Command Reference for Encounter® RTL Compiler

Product Version 9.1
July 2009

Contents

Alphabetical List of Commands

Preface

About This Manual

This manual provides a concise reference of the commands available to the user when using Encounter™ RTL Compiler. This manual describes each command available within the RTL Compiler shell with their command options.

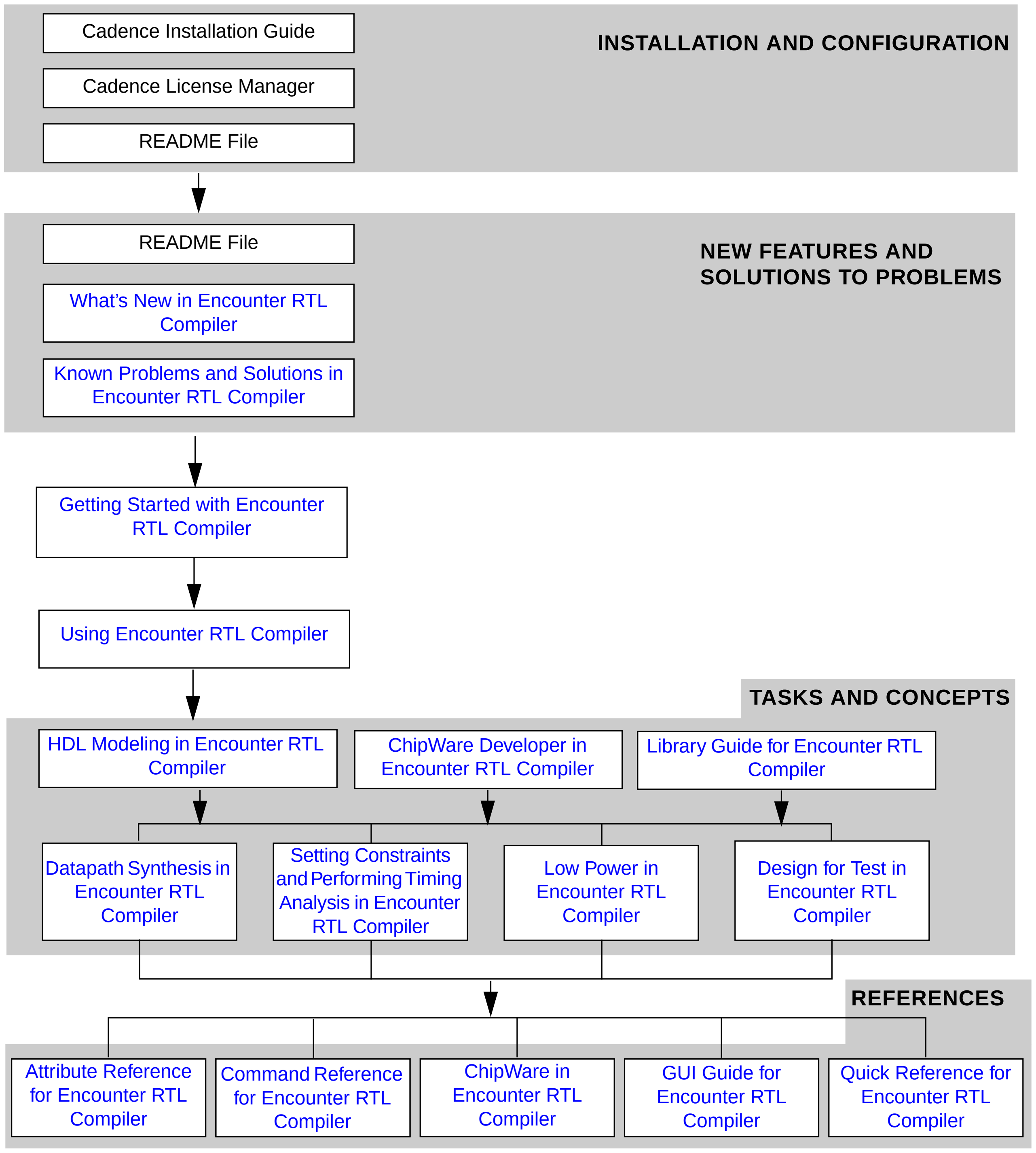
Additional References

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

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

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

How to Use the Documentation Set


Cadence Installation Guide
Cadence License Manager
README File
INSTALLATION AND CONFIGURATION
README File
What's New in Encounter RTL Compiler
Known Problems and Solutions in Encounter RTL Compiler
NEW FEATURES AND SOLUTIONS TO PROBLEMS
Getting Started with Encounter RTL Compiler
Using Encounter RTL Compiler
TASKS AND CONCEPTS
HDL Modeling in Encounter RTL Compiler
ChipWare Developer in Encounter RTL Compiler
Library Guide for Encounter RTL Compiler
Datapath Synthesis in Encounter RTL Compiler
Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler
Low Power in Encounter RTL Compiler
Design for Test in Encounter RTL Compiler
REFERENCES
Attribute Reference for Encounter RTL Compiler
Command Reference for Encounter RTL Compiler
ChipWare in Encounter RTL Compiler
GUI Guide for Encounter RTL Compiler
Quick Reference for Encounter RTL Compiler

Reporting Problems or Errors in Manuals

The Cadence® Help online documentation, lets you view, search, and print Cadence product documentation. You can access Cadence Help by typing `cdnshelp` from your Cadence tools hierarchy.

Contact Cadence Customer Support to file a CCR if you find:

- An error in the manual
- An omission of information in a manual
- A problem using the Cadence Help documentation system

Customer Support

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

Cadence Online Support

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

For more information on Cadence online support go to:

http://support.cadence.com

Other Support Offerings

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

For more information on these support offerings go to:

http://www.cadence.com/support

Messages

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

- Use the `report messages` command.

  For example:

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

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

- Use the `man` command.

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

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

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

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

rc_msg_improvement@cadence.com

Man Pages

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

To use our man pages from the UNIX shell:

1. Set your environment to view the correct directory:

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

2. Enter the name of the command or attribute that you want. For example:

   - `man check_dft_rules`
   - `man cell_leakage_power`

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

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

Command-Line Help

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

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

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

rc_pubs@cadence.com

Getting the Syntax for a Command

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

For example:

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

This returns the syntax for the `path_delay` command.

Getting the Syntax for an Attribute

Type the following:

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

For example:

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

This returns the syntax for the `max_transition` attribute.

Searching for Attributes

You can get a list of all the available attributes by typing the following command:

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

You can type a sequence of letters after the `set_attribute` command and press `Tab` to get a list of all attributes that contain those letters.

```
rc:/> set_attr li
```

```
ambiguous "li": lib_lef_consistency_check_enable lib_search_path libcell
liberty_attributes libpin library library_domain line_number
```

Searching For Commands When You Are Unsure of the Name

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

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

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

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

- You can type a single letter and press `Tab` to get a list of all commands that contains that letter.

  For example:

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

  This returns the following commands:

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

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

  For example:

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

  This returns the following commands:

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

Documentation Conventions

To aid the readers understanding a consistent formatting style has been used throughout this manual.

- UNIX commands are shown following the `unix>` string.
- RTL Compiler commands are shown following the `rc:/>` string.

Text Command Syntax

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

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

1

Navigation

basename

`basename` *`pathname`*

Removes the leading directory names of the specified path name and returns only the object name. This command behaves similarly to the UNIX `basename` command.

Options and Arguments

`pathname`	Specifies the path name of the object, including the object name.

Examples

- The following example removes the directory name of the `CW_absval` ChipWare component and only returns the component name:

```
rc:/> basename \
    /hdl_libraries/CW/components/CW_absval/comp_architectures/CW_absval
CW_absval
```

- The following example uses the `basename` command with the `dirname` command to return the name of the library to which the `ND2X1` library cell belongs:

```
rc:/> set libcell /libraries/LIB/libcells/ND2X1
rc:/> basename [dirname [dirname $libcell]]
```

Related Information

Related commands:	dirname on page 35

cd

```
cd [ directory ]
```

Sets the current directory in the design hierarchy and navigates the design hierarchy. This command is similar to its UNIX counterpart. A description of the design hierarchy is given in the *Using Encounter RTL Compiler*.

Options and Arguments

directory	Specifies the name of the directory to be set as the current directory. The ".", "..", and "/"have the same meaning as their UNIX counterparts (current directory, parent directory, and root directory respectively). You can use wildcards when they do not produce an ambiguous reference (more than one match).

Examples

- The following command returns you to the top of the design hierarchy:

```
rc:/designs> cd
rc:/> pwd
/
```

- The following command specifies the absolute path to the target directory:

```
rc:/> cd /designs/alu/timing/exceptions
```

- The following command specifies the relative path to the target directory:

```
rc:/> cd designs/alu
rc:/designs/alu> cd timing/exceptions
```

- The following command changes the current directory to a parent directory:

```
rc:/designs/alu/timing/exceptions cd ../../subdesigns
rc:/designs/alu/subdesigns> pwd
/designs/alu/subdesigns
```

- The following command uses wildcards in the path specification:

```
rc:/designs/cmplx_alu/subdesigns/addinc> cd ../../tim*/exc*
rc:/designs/cmplx_alu/timing/exceptions> pwd
/designs/cmplx_alu/timing/exceptions
```

Related Information

Affects these commands:	dirs on page 36
	ls on page 46
	popd on page 50
	pushd on page 51
	pwd on page 52

dirname

```
dirname pathname [-times integer]
```

Removes the object name of the specified path name and only returns the directory name. This command behaves similarly to the UNIX `dirname` command.

Options and Arguments

`pathname`	Specifies the path name of the object, including the object name.
`-times` *`integer`*	Specifies the number of times to apply dirname. *Default:* `1`

Examples

■ The following example removes the `CW_absval` ChipWare component name and only returns its directory name:

```
rc:/>dirname \
    /hdl_libraries/CW/components/CW_absval/comp_architectures/CW_absval
/hdl_libraries/CW/components/CW_absval/comp_architectures
```

■ The following command applies the `dirname` command 3 times to the specified path.

```
rc:/> dirname -times 3 \
/hdl_libraries/CW/components/CW_absval/comp_architectures/CW_absval
/hdl_libraries/CW/components
```

■ The following example uses the `basename` command with the `dirname` command to return the name of the library to which the `ND2X1` library cell belongs:

```
rc:/> set libcell /libraries/LIB/libcells/ND2X1
rc:/> basename [dirname [dirname $libcell]]
```

Related Information

Related commands:	basename on page 32

dirs

```
dirs
```

Displays the contents of the design directory stack. This command is similar to its UNIX counterpart and is used in conjunction with the pushd and popd commands.

Example

- The following commands respectively add designs and libraries to the design directory stack, then display the contents of the directory stack:

  ```
  rc:/> pushd designs
  /designs /

  rc:/designs> pushd /libraries
  /libraries /designs /

  rc:/libraries> dirs
  /libraries /designs /
  ```

- The following commands respectively remove the last added directories and then display the contents of the directory stack:

  ```
  rc:/libraries> popd
  /designs /

  rc:/designs> popd
  /

  rc:/> dirs
  /
  ```

Related Information

Related commands:	ls on page 46
	popd on page 50
	pushd on page 51
	pwd on page 52

filter

```
filter [-invert] [-regexp] attribute_name
     attribute_value ... [object_list...]
```

Filters a set of objects based on the values of the given attributes using the pattern matching mechanism from the `glob` Tcl command.

This command provides a powerful means of selecting objects within the design hierarchy at a more discrete level than is allowed by the directory structure alone.

Use the `get_attribute` command to list the attributes available for each object type.

Options and Arguments

`attribute_name`	Specifies the name of an attribute to use as filter. This argument is required. A compound string (containing spaces) should be represented as a list either by using double-quotes or braces ({ }).
`attribute_value`	Specifies the value of an attribute to use as filter.
`-invert`	Filters out objects that match the expression and returns those that do not.
`object_list`	Specifies a Tcl list of objects to filter.
`-regexp`	Overrides the default Tcl glob pattern matching with Tcl regular expression matching.

Examples

- The following command finds the list of all matching library cells on which the `preserve` attribute has been set to true:

  ```
  rc:/> filter preserve true [find . -libcell *]
  ```

- The following command stores the Tcl list of all matching cells returned by the `filter` command in a variable for use in scripting later.

  ```
  rc:/> set preserved_cells [filter preserve true [find . -libcell *]]
  ```

- The following command embeds the Tcl list of all matching cells returned by the filter command as part of a larger command.

  ```
  rc:/> report timing -through [filter preserve true [find . -libcell *]]
  ```

- The following Tcl code fragment sets the variable `result` to all instances that start with the letter g and whose corresponding library cell starts with `inv`:

  ```
  set result {}
  foreach inst [find . -inst g*] {
    if {[string match "inv*" [get_att libcell $inst]]}
       {lappend result $inst}
  }
  puts $result
  ```

- The following example returns only returns those pins that have the `preserve` attribute set to either `true` or `false` and ignores those with `size_ok` values:

  ```
  rc:/> filter -regexp preserve {true|false} [find / -pin *]
  ```

- Use the `ls -dir` command to format the output:

  ```
  rc:/> ls -dir [filter preserve true [find . -libcell *]]
  /libraries/penny/libcells/ANTENNA/
  /libraries/penny/libcells/FILL1/
  /libraries/penny/libcells/RF1R1WX2/
  /libraries/penny/libcells/RF2R1WX2/
  ```

- Use the ls -dir command and the redirect arrow to redirect the output to the specified file:

  ```
  rc:/> ls -dir [filter preserve true [find . -libcell *]] > areid.txt
  ```

  You can also append arrows (">>").

