> filter find(design, "**") "@is_hierarchical==true"
```

---

## Controlling Reporting of Bidirectional Port Violations

Bidirectional ports can be in either input mode or output mode, as determined by the value of the control signal. Because the timing verifier does not know the value of the control signal, the bidirectional port has two sets of constraints: one created by you and one propagated from core logic through the bidirectional cell and then looped back into the bidirectional cell. Design Compiler takes the most restrictive constraint as the path constraint but does not guarantee that it is the correct constraint.

To verify the timing constraints on a bidirectional port, use the `set_disable_timing` command to disable the timing arc for one mode. The following command sequence verifies the timing constraints for each mode independently:

```
/* check timing constraints in input mode */
set_disable_timing my_lib/bidi -from A -to IO
report_constraint -all_violators

/* check timing constraints in output mode */
remove_attribute my_lib/bidi disable_timing
set_disable_timing my_lib/bidi -from IO -to ZI
report_constraint -all_violators
```

If your design does not use one of the bidirectional modes, mask false violation messages by using the `set_disable_timing` command to disconnect the timing arc for the unused mode.

---

## Disabling Reporting of Scan Chain Violations

If your design contains scan chains, it is likely that these chains are not designed to run at system speed. This can cause false violation messages when you perform timing analysis. To mask these messages, use the `set_disable_timing` command to break the scan-related timing paths (scan input to scan output and scan enable to scan output).

```
dc_shell> set_disable_timing my_lib/scanf \
          -from TI -to Q
dc_shell> set_disable_timing my_lib/scanf \
          -from CP -to TE
```

This example assumes that

- `scanf` is the scan cell in your technology library
- `TI` is the scan input pin on the `scanf` cell

- TE is the scan enable on the scanf cell
- Q is the scan output pin on the scanf cell

You can use the following command sequence to identify the scan pins in your technology library.

```
filter find(cell, my_lib/*) "@is_sequential==true"
seq_cell_list = dc_shell_status
foreach (seq_cell, seq_cell_list) {
    seq_pins = seq_cell + "/*"
    filter find(pin, seq_pins) \
        "@signal_type==test_scan_in"
    if (dc_shell_status) {
        echo "scan input for " seq_cell
        filter find(pin, seq_pins) \
            "@signal_type==test_scan_out"
        echo "scan output for " seq_cell
    }
}
```

---

## Insulating Interblock Loading

Design Compiler determines load distribution in the driving block. If a single output port drives many blocks, a huge incremental cell delay can result. To insulate the interblock loading, fan the heavily loaded net to multiple output ports in the driving block. Evenly divide the total load among these output ports.

---

## Preserving Dangling Logic

By default, Design Compiler optimizes away dangling logic. Use one of the following methods to preserve dangling logic (for example, spare cells) during optimization:

- Place the `dont_touch` attribute on the dangling logic.

- Connect the dangling logic to a dummy port.

---

## Preventing Wire Delays on Ports

If your design contains unwanted wire delays between ports and I/O cells, you can remove these wire delays by specifying zero resistance (infinite drive strength) on the net. Use the `set_resistance` command to specify the net resistance. For example,

```
dc_shell> set_resistance 0 find(net, wire_io4)
```

---

## Breaking a Feedback Loop

Follow these steps to break a feedback loop in your design:

1. Find the feedback loop in your design, using the `report_timing -loop` option.
2. Break the feedback loop by specifying the path as a false path.

---

## Analyzing Buffer Problems

Note:

This section uses the term *buffer* to indicate either a buffer or an inverter chain.

This section describes

- Understanding buffer insertion
- Debugging missing buffers
- Debugging extra buffers

- Debugging hanging buffers
- Debugging modified buffer networks

## Understanding Buffer Insertion

Design Compiler inserts buffers to correct maximum fanout load or maximum transition time violations. If Design Compiler does not insert buffers during optimization, the tool probably does not identify a violation. See “Setting Design Rule Constraints” in Chapter 7 for more information about the maximum fanout load and maximum transition time design rules.

Use the `report_constraint` command to get details on constraint violations.

Figure 9-1 shows a design containing the IV1 cell.

*Figure 9-1 Buffering Example*

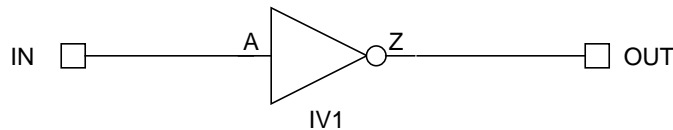


Table 9-5 gives the attributes defined in the technology library for the IV1 cell.

*Table 9-5 IV1 Library Attributes*

Pin	Attribute	Value
	<b>direction</b>	<b>input</b>
A	capacitance	1.5
	fanout_load	1

*Table 9-5 IV1 Library Attributes (continued)*

Pin	Attribute	Value
	<b>direction</b>	<b>output</b>
Z	rise_resistance	0.75
	fall_resistance	0.75
	max_fanout	3
	max_transition	2.5

Example 9-3 shows the result of the following command sequence:

```
dc_shell> set_drive 0 find(port,IN)
dc_shell> set_load 0 find(port,OUT)
dc_shell> report_constraint
```

### Example 9-3 Constraint Report

```
*****
Report : constraint
Design : buffer_example
Version: 1999.05
Date   : Mon Jan 4 10:56:49 1999
*****
```

Constraint	Cost
-----	-----
max_transition	0.00 (MET)
max_fanout	0.00 (MET)

To see the constraint cost functions used by Design Compiler, specify the `-verbose` option of the `report_constraint` command (shown in Example 9-4).

### Example 9-4 Constraint Report (-verbose)

```
*****
Report : constraint
        -verbose
Design : buffer_example
```

```

Version: 1999.05
Date   : Mon Jan 4 10:56:49 1999
*****

```

```

Net: OUT

```

```

max_transition          2.50
- Transition Time      0.00
-----
Slack                   2.50 (MET)

```

```

Net: OUT

```

```

max_fanout             3.00
- Fanout               0.00
-----
Slack                   3.00 (MET)

```

The verbose constraint report shows that two constraints are measured:

- Maximum transition time (2.50)
- Maximum fanout load (3.00)

Design Compiler derives the constraint values from the attribute values on the output pin of the IV1 cell.

When you compile this design, Design Compiler does not modify the design because the design meets the specified constraints.

To list all constraint violations, use the `-all_violators` option of the `report_constraint` command (shown in Example 9-5).

### *Example 9-5 Constraint Report (-all\_violators)*

```

*****
Report : constraint

```

```

    -all_violators
Design : buffer_example
Version: 1999.05
Date   : Mon Jan 4 10:56:49 1999
*****

```

This design has no violated constraints.

This design does not have any constraint violations. Changing the port attributes, however, can cause constraint violations to occur. Example 9-6 shows the result of the following command sequence:

```

dc_shell> set_drive 2.5 IN
dc_shell> set_max_fanout 0.75 IN
dc_shell> set_load 4 OUT
dc_shell> set_fanout_load 3.5 OUT
dc_shell> report_constraint -all_violators -verbose

```

### *Example 9-6 Constraint Report (After Modifying Port Attributes)*

```

*****
Report : constraint
        -all_violators
        -verbose
Design : buffer_example
Version: 1999.05
Date   : Mon Jan 4 10:56:49 1999
*****

Net: OUT

max_transition          2.50
- Transition Time      3.00
-----
Slack                   -0.50 (VIOLATED)

Net: OUT

```

max_fanout	3.00	
- Fanout	3.50	
-----		
Slack	-0.50	(VIOLATED)

Net: IN

max_fanout	0.75	
- Fanout	1.00	
-----		
Slack	-0.25	(VIOLATED)

This design now contains three violations:

- Maximum transition time violation at OUT  
Actual transition time is  $4.00 * 0.75 = 3.00$ , which is greater than the maximum transition time of 2.50.
- Maximum fanout load violation at OUT  
Actual fanout load of 3.5, which is greater than the maximum fanout load of 3.00.
- Maximum fanout load violation at IN  
Actual fanout load of 1.00, which is greater than the maximum fanout load of 0.75.

There is no `max_transition` violation at IN, even though the transition time on this net is  $2.5 * 1.5 = 3.75$ , which is well above the `max_transition` requirement of 2.50.

Design Compiler does not recognize this as a violation because the requirement of 2.50 is a design rule from the output pin of cell IV1. This requirement applies only to a net driven by this pin. The IV1 output pin does not drive the net connected to port IN, so the `max_transition` constraint does not apply to this net.

If you want to constrain the net attached to port IN to a maximum transition time of 2.50, enter the command

```
dc_shell> set_max_transition 2.5 find(port,IN)
```

This command causes `report_constraint -verbose -all_violators` to add the following lines to the report shown in Example 9-6:

```
Net: IN
max_transition          2.50
- Transition Time      3.75
-----
Slack                  -1.25 (VIOLATED)
```

When you compile this design, Design Compiler adds buffering to correct the `max_transition` violations.

Remember the following points when you work with buffers in Design Compiler:

- The `max_fanout` and `max_transition` constraints control buffering; be sure you understand how each is used.
- Design Compiler fixes only violations it detects.
- The `report_constraint` command identifies any violations.

## Debugging Missing Buffers

This is the most frequent buffering problem. It usually results from one of the following problems:

- Incorrectly specified constraints
- Improperly constrained designs
- Incorrect assumptions about constraint behavior

To debug the problem, generate a constraint report (`report_constraint`) to determine if Design Compiler recognized any violations.

If Design Compiler reports no `max_fanout` or `max_transition` violations, check the following:

- Are constraints applied?
- Is the library modeled for the correct attributes?
- Are the constraints tight enough?

If Design Compiler recognizes a violation but `compile` does not insert buffers to remove the violation, check the following:

- Does the violation exist after compile?
- Are there `dont_touch` or `dont_touch_network` attributes?
- Are there three-state pins that require buffering?
- Have you considered that `max_transition` takes precedence over `max_fanout`?

### **Incorrectly Specified Constraints**

A vendor might omit an attribute you want to use, such as `fanout_load`. If a vendor has not set this attribute in the library, the tool does not find any violations for the constraint. You can check whether the attribute exists by using the `get_attribute` command. For example, to determine if a pin has a `fanout_load` attribute, enter