Related Information

Affects these commands:	ls on page 46
	get_attribute on page 68
	set_attribute on page 98

find

```
find [root_path] [-maxdepth integer] [-mindepth integer]
     [-ignorecase] [-regexp <expression>] [-vname]
     {-option...|-*} object
```

Searches the design hierarchy for the specified types of objects and returns a Tcl list containing the full paths to any matching objects. This list can then be used by other commands to operate on groups of objects.

The `find` command supports the `*` and `?` wildcard characters.

Note: The `find` command is very powerful but overusing it can increase the execution time. To reduce the execution time, be as specific as possible when specifying the object to match.

Options and Arguments

`-ignorecase`	Ignores the case (upper case or lower case) of the parameters. Alternatively, specifies the search to be case insensitive.
`-maxdepth` *`level`*	Descends no more than the specified number (non-negative integer) of levels below *`root_path`*. A level of `0` searches only the `root_path`. *Default*: infinity
`-mindepth` *`integer`*	Skips the specified number (non-negative integer) of levels below *`root_path`* before finding objects. A level of `1` searches all objects except *`root_path`*. *Default*: `0`
`object`	Specifies the name of the object to match. The name can include wildcard characters.
-*`option`*	Specifies the type of object you want to find. You can specify multiple options or look in all types by specifying `*`. Option can have one of the following values:

actual_scan_chain	hdl_param	operating_condition
actual_scan_segment	hdl_pin	pin
attribute	hdl_proc	pin_bus
clock	hdl_subp	port
clock_domain	instance	port_bus

constant	isolation_rule	power_domain
cost_group	jtag_instruction	power_ground_net
design	jtag_instruction_register	root
exception	jtag_port	scan_chain
external_delay	level_shifter_group	scan_segment
hdl_arch	level_shifter_rule	seq_function
hdl_bind	libarc	subdesign
hdl_block	libcell	subport
hdl_comp	libpin	subport_bus
hdl_config	library	test_clock
hdl_impl	library_domain	test_clock_domain
hdl_inst	message	test_signal
hdl_label	message_group	violation
hdl_lib	mode	wireload
hdl_oper	net	wireload_selection
hdl_pack	object_type	

Note: The `clock_domain`, `test_clock_domain`, and `wireload_selection` object types currently do not have any attributes associated with them.

`-regexp` Specifies regular expression.

`root_path` Specifies the name of the directory from where to start searching. The name can include wildcard characters.

By default, an explicit search is done for the specified object. During an explicit search every object directory is searched, instead of starting from *`root_path`*. Specifying a *`root_path`* reduces the numbers of locations where the `find` command will search which reduces the execution time.

`-vname` Removes the container directories in the pathname and returns Verilog style names where appropriate. This option will only work on the following objects: `pin`, `port`, `net`, `subdesign`, and `instance`.

Examples

- The following command searches for any object type whose name contains `add`, starting from the current directory (`/designs`):

  ```
  rc:/designs> find . * *add*
  ```

  ```
  /designs/alu/instances_hier/ops1_add_25 /designs/alu/subdesigns/addinc64
  ```

- The following command finds all registers in all designs:

  ```
  rc:/> find des* -instance *seq/*
  ```

  ```
  /designs/alu/instances_hier/RC_CG_HIER_INST_0/instances_seq/RC_CGIC_INST
  /designs/alu/instances_seq/aluout_reg_7
  /designs/alu/instances_seq/aluout_reg_6
  /designs/alu/instances_seq/aluout_reg_4
  /designs/alu/instances_seq/aluout_reg_0
  /designs/alu/instances_seq/aluout_reg_3
  /designs/alu/instances_seq/aluout_reg_5
  /designs/alu/instances_seq/aluout_reg_2
  /designs/alu/instances_seq/aluout_reg_1
  /designs/alu/instances_seq/zero_reg
  ```

- The following command finds all input ports in design `alu`:

  ```
  rc:/> find des*/alu -port ports_in/*
  ```

  ```
  {/designs/alu/ports_in/opcode[2]} {/designs/alu/ports_in/opcode[1]}
  {/designs/alu/ports_in/opcode[0]} {/designs/alu/ports_in/data[7]}
  {/designs/alu/ports_in/data[6]} {/designs/alu/ports_in/data[5]}
  {/designs/alu/ports_in/data[4]} {/designs/alu/ports_in/data[3]}
  {/designs/alu/ports_in/data[2]} {/designs/alu/ports_in/data[1]}
  {/designs/alu/ports_in/data[0]} {/designs/alu/ports_in/accum[7]}
  {/designs/alu/ports_in/accum[6]} {/designs/alu/ports_in/accum[5]}
  {/designs/alu/ports_in/accum[4]} {/designs/alu/ports_in/accum[3]}
  {/designs/alu/ports_in/accum[2]} {/designs/alu/ports_in/accum[1]}
  {/designs/alu/ports_in/accum[0]} /designs/alu/ports_in/clock
  /designs/alu/ports_in/ena /designs/alu/ports_in/reset
  ```

- The following command searches for an external delay whose name starts with `in`, starting from the `designs` directory:

  ```
  rc:/designs> find designs -external_delay in*
  ```

  ```
  /designs/alu/timing/external_delays/in_del_1
  ```

- The following command finds all designs that are four characters:

  ```
  rc:/> find . -designs ????
  ```

  ```
  /designs/test
  ```

- The following command finds all design names with four characters that end with the letter "i":

  ```
  rc:/> find . -designs ???i
  ```

  ```
  /designs/topi
  ```

- The following command performs a case insensitive search for the design TEST:

  ```
  rc:/> find . -ignorecase -design test
  ```

```
/designs/TEST
```

- To find hierarchical objects, you can just specify the top-level object instead of the root or current directory. Doing so can provide faster results because it minimizes the number of hierarchies that RTL Compiler traverses. In the following example, if we wanted to only find the output pins for `inst1`, the first specification is more efficient than the second. The second example not only traverses more hierarchies, it also returns `inst2` instances.

```
rc:/> find inst1 -pin out*
{/designs/woodward/instances_hier/inst1/pins_out/out1[3]}
rc:/>find / -pin out*
{/designs/woodward/instances_hier/inst1/pins_out/out1[3]}
{/designs/woodward/instances_hier/inst2/pins_out/out1[3]}
```

- The following example uses the `find` command to return a list of all the instances in a small design.

```
rc:/> find / -instance *
/designs/MOD69/instances_hier/inst1
/designs/MOD69/instances_hier/inst1/instances_comb/g21
```

  The `-vname` option removes the container directories (in this case `instance_hier`) and presents the list more concisely:

```
rc:/> find / -instance -vname *
inst1
inst1/g21
```

  This option is useful when you want to present the object in a report because the name is more concise. The disadvantage of the shortened name is that it may no longer refer to a unique object because an instance, pin, net, and subport may all share the same Verilog name.

- The following example uses the `-regexp` option to return all message objects that contain at least `VLOGPT-6`:

```
rc:/> find / -regexp (VLOGPT-6+) -messsages * *
```

  The following example returns all combinational instances named `g58` or `g59` in the Verilog name style:

```
rc:/> find / -regexp {g[5][8-9]} -vname -instance instances_comb/*
```

- Use the `ls -dir` command to format the output row over row:

```
rc:/> ls -dir [find / -instance -vname *]
/designs/quea/instances_hier/inst1/
/designs/quea/instances_hier/inst1/instances_comb/g41/
/designs/quea/instances_hier/inst1/instances_comb/g42/
/designs/quea/instances_hier/inst1/instances_comb/g43/
/designs/quea/instances_hier/inst1/instances_comb/g44/
```

- Use the ls -dir command and the redirect arrow to redirect the output to the specified file:

  ```
  rc:/> ls -dir [find / -instance -vname *] > quea.txt
  ```

- You can also append arrows (">>").

Related Information

Related command:	clock_ports on page 321

inout_mate

`inout_mate {`*`subport_bus`*`|`*`port_bus`*`|`*`subport`*`|`*`port`*`|`*`pin`*`}`

Distinguishes bidirectional pins, ports, port_busses, subports, and subport_busses inout objects so you can find the input object and the corresponding output object.

These bidirectional inout objects are represented as two distinct objects: the input pin, port, port_bus, subport, and subport_bus object and the corresponding output object.The pair cannot be broken apart, such as using an `edit_netlist rm` command to delete just one of the two objects, because together, they represent the inout object.

Examples

- The following example is a design called `top` that has a primary inout port named `io`, which RC Encounter represents as two distinct ports:

  ```
  /designs/top/ports_in/io
  /designs/top/ports_out/io
  ```

  - Use the `inout_mate` command on the input object to find the corresponding output object:

    ```
    rc:/> inout_mate ports_in/io
    /designs/top/ports_out/io
    ```

  - Use the `inout_mate` command on the output object to find the corresponding input object

    ```
    rc:/> inout_mate ports_out/io
    /designs/top/ports_in/io
    ```

- The following example is an hierarchical instance called `s4` that has an inout port named `io`, which RC Encounter represents as two distinct subports:

  ```
  /designs/top/instances_hier/s4/subports_in/io
  /designs/top/instances_hier/s4/subports_out/io
  ```

  - Use the `inout_mate` command on the input object to find the corresponding output object:

    ```
    rc:/> inout_mate s4/subports_in/io
    /designs/top/instances_hier/s4/subports_out/io
    ```

  - Use the `inout_mate` command on the output object to find the corresponding input object:

    ```
    rc:/> inout_mate s4/subports_out/io
    /designs/top/instances_hier/s4/subports_in/io
    ```

ll

`ll` *`virtual_dir`*

Lists the contents of the specified virtual directory in long format

A virtual directory is a directory in the design hierarchy.

Options and Arguments

`virtual_dir`	Specifies the virtual directory whose contents to list. If no directory is specified, the contents of the current directory in the design hierarchy is listed.

Examples

- The following example lists the contents of the `counter` (design) directory.

```
rc:/> ll counter
/designs/counter:
Total: 17 items
./                       (design)
constants/
dex_settings/
dft/
instances_comb/
instances_hier/
instances_seq/
modes/
nets/
physical/
port_busses_in/
port_busses_out/
ports_in/
ports_out/
power/
subdesigns/
timing/
rc:/>
```

Related Information

Affected by this command:	cd on page 33
Related command:	pwd on page 52

ls

```
ls [-computed] [-attribute] [-long] [-dir]
    [-width integer] [object]... [>file]
```

Lists information about any objects in the design hierarchy (designs, library cells, clocks, and so on). This command is similar to its UNIX counterpart.

Options and Arguments

`-attribute`	List the attributes for the specified object whose values are different from the default values.
`-computed`	Lists all computed attributes. Computed attributes are potentially very time consuming to process and are therefore by default not listed.
`-dir`	Lists only the directory name not its contents.
`file`	Specifies the name of the file to which to list the information.
`-long`	Lists the contents (long listing) of the directory.
`object`	Specifies the directory for which you want to list information. *Default*: current directory
`-width` *`integer`*	Specifies the width of the screen that can be used to show the information *Default*: `80`

Examples

- The following command lists the contents of the `libraries` directory:

```
rc:/> ls libraries
/libraries:
./                em333s/           sm333s/
```

- The following command shows information of cell `buf1` in regular and long listing format:

```
rc:/libraries/tutorial/libcells/buf1> ls
./           A            Y
```

```
rc:/libraries/tutorial/libcells/buf1> ls -long
Total: 3 items
./                (libcell)
A                 (libpin)
Y                 (libpin)
```

- The following command lists only the attributes of `buf1` whose values are different from the default values:

```
rc:/libraries/tutorial/libcells/buf1> ls -attribute
Total: 3 items
./                 (libcell)
    Attributes:
      area = 1
      buffer = true
      cell_leakage_power = 0.0 nW
      combinational = true
      liberty_attributes = area 1.0
A/                 (libpin)
    Attributes:
      capacitance = 25.0 25.0 femtofarads
      fanout_load = 1.000 fanout_load units
      input = true
      liberty_attributes = capacitance 0.025 direction input
      max_transition = 4500.0
      outgoing_timing_arcs = /libraries/tutorial/libcells/buf1/Y/inarcs/A_n90
Y/                 (libpin)
    Attributes:
      capacitance = 0.0 0.0 femtofarads
      fanout_load = 0.000 fanout_load units
      function = A
      incoming_timing_arcs = /libraries/tutorial/libcells/buf1/Y/inarcs/A_n90
      liberty_attributes = direction output function A
      max_transition = 4500.0
      output = true
```

- The following command lists all attributes (except for the computed attributes) for `buf1`, even those with the default value:

```
rc:/libraries/tutorial/libcells/buf1> ls -long -attribute
Total: 3 items
./                 (libcell)
    All attributes:
      adder = false
      area = 1.0
      async_clear =
      async_preset =
      avoid = false
      buffer = true
      cell_delay_multiplier = 1.0
      cell_leakage_power = 0.0 nW
      clock =
      clock_gating_integrated_cell =
      combinational = true
      constraint_multiplier = 1.0
      flop = false
      height = no_value
      ...
      usable = true
      user_defined =
      width = no_value
A                  (libpin)
    All attributes:
      async_clear_phase = none
      async_preset_phase = none
      capacitance = 25.0 25.0 femtofarads
```

```
          clock_gate_clock_pin = false
          clock_gate_enable_pin = false
          clock_gate_obs_pin = false
          clock_gate_out_pin = false
          clock_gate_reset_pin = false
          clock_gate_test_pin = false
          ...
          tristate = false
          user_defined =
    Y                  (libpin)
        All attributes:
          async_clear_phase = none
          async_preset_phase = none
          capacitance = 0.0 0.0 femtofarads
          clock_gate_clock_pin = false
          clock_gate_enable_pin = false
          clock_gate_obs_pin = false
          clock_gate_out_pin = false
          clock_gate_reset_pin = false
          clock_gate_test_pin = false
          ...
          tristate = false
          user_defined =
```

- The following command uses wildcard strings and the results of a `find` command:

```
rc:/libraries> ls -long [find . -libcell A*]
/libraries/cg/libcells/AND2A:
Total: 4 items
./                 (libcell)
A/                 (libpin)
B/                 (libpin)
Z/                 (libpin)

/libraries/cg/libcells/AO21A:
Total: 5 items
./                 (libcell)
A/                 (libpin)
B/                 (libpin)
C/                 (libpin)
Z/                 (libpin)
```

- The following command shows that the computed attribute `timing_case_computed_value` has been turned on:

```
rc:/>ls -computed /designs/violet/instances_comb/U1/pins_in/S
timing_case_computed_value = 1
```

- The following examples show the difference between the `ls -attribute` and `get_attribute` commands.

```
rc:/designs> ls -attribute
Total: 2 items
./
async_set_reset_flop_n/                 (design)
    Attributes:
      dft_mix_clock_edges_in_scan_chains = false
      wireload = /libraries/slow/wireload_models/sartre18_Conservative

rc:/designs> get_attribute wireload /designs/async_set_reset_flop_n/
/libraries/slow/wireload_models/sartre18_Conservative
```

The `ls -attribute` command lists all *user-modified* attributes and their values. The `get_attribute` command lists only the value of the specified attribute. The `get_attribute` command is especially useful in scripts where its returned values can be used as arguments to other commands.

Related Information

Related command:	get_attribute on page 68

popd

`popd`

Removes the topmost element of the directory stack, revealing a new top element and changes the current directory to the new top element. This command is similar to its UNIX counterpart.

Note: If `popd` is issued on a directory stack that has only one element, an appropriate warning message is printed and the `popd` command exits without changing the directory stack.

Examples

- In the following example, the directory stack starts off as `/libraries /designs:` `/libraries` is the current directory and the top element of the directory stack, and `/designs` is next on the stack). When the `popd` command is issued, `/libraries` is popped off, and `/designs` becomes the top (and only element) of the stack and the current directory.

  ```
  rc:/libraries> dirs
  /libraries /designs

  rc:/libraries> popd
  /designs

  rc:/designs> dirs
  /designs

  rc:/designs> pwd
  /designs
  ```

- In the following example, a `popd` command is issued on a directory stack that has only one element.

  ```
  rc:/> cd /designs

  rc:/designs> dirs
  /designs

  rc:/designs> popd
  Directory stack empty
  /designs
  ```

  This can happen when more `popd` commands are issued than `pushd` commands.

Related Information

Affects these commands:	dirs on page 36
	ls on page 46
	pushd on page 51
	pwd on page 52

pushd

`pushd` *`directory`*

Pushes the specified new target directory onto the directory stack (as the topmost element) and changes the current directory to that specified directory. This command is similar to its UNIX counterpart.

Options and Arguments

`directory`	Specifies the name of the directory to be set as target directory. You can use wildcards when they do not produce an ambiguous reference (more than one match).

Examples

- In the following example, `/libraries` is pushed onto the top of the `/designs` directory stack and the current directory is changed to it:

```
rc:/designs> pushd /libraries
/libraries /designs
rc:/libraries>
```

- In the following example, the push operation does not succeed because there is more than one directory that starts with `ex`.

```
rc:/libraries> pushd ex*
Error   : A single object was expected, but multiple objects were found.[TUI-62]
        : The argument that found multiple objects was ’ex*’.
  pushd: push current dir onto stack and cd to new dir
Usage: pushd <object>
    <object>:
        new target directory
Failed on pushd ex*
```

Related Information

Affects these commands:	dirs on page 36
	ls on page 46
	popd on page 50
	pwd on page 52

pwd

```
pwd
```

Displays the current position in the design hierarchy. This command is similar to its UNIX counterpart.

Examples

- The following example shows the current directory after changing the current directory first:

  ```
  rc:/> cd /designs
  rc:/designs> pwd
  /designs
  ```

Related Information

Related commands:	dirs on page 36
	ls on page 46
	popd on page 50
	pushd on page 51

vdir_lsearch

`vdir_lsearch` *`list object`*

Performs an lsearch of a *vdir* type object in a list of objects. If a match is found, it returns the position of that object in the list. If no match is found, it returns `-1`.

While performing the search, the command performs a (fast) direct comparison of every vdir type object in the list with the pointer of the given object; whereas for every non-vdir type object in the list, it creates a string-name of the object and compares it with the string-name of the given object. This makes it faster than the regular lsearch command, which does string-name derivation and string comparison for every object in the list.

Note: Examples of commands that return vdir type objects are `find`, `edit_netlist` and `get_attribute`.

Options and Arguments

`list`	Specifies a list of objects. The objects can be regular objects or can be of type vdir.
`object`	Specifies the object for which you want to know the position in the list. The object must be of type vdir.

Example

In the following example, the first search, performed on a list of vdir type objects (result of `find` command), returns `-1`. To this list, two regular objects are added: `abc` and `/designs/test/instances_hier/mux_oo_5_10/pins_out/z`. The search on this mixed list of objects returns 4.

```
rc:/> set pin [get_attr driver /designs/test/nets/oo]
/designs/test/instances_hier/mux_oo_5_10/pins_out/z
rc:/> set list [find / -pin in*]
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_0
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_1
/designs/test/instances_comb/g4/pins_in/in_0
rc:/> vdir_lsearch $list $pin
-1
rc:/> lappend list "abc"
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_0
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_1
/designs/test/instances_comb/g4/pins_in/in_0 abc
rc:/> lappend list "/designs/test/instances_hier/mux_oo_5_10/pins_out/z"
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_0
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_1
/designs/test/instances_comb/g4/pins_in/in_0 abc
/designs/test/instances_hier/mux_oo_5_10/pins_out/z
rc:/> vdir_lsearch $list $pin
4
```

what_is

`what_is` *`object`*

Returns a string describing the type of object given as its argument. The command will work only if a single object is specified. A list of valid objects can be obtain by typing `find -help`.

This command is mostly used for writing Tcl scripts (rather than being an interactive command). It is useful for checking the types of arguments to the Tcl procedures.

Options and Arguments

`object`	Specifies the object for which you want to know the type.

Examples

- The following example returns pin:

  ```
  what_is /designs/TOP/instances_hier/SUB/pins_in/A[0]
  pin
  ```

- In the following example, the top-level design has two clocks `clock2` and `clock3`. The following command returns the type of `clock2`.

  ```
  rc:/> what_is clock2
  clock
  ```

- The following command fails because there is more than one object of the given name in the current tree structure.

  ```
  rc:/designs/alu> what_is alu*
  Error : A single object was expected, but multiple objects were found. [TUI-62]
        : The argument that found multiple objects was 'alu*'.
    what_is: return an object's type
  Usage: what_is <object>
      <object>:
          drs object of interest
  Failed on what_is alu*
  ```

2

General

?

`? [`*`command`*`]...[> `*`file`*`]`

Provides help on the specified RTL Compiler commands.

Note: You can get help at any stage of the design.

Options and Arguments

`command`	Specifies the command for which you want help. If you do not specify a command name, you get a brief summary of all RTL Compiler commands.
`file`	Specifies the name of the file to which to write the help.

Examples

- The following examp[le requests help for the

```
rc:/> ? all_inputs
That command is:

    General
==============================================================================
all_inputs  returns all the input ports.

Command details:

  all_inputs: returns all the input ports.

Usage: all_inputs [-design <design>]

    [-design <design>]:
        limits list of input ports to the specified top-level design
```

- The following example requests help for the `synthesize` and `report` commands:

```
rc:/> ? synthesize report
Commands are:

    Analysis
==============================================================================
report      generates one of various reports

    Synthesis
==============================================================================
synthesize  synthesizes the design
```

alias

`alias` *`alias_name command_name`*

Defines an alias for the specified name.

Options and Arguments

`alias_name`	Specifies the alias name.
`command_name`	Specifies the name of the command for which you want to create an alias.

Example

The following command creates an alias ai for the all_inputs command.

```
rc:/> alias ai all_inputs
ai
rc:/> ai -h
  all_inputs: returns all the input ports.

Usage: all_inputs [-design <design>]

    [-design <design>]:
        limits list of input ports to the specified top-level design
```

all_inputs

`all_inputs [-design` *`design`*`]`

Returns the input ports of the specified design.

Options and Arguments

`-design` *`design`*	Specifies the name of the top-level design for which you want to list all input ports. If you omit the design name, the input ports of all loaded designs are listed.

Examples

■ The following example lists the input ports of all loaded designs.

```
rc:/designs/top> all_inputs
```

```
/designs/top/ports_in/enable {/designs/top/ports_in/in1[7]}
{/designs/top/ports_in/in1[6]} {/designs/top/ports_in/in1[5]}
{/designs/top/ports_in/in1[4]} {/designs/top/ports_in/in1[3]}
{/designs/top/ports_in/in1[2]} {/designs/top/ports_in/in1[1]}
{/designs/top/ports_in/in1[0]} {/designs/top/ports_in/in2[7]}
{/designs/top/ports_in/in2[6]} {/designs/top/ports_in/in2[5]}
{/designs/top/ports_in/in2[4]} {/designs/top/ports_in/in2[3]}
{/designs/top/ports_in/in2[2]} {/designs/top/ports_in/in2[1]}
{/designs/top/ports_in/in2[0]} /designs/top/ports_in/clk
/designs/my_CG_MOD/ports_in/ck_in /designs/my_CG_MOD/ports_in/enable
/designs/my_CG_MOD/ports_in/test /designs/my_CG_MOD_neg/ports_in/ck_in
/designs/my_CG_MOD_neg/ports_in/enable /designs/my_CG_MOD_neg/ports_in/test
```

■ The following example lists the input ports of design `my_CG_MOD`.

```
rc:/designs/top> all_inputs -design my_CG_MOD
```

```
/designs/my_CG_MOD/ports_in/ck_in /designs/my_CG_MOD/ports_in/enable /
designs/my_CG_MOD/ports_in/test
```

■ Use the `ls -dir` command to format the output of the command

```
rc:/> ls -dir [all_inputs]
```

```
/designs/ksable/ports_in/in1[0]
/designs/ksable/ports_in/in1[1]
/designs/ksable/ports_in/in1[2]
```

■ Use the `ls -dir` command with the redirect arrow to redirect the output to a specified file:

```
rc:/> ls -dir [all_inputs] > areid.txt
```

You can also append arrows (">>").

all_outputs

`all_outputs [-design` *`design`*`]`

Returns the output ports of the specified design.

Options and Arguments

`-design` *`design`*	Specifies the name of the top-level design for which you want to list all output ports. If you omit the design name, the output ports of all loaded designs are listed.

Examples

- The following example lists the output ports of all loaded designs.

  ```
  rc:/designs/top> all_outputs
  ```

  ```
  {/designs/top/ports_out/out1[7]} {/designs/top/ports_out/out1[6]}
  {/designs/top/ports_out/out1[5]} {/designs/top/ports_out/out1[4]}
  {/designs/top/ports_out/out1[3]} {/designs/top/ports_out/out1[2]}
  {/designs/top/ports_out/out1[1]} {/designs/top/ports_out/out1[0]}
  {/designs/top/ports_out/out2[7]} {/designs/top/ports_out/out2[6]}
  {/designs/top/ports_out/out2[5]} {/designs/top/ports_out/out2[4]}
  {/designs/top/ports_out/out2[3]} {/designs/top/ports_out/out2[2]}
  {/designs/top/ports_out/out2[1]} {/designs/top/ports_out/out2[0]}
  /designs/my_CG_MOD/ports_out/ck_out /designs/my_CG_MOD_neg/ports_out/ck_out
  ```

- The following example lists the output ports of design `my_CG_MOD`.

  ```
  rc:/designs/top> all_outputs -design my_CG_MOD
  /designs/my_CG_MOD/ports_out/ck_out
  ```

- Use the `ls -dir` command to format the output of the command

  ```
  rc:/> ls -dir [all_inputs]
  /designs/ksable/ports_out/out1[0]
  /designs/ksable/ports_out/out1[1]
  /designs/ksable/ports_out/out1[2]
  ```

- Use the `ls -dir` command with the redirect arrow to redirect the output to a specified file:

  ```
  rc:/> ls -dir [all_outputs] > areid.txt
  ```

  You can also append arrows (">>").

apropos

```
apropos [-skip_help] [-skip_commands]
     [-skip_attributes] string
```

Performs a case insensitive wildcard search of commands (including their options) and attributes. The command will even encompass the help text of commands and attributes. The search results will be categorized into the following different sections:

- Commands that match search text
- Commands with help text that match search text
- Commands with options (both option name and option help text) that match search text
- Attributes that match search text
- Attributes with help text and attribute objects that matches search text

Options and Arguments

`-skip_help`	Do not include help text in the search results.
`-skip_command`	Do not include commands in the search results.
`-skip_attributes`	Do not include attributes in the search results.
`string`	Specifies the search string

Examples

- The following example searches for the term "generic":

```
rc:/> apropos generic
Commands with option text matching search string:
  synthesize
  write_hdl

Attributes with help text matching search string:
  dp_perform_csa_operations (root)
  dp_perform_sharing_operations (root)
  dp_perform_speculation_operations (root)
  hdl_auto_sync_set_reset (root)
  timing_driven_muxopto (design)
  timing_driven_muxopto (subdesign)
```

- The following example searches for the term "generic" among commands only:

```
rc:/> apropos -skip_attributes generic
Commands with option text matching search string:
  synthesize
  write_hdl
```

clear

```
clear
```

Clears the terminal screen.

date

```
date
```

Returns the data and time. This command is equivalent to the UNIX `date` command.

Examples

```
rc:/> date
Thu Apr 23 01:28:55 PM PDT 2009
```

enable_transparent_latches

```
enable_transparent_latches
```

Enables transparent latches in the design by disabling the `EN` to `Q` arcs in the latch. This command must be used after `elaborate`. Transparent latches are latches with the enable signal held constant at the active state. Without enabling transparent latches, paths through them cannot be traced.