```
dc_shell> get_attribute \  
          find(pin,library/cell/pin) fanout_load
```

The vendor might have defined `default_fanout_load` in the library. If this value is set to 0 or to an extremely small number, any pin that does not have an explicit `fanout_load` attribute inherits this value.

### Improperly Constrained Designs

Occasionally, a vendor uses extremely small capacitance values (on the order of 0.001). If your scripts do not take this into account, you might not be constraining your design tightly enough. Try setting an extreme value, such as 0.00001 and run `report_constraint` to make sure a violation occurs.

You can use the `load_of` command to check the capacitance values in the technology library.

```
dc_shell> load_of find(pin,library/cell/pin)
```

You can use the `get_attribute` command to check the fanout load values:

```
dc_shell> get_attribute \  
          find(pin,library/cell/pin) fanout_load
```

### Incorrect Assumptions About Constraint Behavior

Check to make sure you are not overlooking one of the following aspects of constraint behavior:

- A common mistake is the assumption that the `default_max_transition` or the `default_max_fanout` constraint in the technology library applies to input ports. These constraints apply only to the output pins of cells within the library.

- Maximum transition time takes precedence over maximum fanout load within Design Compiler. Therefore, a maximum fanout violation might not be corrected if the correction affects the maximum transition time of a net.
- Design Compiler might have removed the violation by sizing gates or modifying the structure of the design.

Generate a constraint report after optimization to verify that the violation still exists.

- Design Compiler cannot correct violations if `dont_touch` attributes exist on the violating path.

You might have inadvertently placed `dont_touch` attributes on a design or cell reference within the hierarchy. If so, Design Compiler reports violations but cannot correct them during optimization.

Use the `report_cell` command and the `get_attribute` command to see whether these attributes exist.

- Design Compiler cannot correct violations if `dont_touch_network` attributes exist on the violating path.

If you have set the `dont_touch_network` attribute on a port or pin in the design, all elements in the transitive fanout of that port or pin inherit the attribute. If this attribute is set, Design Compiler reports violations but does not modify the network during optimization.

Use the `remove_attribute` command to remove this attribute from the port or net.

- Design Compiler does not support additional buffering on three-state pins.

For simple three-state cells, Design Compiler attempts to enlarge the cell to a stronger three-state buffer.

For complex three-state cells, such as sequential elements or RAM cells, Design Compiler cannot build the logic necessary to duplicate its function. In these cases you must manually add the extra logic or rewrite the source HDL to decrease the fanout load of such nets.

## Debugging Extra Buffers

Extremely conservative numbers for `max_transition`, `max_fanout`, or `max_capacitance` force Design Compiler to buffer nets excessively. If your design has an excessive number of buffers, check the accuracy of the design rule constraints applied to the design.

If you have specified design rule constraints that are more restrictive than those specified in the technology library, evaluate the necessity for these restrictive design rules.

You can debug this type of problem by setting the priority of the maximum delay cost function higher than the maximum design rule cost functions (using the `set_cost_priority -delay` command). Changing the priority prevents Design Compiler from fixing the maximum design rule violations if the fix results in a timing violation.

## Debugging Hanging Buffers

A buffer that does not fan out to any cells is called a hanging buffer. Hanging buffers often occur because the buffer cells have `dont_touch` attributes. These attributes either can be set by you, in the hope of retaining a buffer network, or can be inherited from a library.

The `dont_touch` attribute on a cell signals to Design Compiler that the cell should not be touched during optimization. Design Compiler follows these instructions by leaving the cell in the design. But because the buffer might not be needed to meet the constraints that are set, Design Compiler disconnects the net from the output. The design meets your constraints, but because the cell has the `dont_touch` attribute, the cell cannot be removed. Remove the `dont_touch` attribute to correct this problem.

## Debugging Modified Buffer Networks

Sometimes it appears that Design Compiler modifies a buffer network that has `dont_touch` attributes. This problem usually occurs when you place the `dont_touch` attribute on a cell and expect the cells adjacent to that cell to remain in the design.

Design Compiler does not affect the cell itself but modifies the surrounding nets and cells to attain the optimal structure. If you are confident about the structure you want, you can use one of the following strategies to preserve your buffer network:

- Group the cells into a new hierarchy and set `dont_touch` attributes on that hierarchy.
- Set the `dont_touch_network` attribute on the pin that begins the network.

- Set the `dont_touch` attribute on all cells and nets within the network that you want to retain.