Related Information

Affects this command:	report disabled_transparent_latches on page 373

exec_embedded_script

```
exec_embedded_script
    [-design string] [-subdesign string]
```

Executes embedded scripts found in a specified design or subdesign. To execute the scripts on all top-designs and their subdesigns, use the `exec_embedded_script` command without any arguments. Use this command after the `elaborate` command.

The embedded script of a design or subdesign is stored in the `embedded_script` attribute of that design or subdesign.

Options and Arguments

`-design` *`string`*	Specifies the top level design for which the embedded script need to be executed. Using this option executes scripts for the specified design and every subdesign in it.
`-subdesign` *`string`*	Specifies the full path of the subdesign for which the embedded script needs to be executed. Using this option executes the script only for the specified subdesign.

Examples

- The following commands execute a SDC script, if any, embedded in the RTL description of the design.

  ```
  rc> exec_embedded_script
  rc> exec_embedded_script -design name of top design
  rc> exec_embedded_script -design full vdir path to top design
  rc> exec_embedded_script -subdesign full vdir path to subdesign
  ```

- If design or subdesign is not specified, then using this command executes the embedded SDC script of all designs and all subdesigns.
- If a design is specified, then this command executes the embedded SDC script in the RTL code of the given design and all its subdesigns.
- If a subdesign is specified, then this command executes the embedded SDC script in the RTL code of the specific subdesign only.

  Note: The impact of executing only this embedded script can still be hierarchical. For example, this can happen if an SDC command in the embedded script at this level of design hierarchy specifies a constraint for some instance down in the hierarchy.

- If both a design and a subdesign is specified, then the subdesign setting is ignored.

Related Information

Related command:	elaborate on page 284
Related attributes:	hdl_auto_exec_sdc_scripts

exit

```
exit [code]
```

Exits from the RTL Compiler shell without saving design data.

`Control-c` is a shortcut to the `exit` and `quit` commands.

Important

When exiting, no design data is saved, so it is important to save the design using the `write_hdl` and `write_script` commands.

Options and Arguments

`code`	Specifies the exit code. The following are the built-in exit codes: 0 – normal exit 1 – abnormal exit 246 – exit when no license server is available 245 – exit when no license feature is available 244 – exit when syntax error in script

get_attribute

```
get_attribute {attribute_name [object]
     |-h  {attribute_name | *] {type|*} }
```

Retrieves the value of an attribute set by RTL Compiler or via the `set_attribute` command. You can also use this command to list all attributes of a given object type.

This command is most commonly used within scripts to control the way a script operates based on the value of the attribute, or to retrieve basic information on the tool.

Options and Arguments

`attribute_name`	Specifies the name of an attribute whose value you want to retrieve.
`object`	Specifies the path from where the attribute should be retrieved. *Default*: current working directory
`type`	Specifies the object type for which you want the list of attribute names. Specify any of the following:

actual_scan_chain	hdl_param	operating_condition
actual_scan_segment	hdl_pin	pin
attribute	hdl_proc	pin_bus
clock	hdl_subp	port
clock_domain	instance	port_bus
constant	isolation_rule	power_domain
cost_group	jtag_instruction	power_ground_net
design	jtag_instruction_register	root
exception	jtag_port	scan_chain
external_delay	level_shifter_group	scan_segment
hdl_arch	level_shifter_rule	seq_function
hdl_bind	libarc	subdesign
hdl_block	libcell	subport
hdl_comp	libpin	subport_bus
hdl_config	library	test_clock
hdl_impl	library_domain	test_clock_domain

hdl_inst	message	test_signal
hdl_label	message_group	violation
hdl_lib	mode	wireload
hdl_oper	net	wireload_selection
hdl_pack		

Note: The `clock_domain`, `test_clock_domain`, and `wireload_selection` object types currently do not have any attributes associated with them.

Examples

- The following example retrieves the current value of the `library` attribute on the root directory:

```
rc:/> get_attribute library /
tutorial.lbr
```

- The following example assumes you are already at the root of the design hierarchy, so the object specification is omitted:

```
rc:/> get_attribute library
tutorial.lbr
```

- The following example stores the attribute value in a variable:

```
rc:/> set old_library [get_attribute library /]
tutorial.lbr
```

- The following example returns the area of library cell:

```
rc:/> get_attribute area [find /lib* -libcell nor*]
1.5
```

- The following example lists all root-level attributes starting with `lp`:

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

- The following example lists all attributes for all object types:

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

- The following examples show the difference between the ls `-attribute` and `get_attribute` commands.

  - Using the ls `-attribute` command:

```
rc:/designs> ls -attribute
Total: 2 items
./
async_set_reset_flop_n/                (design)
```

```
Attributes:
  dft_mix_clock_edges_in_scan_chains = false
  wireload = /libraries/slow/wireload_models/sartre18_Conservative
```

- ❑ Using the `get_attribute` `wireload` command:

  ```
  rc:/designs> get_attribute wireload /designs/async_set_reset_flop_n/
  ```

  returns the following wireload model:

  ```
  /libraries/slow/wireload_models/sartre18_Conservative
  ```

  The `ls -attribute` command lists all user modified attributes and their values. The `get_attribute` command lists only the value of the specified attribute. The `get_attribute` command is especially useful in scripts where its returned values can be used as arguments to other commands.

Related Information

Affected by this command:	set_attribute on page 98
Related command:	ls on page 46

get_liberty_attribute

```
get_liberty_attribute attribute_name
         {libarc|libcell|libpin|library|operating_condition|wireload}
```

Options and Arguments

attribute_name	Specifies the name of the Liberty attribute whose value you want to retrieve.
libarc|*libcell*|*libpin*|*library*|*operating_condition*|*wireload*	Specifies the name of the object for which you want to retrieve a Liberty attribute value.

Example

The following command retrieves the `default_operating_conditions` attribute of the `tutorial` library.

```
rc:/> get_liberty_attribute default_operating_conditions tutorial
typical_case
```

get_read_files

```
get_read_files [-quiet] [-command command]
```

Returns information on the files that have been read for the specified command

Options and Arguments

`-command` *`command`*	Specifies the command for which you want the information.
`-quiet`	Suppresses the status message.

Example

```
rc:/> get_read_files -command read_def
point.def
```

help

`help [`*`command`*`]...[> `*`file`*`]`

Provides help on the specified RTL Compiler commands.

Note: You can get help at any stage of the design.

Options and Arguments

`command`	Specifies the command for which you want help. If you do not specify a command name, you get a brief summary of all RTL Compiler commands.
`file`	Specifies the name of the file to which to write the help.

Examples

- The following example requests help for the `report` command:

```
rc:/> help report
That command is:
        report    generate one of various reports
```

- The following example requests help for the `synthesize` and `report` commands:

```
rc:/> help synthesize report
Commands are:
        report        generate one of various reports
        synthesize    synthesize the design
```

include

`include` *`file`*

Executes scripts containing RTL Compiler or Tcl commands in the order they are listed in the `include` command.

When RTL Compiler begins executing, a number of scripts are automatically included. Information on these configuration scripts is given in the *Using Encounter RTL Compiler*.

The use of script files allows automation and modularization of the design flow by placing commonly used commands and sequences of commands into their own scripts. For externally defined components like memories, the vendor can supply a script to create and set up the memory.

This command is identical to the `source` command.

Note: If an error occurs during execution of one of the scripts, execution of that script (and all scripts following it) is stopped.

Options and Arguments

`file`	Specifies the name of the script file to include.

Examples

- The following example includes one script file:

  ```
  rc:/> include constraint.g
  ```

- The following example includes multiple scripts at once:

  ```
  rc:/> include constraint.g; include synth.g
  ```

Related Information

Affected by these attributes:	hdl_search_path
	lib_search_path
	script_search_path

lcd

```
lcd directory
```

Changes the UNIX working directory to the specified directory.

Options and Arguments

directory	Specifies the UNIX directory to which to change the current directory.

Examples

- The following example changes the working directory to the `rtl_lab01` directory in the current UNIX directory.

  ```
  rc:>/ lcd rtl_lab01
  ```

Related Information

Affects these commands:	lls on page 80
	lpwd on page 83
	shell on page 101

license

```
license {checkin | checkout | list}
```

Manages the checking in and checking out of licenses.

Options and Arguments

`checkin`	Checks in a product license that was previously checked out.
`checkout`	Checks-out additional licenses.
`list`	Returns a list of licenses currently checked out.

Related Information

Related commands:	license checkin on page 77
	license checkout on page 78
	license list on page 79

license checkin

`license checkin` *`license`*

Checks in a product license that was previously checked out.

Options and Arguments

`license`	Specifies the license to be checked in. The licenses that can be checked in are: ■ `FE_GPS` ■ `First_Encounter_GXL` ■ `RTL_Compiler_Ultra` ■ `RTL_Compiler_Verification` ■ `RTL_Compiler_Ultra_II_Option` ■ `SOC_Encounter_GPS` ■ `SOC_Encounter_GXL`

Examples

- The following example checks in the `RTL_Compiler_Verification` license:

```
rc:/> license checkin RTL_Compiler_Verification
```

license checkout

`license checkout` *`license`* `[-version` *`float`*`] [-wait]`

Checks-out an additional product license. Licenses can only be checked-out one at a time.

If the license is available, it is checked out and `1` is returned. If the license is not available, a warning message is issued and `0` is returned. To check-in a license, use the `license checkin` command or quit RTL Compiler.

Options and Arguments

`license`	Specifies the license to be checked out. The licenses that can be checked out are: ■ `FE_GPS` ■ `First_Encounter_GXL` ■ `RTL_Compiler_Ultra` ■ `RTL_Compiler_Verification` ■ `RTL_Compiler_Ultra_II_Option` ■ `SOC_Encounter_GPS` ■ `SOC_Encounter_GXL`
`-version` *`float`*	Allows you to specify a specific version of the license. By default, the command uses the hardcoded primary or alternate version of the license that you want to check out.
`-wait`	Waits until a license becomes available. RTL Compiler will wait indefinitely, until the license is available.

Examples

- The following example checks-out the `RTL_Compiler_Verification` license:

```
rc:/> license checkout RTL_Compiler_Verification
Checking out license ’RTL_Compiler_Verification’......   (0 seconds elapsed)
1
```

- The following command checks out version 6.2 of the SOC_Encounter_GPS license. For example, the primary license could be 8.1 and the alternate version 7.1.

```
license checkout SOC_Encounter_GPS -version 6.2
```

license list

```
license list
```

Returns a Tcl list of additional licenses that are currently checked-out. The license used to launch RTL Compiler is not included in this list.

Example

- The following example shows that four additional licenses are currently checked-out. Notice that two instances of `FE_GPS` are checked-out:

  ```
  rc:/> license list
  FE_GPS FE_GPS SOC_Encounter_GPS RTL_Compiler_Verification
  ```

lls

`lls` *`directory`*

Lists the contents of the specified UNIX directory.

Options and Arguments

`directory`	Specifies the UNIX directory whose contents to list.

Examples

- The following example lists the contents of the `rtl_lab01` directory in the current UNIX directory.

```
rc:/> lls rtl_lab01
alu.v
lab01_gates.v
rc.cmd
rc.log
script.g
tutorial.lbr
rc:/>
```

Related Information

Affected by this command:	lcd on page 75
Related commands:	lpwd on page 83
	shell on page 101

lpopd

```
lpopd
```

Removes the topmost element of the UNIX directory stack, revealing a new top element and changes the current directory to the new top element. This command is equivalent to the UNIX `popd` command.

Note: If `lpopd` is issued on a directory stack that has only one element, an appropriate warning message is printed and the `lpopd` command exits without changing the directory stack.

Example

In the following example, the synthesis directory is removed from the UNIX directory stack.

```
rc:/> lpwd
/home/ria/rc_ex/synthesis
rc:/> lpopd
/home/ria/rc_ex
```

Related Information

Affects these commands:	lls on page 80
	lpushd on page 82
	lpwd on page 83

lpushd

```
pushd directory
```

Pushes the specified new target directory onto the UNIX directory stack (as the topmost element) and changes the current directory to that specified directory. This command is equivalent to the UNIX `pushd` command.

Options and Arguments

`directory`	Specifies the name of the UNIX directory to be set as target directory. You can use wildcards when they do not produce an ambiguous reference (more than one match).

Example

In the following example, the synthesis directory is pushed on top of the current UNIX directory.

```
rc:/> lpwd
/home/ria/rc_ex
rc:/> lpushd synthesis
synthesis
rc:/> lpwd
/home/ria/rc_ex/synthesis
```

Related Information

Affects these commands:	lls on page 80 lpopd on page 81 lpwd on page 83

lpwd

```
lpwd
```

Returns the UNIX working directory.

Examples

- The following example shows the current UNIX directory.

  ```
  rc:/> lpwd
  /usr3/verilog_labs
  ```

Related Information

Affected by this command:	lcd on page 75
Related commands:	lls on page 80
	shell on page 101

man

`man { `*`command_name`*` | `*`attribute_name`*` | `*`message_ID`*`}`

Returns detailed information about the specified command, attribute, or message from the online reference manual. You must first set your `MANPATH` environment variable to the following path:

`$CDN_SYNTH_ROOT/share/synth/man`

After setting the `MANPATH` variable, you can obtain manpages for commands and attributes both within RTL Compiler or within the Unix shell. To obtain more information about RTL Compiler messages, you must use the `man` command within RTL Compiler.

Example

- The following example returns the manpage for the `synthesize` command:

```
rc:/> man synthesize
User Commands                                        synthesize(1)

NAME
     synthesize

SYNTAX
          synthesize [-effort {high|medium|low}] [-to_generic]
          [-to_mapped] [-incremental] [<design>]...

DESCRIPTION
     Determines the most suitable design implementation using the
     given  design constraints (clock cycle, input delays, output
     delays, technology library, and so on).
...
```

- The following example returns information about the `TIM-11` message:

```
rc:/> man TIM-11
Entry       : TIM-11
Severity    : Warning
Verbosity   : Message is visible at any ’information_level’ above ’1’.
Description : Possible timing problems have been detected in this design.
Help        : Use ’report timing -lint’ for more information
```

more

`more` *`command`*

Displays the output of the specified command one screen at a time.This command is similar to the UNIX more command.

If the output is several screens, the percentage of characters displayed so far is also shown at the bottom of the screen.

To scroll down the display one line at a time, press *Return.*

To display the next screen, press the SPACE bar.

To stop the display, press *q.*

Options and Arguments

`command`	Specifies the command whose output you want to display with the `more` command.

Examples

The following command displays one screen for the output of the `write_scandef` command.

```
rc:/> more write_scandef
```

```
VERSION 5.4 ;
NAMESCASESENSITIVE ON ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN ria_test ;

SCANCHAINS 6 ;
  - AutoChain_1_seg1_test_clk1_falling
#     + PARTITION p_test_clk1_falling
#       MAXBITS 5
    + START PIN DFT_sdi_1
    + FLOATING
       out1_reg[4] ( IN SI ) ( OUT QN )
       out1_reg[5] ( IN SI ) ( OUT QN )
       out1_reg[6] ( IN SI ) ( OUT QN )
       out1_reg[7] ( IN SI ) ( OUT QN )
       out1_reg[8] ( IN SI ) ( OUT QN )
    + STOP DFT_lockup_g1 D
;

  - AutoChain_1_seg2_test_clk2_falling
```

```
#     + PARTITION p_test_clk2_falling
#       MAXBITS 5
    + START DFT_lockup_g1 Q
    + FLOATING
       out2_reg[4] ( IN SI ) ( OUT QN )
       out2_reg[5] ( IN SI ) ( OUT QN )
       out2_reg[6] ( IN SI ) ( OUT QN )
       out2_reg[7] ( IN SI ) ( OUT QN )
       out2_reg[8] ( IN SI ) ( OUT QN )
    + STOP DFT_lockup_g259 D
;

  - AutoChain_1_seg3_test_clk3_falling
#     + PARTITION p_test_clk3_falling
#       MAXBITS 5
    + START DFT_lockup_g259 Q
    + FLOATING
       out3_reg[4] ( IN SI ) ( OUT QN )
       out3_reg[5] ( IN SI ) ( OUT QN )
       out3_reg[6] ( IN SI ) ( OUT QN )
       out3_reg[7] ( IN SI ) ( OUT QN )
       out3_reg[8] ( IN SI ) ( OUT QN )
--More--(51%)
```

- The following command outputs the library check report results:

  ```
  more report checks -library
  ```

- The following command outputs the HDL lint check report results, displaying every message for every message ID, one message per line:

  ```
  more report checks -hdl_lint -detail
  ```

quit

```
quit
```

Exits from the RTL Compiler shell without saving design data.
Control-C is a shortcut to the `exit` and `quit` commands.

Important

When exiting, no design data is saved, so it is important to save the design using the `write_hdl` and `write_script` commands.

Options and Arguments

code	Specifies the exit code. The following are the built-in exit codes: 0 – normal exit 1 – abnormal exit 246 – exit when no license server is available 245 – exit when no license feature is available 244 – exit when syntax error in script

rc

```
rc [-32] [-64]  [-E]
     [-execute command] [-files file] [-post command]
     [-gui] [-no_custom]
     [-cmdfile file] [-logfile log_file]  [-overwrite]
     [-lsf_cpus integer] [-lsf_queue string]
     [-N integer] [-version]
     [-queue] [-cfm_eco] [-cfo_eco_gxl] [-rcl] [-vdi]
     [-use_license {Conformal_ECO | Conformal_ECO_GXL
      | Encounter_Design_Planner_XL
      | Encounter_Digital_Impl_Sys_XL
      | FE_GPS | First_Encounter_GXL
      | RTL_Compiler_L | RTL_Compiler_Physical
      | RTL_Compiler_Ultra | RTL_Compiler_Verification
      | SOC_Encounter_GPS | SOC_Encounter_GXL
      | Virtuoso_Digital_Implement | Virtuoso_Digital_Implem}]
```

Starts RTL Compiler from the UNIX environment. If you specify multiple licenses with the `-use_license`, only the last one will be used.

Note: Use `rc64` instead of `rc` if you are using RTL Compiler on a Solaris 64-bit environment.

Tip

You can abbreviate the options for the `rc` command as long as there are no ambiguities with its other options. In the following example, `-ever` would imply the `-version` option:

```
unix> rc -ver
```

Just using `rc -v` would not work because there is more than one option that starts with the letter “v.”

Options and Arguments

`-32`	Launches RTL Compiler in 32-bit mode. This is the default mode.
`-64`	Launches RTL Compiler in 64-bit mode.You can set the `CDS_AUTO_64BIT` environment variable to `ALL` to launch not only RTL Compiler, but all Cadence tools in 64-bit. You will not need to specify the `-64` option if you use this variable.
`-big`	Enables the back ground mode.
`-cfm_eco`	Starts RTL Compiler with a `Conformal_ECO` license.

`-cfm_eco_gxl`	Starts RTL Compiler with a `Conformal_ECO_GXL` license.
`-cmdfile` *`file`*	Specifies the command file name. If a file with this name already exists in your unix directory, the new command file will overwrite the existing file. *Default*: `rc.cmd`
`-E`	Specifies that RTL Compiler must exit if a script error is found.
`-execute` *`command`*	Specifies the command or Tcl code to execute as a quoted string before any other files specified with the `-files` option are processed.
`-files` *`file`*	Specifies the names of the scripts (or command files) to execute.
`-gui`	Launches the RTL Compiler Graphical User Interface (GUI). **Note:** GUI commands are only available in the GUI version of RTL Compiler. See GUI Guide for Encounter RTL Compiler for detailed information on GUI commands.
`-logfile` *`logfile`*	Specifies the log file name. If a file with this name already exists in your unix directory, the new log file will overwrite the existing file (new content with the same filename). *Default*: `rc.log`
`-lsf_cpus` *`integer`*	Specifies the number of LSF CPUs to use for super-threading.
`-lsf_queue` *`string`*	Specifies the name of the LSF queue.
`-N` *`integer`*	Specifies the number of licenses to use for VDIO and RC-L. *Important* The licenses must be on the same server.
`-no_custom`	Specifies to read only the master `.synth_init` file, located in the installation directory. By default, RTL Compiler also loads the initialization file in your home directory and in your current design directory.
`-overwrite`	Allows overwriting of the default and specified command and log files.
`-post` *`string`*	Specifies the command to be executed after the file(s) specified with the `-files` option is (are) processed.

`-queue`	Puts an RTL Compiler session in a queue if a license is not currently available. Once an RTL Compiler license becomes available, an RTL Compiler session is launched.
`-rcl`	Launches RTL Compiler with the `RTL_Compiler_L` license.
`-use_license` *`string`*	Specifies which license to use at startup. If the specified license is unavailable, startup will not continue and the command will fail. If you specify multiple licenses, only the last one will be used.
`-version`	Returns the version number without launching the executable.
`-vdi`	Launches RTL Compiler with the Virtuoso Digital Implementation license. It will first try to use the `Virtuoso_Digital_Implem` license. If that license is unavailable, then it will use the `Virtuoso_Digital_Implement` license.

Examples

■ The following example first starts RTL Compiler, then executes a script file.

```
unix> rc
rc:/> include script.g
```

■ The following example specifies two script files in the command line:

```
unix> rc -f script1.g -f script2.g
```

■ The following example specifies a logfile name with the `-logfile` option. The default name is `rc.log` if there is no other logfile in the directory from which RTL Compiler is launched:

```
unix> rc -logfile pov.log
```

Do not use the UNIX `tee` command and pipe (`|`) to specify your logname: doing so would not allow you to use the `control-c` key sequence to gracefully exit a process like incremental optimization.

■ The following example uses the `-execute` option to execute a script file:

```
unix> rc -exute "include script.g"
```

■ The following example specifies the library path in the command line:

```
unix> rc -ex "set_attribute lib_search_path /net/serverx/libs/mylib"
```

- The following commands start RTL Compiler with the `RTL_Compiler_Physical` license:

  ```
  unix> rc -physical
  unix> rc -use_license RTL_Compiler_Physical
  ```

- The following example starts RTL Compiler with the `RTL_Compiler_Verification` license:

  ```
  unix> rc -use_license RTL_Compiler_Verification
  ```

  This command will fail if the `RTL_Compiler_Verification` license is unavailable.

  If the `-use_license` option is not specified, then the `RTL_Compiler_Ultra`, `RTL_Compiler_Verification`, `FE_GPS`, and `SOC_Encounter_GPS` licenses will be checked, respectively, and RTL Compiler will start with the first available license. For example, if in the following example the only available license was `FE_GPS`, then RTL Compiler will start with the `FE_GPS` license:

  ```
  unix> rc
  ```

- The following commands have the same effect. Therefore, you should use one or the other and not both in conjunction:

  ```
  unix> rc -use_license Virtuoso_Digital_Implement
  ```

  is the same as:

  ```
  unix> rc -vdi
  ```

- The following example launches RTL Compiler on four LSF CPUs in the LSF queue named `my_queue`:

  ```
  unix> rc -lsf_cpus 4 -lsf_queue my_queue
  ```

- The following example will launch not only RTL Compiler, but all Cadence tools in 64-bit mode:

  ```
  unix> setenv CDS_AUTO_64BIT ALL
  unix> rc
  ```

- The following example specifies that two Virtuoso Digital Implementation licenses be used:

  ```
  unix> rc -vdi -N 2
  ```

Related Information

Affected by this attribute:	command_log

redirect

`redirect [-append] [-tee] [-variable] *target command*`

Redirects the standard output to a file or variable. You can write out a gzip compressed file by adding the `.gz` extension to the filename.

Note: Messages generated by the command whose output you are redirecting, can be mixed with the command output. To limit the number of messages included, you can change the value of the `information_level` root attribute.

Options and Arguments

`-append`	Append the generated output to the specified file or Tcl variable.
`command`	Specifies the command or Tcl code to execute as a quoted string.
`target`	Specifies the name of the file or variable to which to write the output.
`-tee`	Also writes the output to standard output (`stdout`).
`-variable`	Redirects the output to a Tcl variable.

Examples

- The following example sends the output generated by the `report gates` command to a file called `gates.rep`:

```
redirect gates.rep "report gates"
```

- The following example appends information to the existing `gates.rep` file:

```
redirect -append gates.rep "report gates"
```

- The following example sends the report to `stdout` and to a file on the disk:

```
redirect -tee gates.rep "report gates"
```

- The following example prevents output generated during reading of the script from being sent to the screen by sending it to `/dev/null.`

```
redirect /dev/null "include script.g"
```

- The following example stores the timing report in a variable `$rep_var`. You can use Tcl commands to manipulate or parse the variable.

```
redirect -variable rep_var "report timing"
```

- The following examples output a timing and a `scan_power` gzip compressed report file:

```
redirect file.gz "report timing"
redirect file.gz "report scan_power"
```

reset_attribute

```
reset_attribute { [-quiet] attribute_name [object...]
                | -h  {attribute_name | *] {type|*}}
```

Resets an attribute to its default value.

Options and Arguments

attribute_name	Specifies the attribute that should be returned to its default value. The "*" wildcard character is supported.
object	Specifies the object for which the attribute values should be returned to the default values.
`-quiet`	Suppresses those messages that indicate which attributes and objects are being affected.
type	Specifies the object type for which you want the list of attribute names. The "*" wildcard character is supported. Specify any of the following types:

actual_scan_chain	hdl_lib	mode
actual_scan_segment	hdl_oper	net
attribute	hdl_param	operating_condition
clock	hdl_pin	pin
cost_group	hdl_proc	port
design	hdl_subp	port_bus
exception	instance	power_domain
external_delay	isolation_rule	power_ground_net
hdl_arch	level_shifter_group	root
hdl_bind	libarc	subdesign
hdl_block	libcell	subport
hdl_comp	libpin	subport_bus
hdl_impl	library	test_clock
hdl_inst	library_domain	test_signal
hdl_label	message	wireload

Examples

- The following example specifies that all retiming attributes should be returned to their default values:

  ```
  rc:/> reset_attribute retime* *
  ```

- The following example specifies that all attributes on all sequential instances be returned to their default values:

  ```
  rc:/> reset_attribute * [find / -instance *seq/*]
  ```

- The following command lists all valid attributes that you can reset:

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

  Both `type` and `attribute_name` accept wildcard strings.

resume

```
resume
```

Restarts the current Tcl process. This command only works in conjunction with the `suspend` command. See the suspend command to see how these commands work together to stop and restart a Tcl process.

Related Information

Related command:	suspend on page 103

sdc_shell

```
sdc_shell
```

Opens the SDC shell within RTL Compiler. All SDC commands can be used without the `dc::` prefix inside the SDC shell. The command `exit` quits the SDC shell and returns you back to the RTL Compiler prompt.

Example

■ The following example launches the SDC shell within RTL Compiler and then exits:

```
rc:/> sdc_shell
Info    : Entering sdc_shell. [SDC-300]
       : All sdc commands will work without the dc:: prefix inside sdc_shell.
Type ’exit’ to leave the shell.
sdc_shell> exit
Info    : Leaving sdc_shell. [SDC-301]
        : Type sdc_shell to use it again.
rc:/>
```

set_attribute

```
set_attribute
     { attribute_name attribute_value [objects]
       [-quiet] [-lock]
     | -h  {attribute_name | *] {type|*}}
```

Sets the value of a specified attribute or returns a list of valid attributes.

Attributes are placed on directory objects to control the way they are processed by RTL Compiler. They can also be used to control the synthesis process, the style of the generated code, and numerous other things. A complete list of all attributes is contained in the *Attribute Reference for Encounter RTL Compiler*.

Note: Not all attributes can be set. Attempting to set a *read-only* attribute returns an error. The *Attribute Reference for Encounter RTL Compiler* indicates whether an attribute is read-write or read-only.

Options and Arguments

attribute_name	Specifies the name of the attribute whose value you want to set.
attribute_value	Specifies the new attribute value. The value can be either a Boolean, integer, or string. A compound string (containing spaces) should be represented as a list using double-quotes or braces.
-lock	Locks the specified attribute’s value so that it cannot be changed. The attribute becomes read-only.
objects	Specifies the path(s) to the objects. *Default*: current directory
-quiet	Suppresses those messages that indicate which objects are being affected. Alternatively, when setting an attribute on an object, an information message will not be printed.
type	Specifies the object type for which you want the list of attribute names. Specify any of the following:

actual_scan_chain	hdl_lib	mode
actual_scan_segment	hdl_oper	net
attribute	hdl_param	operating_condition

clock	hdl_pin	pin
cost_group	hdl_proc	port
design	hdl_subp	port_bus
exception	instance	power_domain
external_delay	isolation_rule	power_ground_net
hdl_arch	level_shifter_group	root
hdl_bind	libarc	subdesign
hdl_block	libcell	subport
hdl_comp	libpin	subport_bus
hdl_impl	library	test_clock
hdl_inst	library_domain	test_signal
hdl_label	message	wireload

Examples

- The following example lists all valid attributes that you can set:

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

  Both `type` and `attribute_name` accept wildcard strings.

- The following example lists all valid attribute names that contain the string `dont`:

  ```
  rc:/> set_attribute *dont* * -help
  ```

- The following example sets the `information_level` attribute, which controls the verbosity of the tools, to the value of 5 and assumes the current directory for the path:

  ```
  rc:/> set_attribute information_level 5
      Setting attribute of root /: 'information_level' = 5
  ```

- In the following example, the path needed to be specified because `information_level` is a root attribute and would not have been found in the current path:

  ```
  rc:/designs/alu> set_attribute information_level 5 /
      Setting attribute of root /: 'information_level' = 5
  ```

- The following locks the technology library search path to `/home/Test/bree` by locking the `lib_search_path` attribute. For the rest of the session, the `lib_search_path` attribute becomes read-only.:

  ```
  rc:/> set_attribute -lock lib_search_path /home/Test/bree
  ```

Related Information

Affects these commands:	ls on page 46
	get_attribute on page 68

shell

```
shell command_string
```

Executes a UNIX shell command from the RTL Compiler shell.

When executing shell commands, RTL Compiler uses `/bin/sh`.

Important

Attempting to change the working directory for the RTL Compiler shell using `shell cd ..`is not possible because each `shell` command is executed in its own shell, and that shell is killed once the command is complete.

Options and Arguments

command_string	Specifies the UNIX command to execute. You can specify any valid UNIX command. A sequence of commands must be specified in the same string.

Examples

- The following example uses the *sh* command to get the current date:

```
rc:/> shell date
…
```

Related Information

Affected by this command:	lcd on page 75
Related commands:	lls on page 80
	lpwd on page 83

suppress_messages

```
suppress_messages [-n integer] message_id...
```

Suppresses printing of the specified message in the log file.

Options and Arguments

message_id	Specifies the message identification.
-n *integer*	Prints the specified message only *n* number of times. The default value is `0`.

Examples

- The following example suppresses the `VLOG-1` and `VLOG-2` messages:

  ```
  rc:/> suppress_messages { VLOG-1 VLOG-2 }
  ```

Related Information

Related command:	unsuppress_messages on page 104

suspend

```
suspend
```

Stops the current Tcl process. Type `resume` to restart the process from where it was stopped. Issuing the `suspend` command brings up the RTL Compiler prompt and any commands or attributes that are issued during this time will not have access to the temporary variables from the suspended Tcl process. However, global variables and Tcl processes can still be accessed.

Note: Any commands that you enter after issuing the `suspend` command but before the `resume` command will not be included in Tcl's command history. However, these commands will be included in RTL Compiler's command editor history.

unsuppress_messages

`unsuppress_messages` *`message_id`*`...`

Allows a previously suppressed message to be printed.

Options and Arguments

`message_id`	Specifies the message identification.

Examples

- The following example suppresses the `VLOG-1` and `VLOG-2` messages and then allows them to be printed again:

  ```
  rc:/> suppress_messages { VLOG-1 VLOG-2 }
  rc:/> unsuppress_messages { VLOG-1 VLOG-2 }
  ```

Related Information

Related command:	suppress_messages on page 102

3 GUI Text

General GUI Text Commands

gui_hide

```
gui_hide
```

Hides the GUI. Type the `gui_show` command to re-display the GUI.

gui_info

```
gui_info string
```

Adds the specified text string in the info section of the status bar. The text remains until it is overwritten.

This command is usually used to print persistent messages (for example, Design is mapped).

gui_raise

```
gui_raise [-nosync]
```

Keeps the GUI window on top of all other windows.

Options and Arguments

`-nosync`	Does not synchronize the GUI when the GUI is raised.

gui_reset

```
gui_reset
```

Resets the busy indicator if it remains busy.

The busy indicator can remain red if the script that set the busy indicator is broken and therefore the busy indicator does net get reset or cleared.

.gui_selection

`gui_selection`

Returns the selection list, which consists of the object path for instance, net, pin, and port objects. The list can contain multiple objects.

Example

- The following example returns a combinantional instance:

  ```
  rc:/> gui_selection
  /designs/mul_clk/instances_comb/g14
  ```

  If you enable the toolbar by un-checking *Hide Toolbar* under the Preferences menu, the complete path for the last selected object will be displayed.

gui_show

`gui_show [-nosync]`

Displays the GUI after using `gui_hide` command.

Options and Arguments

`-nosync`	Does not synchronize the GUI when the GUI is displayed.

gui_status

`gui_status` *`string`*

Adds the specified text string in the status section of the status bar (window right to the busy indicator). The text remains until it is overwritten.

This command is usually used to print transient messages (for example, Loading library *`name`*).

gui_update

`gui_update`

Updates the GUI. This is the same as selecting *Synchronize GUI* under the File menu.

HDL Viewer GUI Text Commands

gui_hv_clear

```
gui_hv_clear
```

Removes all the data in the HDL Viewer.

gui_hv_get_file

```
gui_hv_get_file
```

Returns the name of the file currently loaded in HDL Viewer.

gui_hv_load_file

`gui_hv_load_file` *`filename`*

Loads the specified file name into the HDL Viewer. Same as clicking the *Open HDL File* icon in the HDL Viewer.

gui_hv_set_indicators

`gui_hv_set_indicators [-clear] `*`line_number column_number`*

Sets the *`line`* and *`column`* number indicators. Using this command is useful if you are writing a script and you want to highlight a specific line in the HDL Viewer.

Options and Arguments

`-clear`	Removes all tag highlighting. If this option is not used, then the specified line number is highlighted.
`column_number`	Specifies the column number. If the *`column_number`* argument is not specified, then the default is `0`.
`line_number`	Specifies the line number.

Example

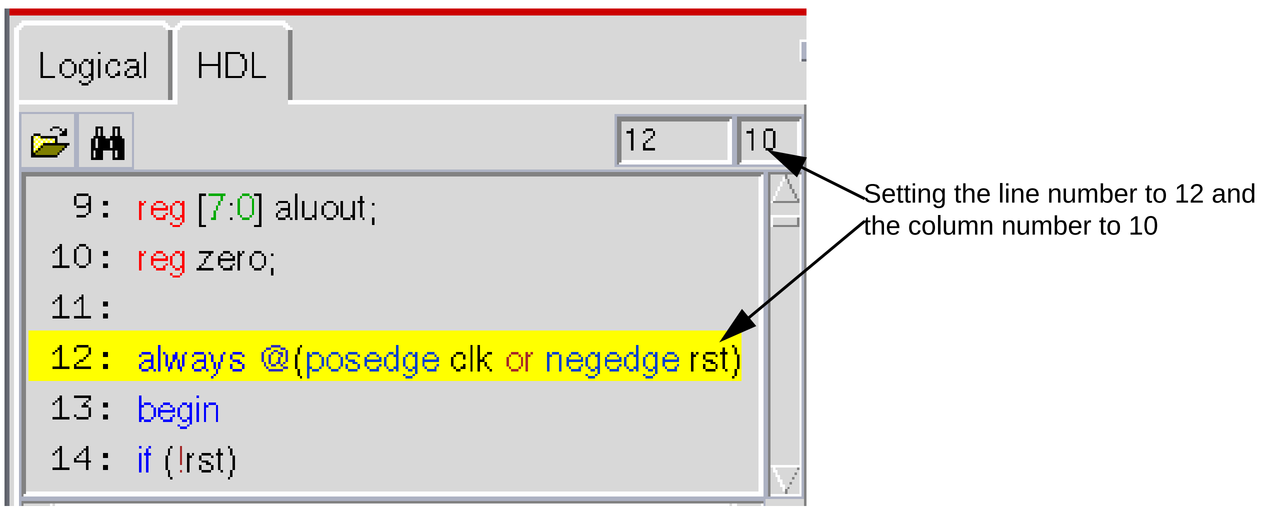
- The following command highlights the specified line number and sets the line number to `12` and the column number to `10`, as shown in Figure 3-1.

  ```
  rc:/> gui_hv_set_indicators 12 10
  ```

- The following commands sets the line number to 12, the column number to 10, and removes the highlighting:

  ```
  rc:/> gui_hv_set_indicators 12 10 -clear
  ```

Figure 3-1 Setting Line and Column Numbers in the HDL Viewer



```
 9: reg [7:0] aluout;
10: reg zero;
11:
12: always @(posedge clk or negedge rst)
13: begin
14: if (!rst)
```

Schematic Viewer GUI Text Commands

gui_sv_clear

```
gui_sv_clear
```

Clears highlighting and selection lists in the Schematic Viewer.

gui_sv_get_instance

```
gui_sv_get_instance
```

Returns the objects for the currently displayed hierarchical instance.

gui_sv_grey

```
gui_sv_grey [on | off]
```

Controls the grey mode. You can also right-click the mouse button in the Schematic Viewer and select *Grey Mode On* or *Grey Mode Off.*

Options and Arguments

`-on`	Turns on grey mode in the Schematic Viewer.
`-off`	Turns off grey mode in the Schematic Viewer.

gui_sv_highlight

```
gui_sv_highlight [-append] [-color color] [-center]
     [-from {port|pin}] [-to {port|pin}] [-zoomto]
     {port|pin|net|instance}
```

Highlights the specified object in the Schematic Viewer.

Options and Arguments

`-append`	Appends an object to the highlighted list.
`-center`	Centers an object in the Schematic Viewer.
`-color` *`color`*	Specifies the color for highlighting an object.
`-from {`*`port`*`|`*`pin`*`}`	Specifies the start point of the highlighted net.
`-to {`*`port`*`|`*`pin`*`}`	Specifies the end point of the highlighted net.
`-zoomto`	Zooms into the specified port, pin, instance or net.

Examples

- The following Tcl procedure highlights inverters in the Schematic Viewer

```
# highlight inverters
proc highlight_inverters {idir type odir} {
    # only proceed if no object under cursor
    if {$type != "none"} return
    # clear any highlight and selection
    gui_sv_clear
    _find_inverters $idir
}
proc _find_inverters {idir} {
    # search for inverters
    foreach inst [find $idir -maxdepth 2 -instance *comb/*] {
        if {[get_attribute inverter $inst] == "true"} {
            gui_sv_highlight $inst -append -color green
        }
    }
}
```

- The following example highlights the falling clock edge flip-flops

```
# highlight falling clock edge flip-flops
proc highlight_falling_edge_ff {idir type odir} {
    # only proceed if no object under cursor
    if {$type != "none"} return
    # clear any highlight and selection
    gui_sv_clear
    _find_falling_edge_ff $idir
}
proc _find_falling_edge_ff {idir} {
    # search for sequential gates with falling clock edge
    foreach inst [find $idir -maxdepth 2 -instance *seq/*] {
        set libcell [get_attribute libcell $inst]
        if {$libcell == ""} continue
        if {[get_attribute flop $libcell] == "true"} {
            foreach libpin [find $libcell -libpin *] {
                if {[get_attribute output $libpin] == "true"} {
                    foreach arc [get_attribute incoming_timing_arcs $libpin] {
                        set liberty [get_attribute liberty_attributes $arc]
                        if {[lsearch $liberty falling_edge] != -1} {
                            gui_sv_highlight $inst -append
                        }
                    }
                }
            }

        }
    }
}
```

gui_sv_load

`gui_sv_load {`*`design`*` | `*`instance`*`}`

Specifies an hierarchical instance or design to load into the main Schematic Viewer.

Physical Viewer GUI Text Commands

gui_pv_airline_add

```
gui_pv_airline_add -name name
     -from {port|instance} -to {port|instance}
     [-color color] [-label label] [-nodisplay]
```

Adds specified airline to the Physical Viewer.

Options and Arguments

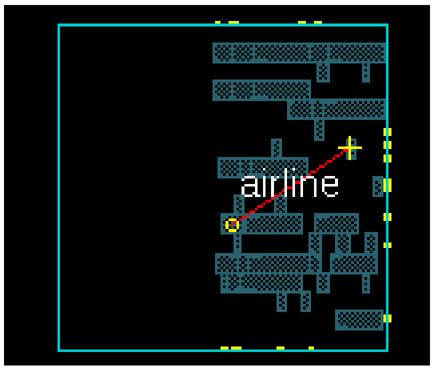
`-color` *`color`*	Specifies the airline color. *Default*: `blue`
`-from {`*`port`*`|`*`instance`*`}`	Specifies the from object.
`-label` *`label`*	Specifies a label to place at the center of the airline.
`-name` *`name`*	Specifies the airline name.
`-nodisplay`	Specifies not to display the airline.
`-to {`*`port`*`|`*`instance`*`}`	Draws an airline from the center of the from object to the center of the to object.

Example

- The following command creates a `red` airline named `test` from the `g442` port to the `g412` port with the `airline` label, as shown in Figure 3-2.

  ```
  gui_pv_airline_add -from g442 -to g412 -color red -label airline -name test
  ```

Figure 3-2 Specified Airline in the Physical Viewer



gui_pv_airline_delete

`gui_pv_airline_delete` *`name`*

Deletes airlines from the Physical Viewer with the specified *`name`*.

Options and Arguments

`name`	Specifies the airline name to be deleted. The airline name must have been added using the gui_pv_airline_add command.

Example

- The following command removes the airline `test` from the Physical Viewer:

```
rc:/> gui_pv_airline_delete test
```

gui_pv_airline_display

`gui_pv_airline_display` *`name`*

Options and Arguments

`name`	Specifies the airline name to be displayed. The airline name must have been added using the gui_pv_airline_add command.

Displays the specified airline name in the Physical Viewer.

Example

- The following command removes the airline `test` from the Physical Viewer:

```
rc:/> gui_pv_airline_display test
```

gui_pv_airline_raw_add

```
gui_pv_airline_raw_add [-name name]
     [-color color] [-label label]
     [-nodisplay] -fx float -fy float
     -tx float -ty float
```

Creates an airline in the Physical Viewer between two points specified by their coordinates.

Options and Arguments

gui_pv_clear

`-color` *`color`*	Specifies the airline color.
	Default: `blue`
`-fx` *`float`*	Specifies the x coordinate of the from object.
`-fy` *`float`*	Specifies the y coordinate of the from object.
`-label` *`label`*	Specifies a label to place at the center of the airline.
`-name` *`name`*	Specifies the airline name.
`-nodisplay`	Specifies not to display the airline.
`-tx` *`float`*	Specifies the x coordinate of the to object.
`-ty` *`float`*	Specifies the y coordinate of the to object.

```
gui_pv_clear [-object_list]
```

Clears highlighting from the specified object list.

Options and Arguments

`-`*`object_list`*	Clears all items in the specified list.

gui_pv_highlight

```
gui_pv_highlight [-color color] [-collect] [-append]
     [-group name] [-label string] [-stipple]
     {blockage | gcell | instance | pcell | port | region}...
```

Highlights objects in the Physical Viewer

Options and Arguments

`-append`	Appends the object to highlight.
`-collect`	Collects objects to highlight
`-color` *color*	Specifies the color for highlighting. *Default*: red
{*blockage* | *gcell* | *instance* | *pcell* | *port* | *region*}	Specifies the object to highlight if it is in the scope of the current Physical Viewer.
`-group` *name*	Specifies the group name for the object.
`-label` *string*	Specifies the object label.
`-stipple`	Specifies to use stipple fill pattern to highlight the object.

Examples

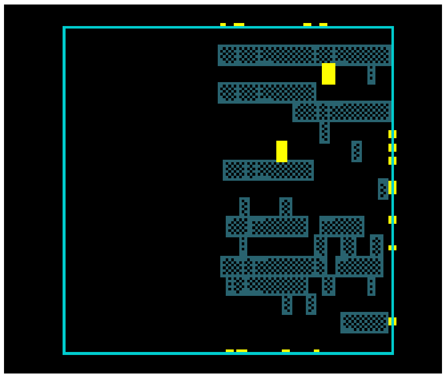
- The following command highlights the g411 port in yellow (see Figure 3-3 on page 121):

  ```
  rc:/> gui_pv_highlight -color yellow g411
  ```

- The following command adds the g405 port to the highlighted list (see Figure 3-3 on page 121):

  ```
  rc:/> gui_pv_highlight -color yellow -append g405
  ```

Figure 3-3 Highlighting an Object in the Physical Viewer



- The following command specifies the instance `eve1` to be highlighted with the `gui_pv_draw_collection` command. The `-group` option indicates that to which group eve1 should belong. In this case, it is `laurence`:

  ```
  rc:/> gui_pv_highlight -collect /designs/bree/instances_hier/eve1/ \
      -group laurence
  ```

Related Information

Related command:	gui_pv_clear on page 119

gui_pv_highlight_update

```
gui_pv_highlight_update -property string
     -value string [-group string]
     {blockage | gcell | instance | pcell | port | region}...
```

Updatea the object highlight in the Physical Viewer.

Options and Arguments

`{`*`blockage`*`|`*`gcell`*`|`*`instance`*`|`*`pcell`*`|`*`port`*`|`*`region`*`}`	
	Specifies the names of the objects for which to update the highlight.
`-group` *`string`*	Specifies the object group name.
`-property` *`value`*	Specifies the property to update.
	You can update the colors, the labels, or the stipple pattern.
`-value` *`value`*	Specifies the object property value.

gui_pv_label

```
gui_pv_label
     [-color color] -x float -y float label
```

Adds a text label at the specified point in the Physical Viewer.

Options and Arguments

`-color` *`color`*	Specifies the label color.
	Default: `white`.
`label`	Specifies the label name.
`-x` *`float`*	Specifies the x coordinate for the label.
`-y` *`float`*	Specifies the y coordinate for the label.

gui_pv_redraw

```
gui_pv_redraw
```

Redraws the contents of the Physical Viewer.

gui_pv_selection

```
gui_pv_selection
```

Returns the list of selected objects in the Physical Viewer.

Note: You can select multiple objects by pressing the Shift key while drawing a region with the left mouse button. All objects overlapping that region are selected.

gui_pv_snapshot

```
gui_pv_snapshot
     [-overwrite] [-png] [-utilization] [-congestion] file
```

Saves a snapshot of the part of the design that is visible in the Physical Viewer.

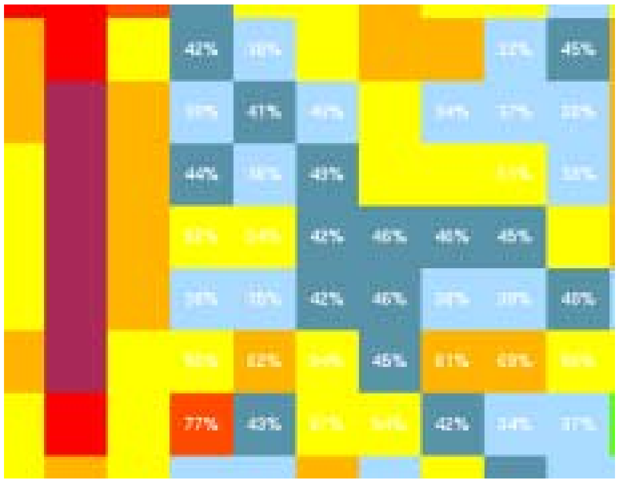
Options and Arguments

`-congestion`	Overlays a congestion map on top of what is visible in the Physical Viewer before making a snapshot.
	Note: This option eliminates the need to first turn on the display of the congestion map.
`file`	Specifies the name of the file in which to save the snapshot.
`-png`	Creates the file in PNG format.
	Default: JPG format
`-overwrite`	Specifies to overwrite an existing file.
	If you forget this option, you'll get a message that indicates that the file exists.
`-utilization`	Overlays a utilization map on the part of the design that is visible in the Physical Viewer before making a snapshot.
	Note: This option eliminates the need to first turn on the display of the utilization map.

Examples

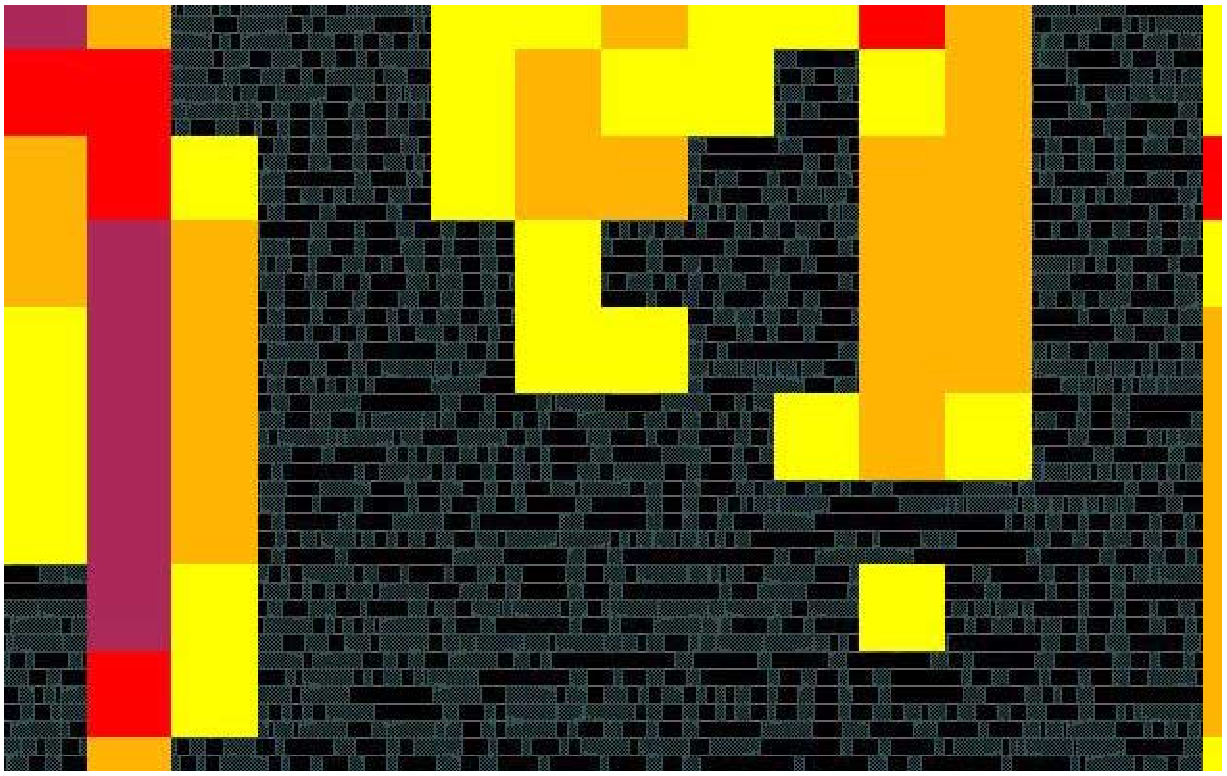
- The following command saves a snapshot of the utilzation map in the util.png file.

  ```
  gui_pv_snapshot -png -utilization util.png
  ```



- The following command saves a snapshot of the congestion map in the file with basename congest. Since the -png option is not specified, the file is specified in JPG format.

  ```
  gui_pv_snapshot -congestion congest
  ```



gui_pv_zoom_fit

```
gui_pv_zoom_fit
```

Performs a “zoom fit” command in the Physical Viewer.

gui_pv_zoom_in

```
gui_pv_zoom_in
```

Performs a “zoom in” command in the Physical Viewer.

gui_pv_zoom_out

```
gui_pv_zoom_out
```

Performs a “zoom out” command in the Physical Viewer.

gui_pv_zoom_to

```
gui_pv_zoom_to
```

Zooms to the bounding box around selected objects in the Physical Viewer.

4

Chipware Developer

cwd

```
cwd {check | create_check | report_check}
```

Controls the ChipWare Developer (CWD) Linter in the ChipWare developer framework.

Options and Arguments

`check`	Invokes the CWD Linter.
`create_check`	Registers a check to the Linter.
`report_check`	Reports information about various check names and check points.

Related Information

Related commands:	cwd check on page 129
	cwd create_check on page 133
	cwd report_check on page 135

cwd check

```
cwd check [-effort string]
     [-skip
       [ hdl_lib | hdl_comp | hdl_pack | hdl_oper
       | hdl_arch | hdl_impl | hdl_bind | hdl_param
       | hdl_pin]... ]
     [ [-summary] [-verbose] | -quiet ]
       [ hdl_lib | hdl_comp | hdl_pack | hdl_oper
       | hdl_arch | hdl_impl | hdl_bind | hdl_param
       | hdl_pin]...]
     [> file]
```

Exercises checking rules and summarizes the outcome in various degrees of verboseness. The `cwd check` command can run on one or more of any of the following `hdl_objects`:

- `hdl_lib`
- `hdl_oper`
- `hdl_comp`
- `hdl_bind`
- `hdl_impl`
- `hdl_param`
- `hdl_pin`
- `hdl_pack`
- `hdl_arch`

The RTL Compiler path to the `hdl_objects` to be checked can either be an absolute path:

```
rc:/> cwd check /hdl_libraries/CW/components/CW_add
```

Or it can be a relative path with respect to the current working directory:

```
rc:/> cd /hdl_libraries/CW/components
rc:/> cwd check CW_add
```

You can use wild cards for specifying multiple `hdl_objects`:

```
rc:/> cwd check /hdl_libraries/CW/components/*add*
```

or:

```
rc:/> cd /hdl_libraries/CW/components
rc:/> cwd check *add*
```

You can specify multiple directories at the same time:

```
rc:/> cwd check {/hdl_libraries/CW /hdl_libraries/DW02}
```

By default, `cwd check` checks all `hdl_objects` underneath the specified set of `hdl_objects`. That is, it traverses the directory tree hierarchically and exercises all checks of all `hdl_objects` it traverses.

Options and Arguments

`-effort` *`string`*	Specifies the effort level. There are two effort levels: `low` and `medium`. The default effort level is `low`. `Low` — CWD linter runs checking rules that are registered as `low` effort. That is, it runs those checking rules that do not require reading any HDL code (of synthesis models). `Medium` — CWD linter runs checking rules that are registered as `low` and `medium` effort. That is, it runs those checking rules that may require parsing HDL code (of synthesis models) but do not require elaborating it. With each `medium` effort level check, the CWD linter automatically loads the HDL code before performing the check. For example, a check at this effort level may look at the `hdl_arch` of an `hdl_impl` and examine ordering of pins and parameters.
`file`	Specifies the filename to store the output of the command.

`hdl_lib` `|` *`hdl_comp`* `|` *`hdl_pack`* `|` *`hdl_oper`* `|` *`hdl_arch`* `|` *`hdl_impl`* `|` *`hdl_bind`* `|` *`hdl_param`* `|` *`hdl_pin`*

Specifies the `hdl_objects` to check.

`-quiet`
Only reports error and warning messages, if any. This is the recommended verbosity level when CWD linting is part of a routine process without any error expectations.

`-skip {`*`hdl_lib`* `|` *`hdl_comp`* `|` *`hdl_pack`* `|` *`hdl_oper`* `|` *`hdl_arch`* `|` *`hdl_impl`* `|` *`hdl_bind`* `|` *`hdl_param`* `|` *`hdl_pin`*`}`

Specify one or more `hdl_objects` to skip.

`-summary`
First reports error or warning messages, if any, and then produces a summary table of the pass/fail count of each checking rules exercised. This level of detail is the default verbosity level.

`-verbose`	Produces a detailed report. In addition to the information produced by the `-summary` option, it also reports the pass/fail status of each check process exercised on each `hdl_object`.

Examples

- The following example runs checking rules on the `CW` libraries as well as all the other `hdl_objects` under it. The CWD linter will run all default (low-effort) mode checking rules up to the severity specified by the `information_level` attribute. A summary will be produced at the end.

```
rc:/> get_attribute information_level
1
rc:/> cwd check /hdl_libraries/CW
Check_name                              Passed Failed
-----------------------------------------------------
component_location                        146     0
component_sim_model_location              128     0
component_syn_model_is_vhdl               146     0
implementation_legality_formula           147     0
implementation_location                   147     0
implementation_preelab_script_location    147     0
non_builtin_implementation_location       147     0
package_default_location                   1      0
package_default_location_filesize          1      0
parameter_formula                         472     0
pin_bit_width                            1136     0
pin_parameter_in_bit_width               1114     0
-----------------------------------------------------
```

- The following example runs checking rules on all the `hdl_objects` under `parameters` and produces a verbose report:

```
rc:/> cwd check /hdl_libraries/CW/components/CW_mult/parameters/* -verbose
checking param wA
Check ::cwd::parameter_formula::check_proc passed on /hdl_libraries/CW/
components/CW_mult/parameters/wA
checking param wB
Check ::cwd::parameter_formula::check_proc passed on /hdl_libraries/CW/
components/CW_mult/parameters/wB
    Check_name    Passed Failed
--------------------------------
parameter_formula    2      0
--------------------------------
```

- The following example will check the `CW_add` component, but will skip checking its bindings and implementations:

```
rc:/> cd /hdl_libraries/CW/components/CW_add
rc:/> cwd check . -skip { /hdl_libraries/CW/components/CW_add/bindings/* \
    /hdl_libraries/CW/components/CW_add/implementations/* }
```

cwd create_check

```
cwd create_check
     -check_name string
     -severity integer
     -description string
     -checklist string
     [-effort string] [-force] [> file]
```

Registers user-defined checking rules for the CWD linter.

Options and Arguments

`-check_name` *`string`*	Specifies an unique name for a new checking rule.
`-checklist` *`string`*	Specifies, as a Tcl list of Tcl lists, checkpoint and Tcl proc pairs. The specification therefore would be in the form: `-checklist {{point_1 proc_1} {point_2 proc_2}}` Every sub-list has two elements. The first element is the name of a check point, defined by RTL Compiler. The second element is the name of a Tcl proc, defined by the user. Each sub-list specifies a check proc that is to be called at a certain check point. This check proc will be executed every time the flow reaches this check point. This Tcl list has one or more sub-lists. One checking rule can be associated with one or more check points. The check procs may or may not be allowed to parse or elaborate the HDL code of the synthesis model, depending on the effort level of this checking rule. The check procs may print out error, warning, or info messages. Each check proc should return a string whose value is either PASS or `FAIL`.
`-description` *`string`*	
	Specifies a character string that concisely describes what this checking rule examines.

`-effort` *`string`*	Specifies the effort level. There are two effort levels: `low` and `medium`. The default effort level is `low`. `Low` — The check is not allowed to parse the HDL code of the synthesis models. It can check correctness, consistency, or completeness of the CWD registration, including availability of UNIX files referred to by the `location` and `sim_model` attributes. `Medium` — The check can read, load, and parse the HDL code of the synthesis models, but it cannot elaborate or synthesize the HDL code. When exercising such a checking rule, the HDL code is automatically loaded before checking is performed.
`file`	Specifies the filename to store the output of the command.
`-force`	Removes the existing checkname and adds the current specification. Alternatively, the same checkname will now correspond to the new definition.
`-severity` *`integer`*	Specifies the severity level of the message produced by this checking rule. The possible values (`0` through `10`) are the same as those for the `information_level` attribute. `-severity 0`: It is an error message to violate this rule `-severity 1`: It is a warning message to violate this rule `-severity 2`: It is a level 2 info message to violate this rule Severity levels 3 through 10 are all info messages at their respective levels.

Example

- The following example defines the name of the check to be `arch_pin_order`. It is a medium effort level check: it has to be performed after the HDL code has been loaded. The severity of the check is `0`, which means it is an error if this check fails. This checking rule is associated with two checkpoints, `HDL_ARCH_PINS_SCANNED` and `HDL_COMP_PINS_SCANNED`. At the `HDL_ARCH_PINS_SCANNED` checkpoint, a Tcl proc named `check_arch_pin_order` is to be called to perform this check. At the `HDL_COMP_PINS_SCANNED` checkpoint, a Tcl proc named `check_component_pin_order` is to be called to perform this check.

```
rc> cwd create_check -check_name "arch_pin_order" -effort "medium" \
    -severity 0 -description "Check Whether the order of pins specified \
    in the synthesis model is consistent with what is defined in the
    registration script"
    -checklist { {HDL_ARCH_PINS_SCANNED check_arch_pin_order} \
    {HDL_COMP_PINS_SCANNED check_component_pin_order} }
```

cwd report_check

```
cwd report_check
     [-checkpoint string] [-checkname string]
     [-max_width string] [> file]
```

Reports the registered checking rules. With each checking rule, it lists the:

- Name of the checking rule
- Checkpoint(s) the rule is associated with
- Check proc(s) attached to the associated checkpoint(s)
- Effort level of the rule
- Severity level of the rule
- Description string of the rule

Note: The `-checkname` and `-checkpoint` options cannot be both used simultaneously.

Options and Arguments

`-checkname` *`string`*	Specifies, by name, a set of checking rules to report.
`-checkpoint` *`string`*	This switch specifies a set of checkpoints to report.
`-max_width` *`string`*	Limits the width of the columns in the table produced by this command. Limiting the width of a column to zero means removing that column from the table. This option takes a Tcl list of Tcl lists. Each sub-list represents a column in the table produced by this command. Each sub-list should have two elements: the first being name of the column (as seen in the report) and the second being an integer representing the maximum number of characters allowed for this column in the table.
`file`	Specifies the filename to store the output of the command.

Examples

- The following example reports details about the checking rule `arch_pin_order`, which uses two check procs to associate with two checkpoints.

```
rc:/> cwd report_check -checkname {arch_pin_order} -max_width \
    {{Check_name 14} {Checkpoint 12} {Check_proc 15} {Effort 3} \
    {Severity 4} {Description 20}}
```

```
This example reports details about the checking rule arch_pin_order, which uses
two check procs to associate with two checkpoints.
-----------------------------------------------------------------------
Check_name    Checkpoint     Check_proc   Eff Seve      Description
                                          ort rity
-----------------------------------------------------------------------
arch_pin_order HDL_ARCH_PIN ::cwd::arch_pin med   0  Check whether the
               S_SCANNED    _order::cwd_pro ium      order of pins
                            c                        specified in the
                                                     synthesis model is
               HDL_COMP_PIN ::cwd::componen          consistent with
               S_SCANNED    t_pin_order::ch          what is defined in
                            eck_proc                 the registration
                                                     script
-----------------------------------------------------------------------
```

- The following example reports details about the checkpoint `HDL_OPER_BINDINGS_SCANNED`:

```
rc:/> cwd report_check -checkpoint {HDL_OPER_BINDINGS_SCANNED}
    -max_width {{Check_name 10} {Checkpoint 15} {Check_proc 15}
    {Effort 3} {Severity 4} {Description 20}}
```

```
-----------------------------------------------------------------------
Check_name    Checkpoint      Check_proc   Eff Seve      Description
                                           ort rity
-----------------------------------------------------------------------
operator_b HDL_OPER_BINDIN ::cwd::operator low   1  check that for
indings    GS_SCANNED      _bindings::chec          every hdl_bindings
                           k_proc                   defined for the
                                                    hdl_operator there
                                                    is at least one
                                                    attribute is set to
                                                    false
-----------------------------------------------------------------------
```

- The following example reports all checking rules that have been registered:

```
rc:/> cwd report_check -checkname {*}
```

- This reports info about all checkpoints:

```
rc:/> cwd report_check -checkpoint {*}
```

- The following example reports all checking rules whose checkname contains string "pin":

```
rc:/> cwd report_check -checkname {*pin*}
```

- The following example reports information about the `arch_pin_order` and `arch_parameter_order` checking rules:

```
rc:/> cwd report_check -checkname {arch_pin_order arch_parameter_order}
```

hdl_create

```
hdl_create { binding | component | implementation | library
           | operator | package | parameter | pin}
```

Creates an HDL object for ChipWare developer.

Options and Arguments

`binding`	Creates a binding between a synthetic operator and a ChipWare component.
`component`	Creates a ChipWare component.
`implementation`	Creates a synthesis model for a ChipWare component.
`library`	Creates a synthetic library to hold ChipWare components, bindings, and implementations.
`operator`	Creates a synthetic operator.
`package`	Creates a package in the Design Information Hierarchy to hold the contents of a VHDL package.
`parameter`	Creates a parameter for a synthetic ChipWare component
`pin`	Creates an input/output/inout pin for a synthetic operator or component

Related Information

Related commands:	hdl_create binding on page 138
	hdl_create component on page 140
	hdl_create implementation on page 142
	hdl_create library on page 144
	hdl_create operator on page 145
	hdl_create package on page 146
	hdl_create parameter on page 148
	hdl_create pin on page 150

hdl_create binding

```
hdl_create binding binding_name
     -operator operator_name
     [hdl_comp | bindings]
```

Creates a binding between a synthetic operator and a synthetic module. A synthetic module is also known as a ChipWare component.

Tip

You can save run-time if you `cd` to the *`component name`* directory and issue the command instead of specifying the entire component pathname.

Options and Arguments

`binding_name`	Specifies the name of the binding that will be created.
`hdl_comp` `|` *`bindings`*	Specifies the pathname of the component that holds this binding.
`-operator`	Specifies the synthetic operator that will be bound by this binding.

Examples

- The following examples both create a binding named `test1` for the `MY_MULT_OP` operator. However, the first example will save run-time over the second by `cd`'ing into the component name directory and issuing the command.

```
rc:/hdl_libraries/my_CW/components/my_CW_mult> hdl_create binding \
    test1 -operator MY_MULT_OP
rc:/hdl_libraries/my_CW/components/my_CW_mult/bindings> ls
./      test1
```

- This example also creates a binding named `test1`. However, the command is issued from the root directory and therefore assumes a run-time penalty.

```
rc:/> hdl_create binding test1 -operator MY_MULT_OP \
    /hdl_libraries/my_CW/components/my_CW_mult
rc:/> ls /hdl_libraries/my_CW/components/my_CW_mult/bindings
./      test1
```

Related Information

Related commands:	hdl_create component on page 140
	hdl_create implementation on page 142
	hdl_create library on page 144
	hdl_create operator on page 145
	hdl_create package on page 146
	hdl_create parameter on page 148
	hdl_create pin on page 150

hdl_create component

```
hdl_create component component_name
     [hdl_lib | components]
```

Creates a ChipWare component.

Options and Arguments

component_name	Specifies the name of the component that will be created.
hdl_lib | *components*	Specifies the pathname of the library that holds this component.

Examples

- The following examples both create a component named `CW_sweet_div`. However, the first example will save run-time over the second by `cd`'ing into the `components` directory and issuing the command:

  ```
  rc:/hdl_libraries/CW/components> hdl_create component CW_sweet_div
  rc:/hdl_libraries/CW/components> ls
  ...
  CW_sweet_div
  ...
  ```

- This example also creates a component named `CW_sweet_div`. However, the command is issued from the root directory and therefore assumes a run-time penalty:

  ```
  rc:/> hdl_create component CW_sweet_div /hdl_libraries/CW/
  rc:/> ls /hdl_libraries/CW/components
  ...
  CW_sweet_div
  ...
  ```

Related Information

Related commands:	hdl_create binding on page 138
	hdl_create implementation on page 142
	hdl_create library on page 144
	hdl_create operator on page 145
	hdl_create package on page 146
	hdl_create parameter on page 148
	hdl_create pin on page 150

hdl_create implementation

```
hdl_create implementation implementation_name
     [-v1995 | -v2001 | -vhdl87 | -vhdl93]
     [hdl_comp | implementations]
```

Creates an implementation for a ChipWare component. All implementations created with this command have a default priority of `1`. A ChipWare implementation is also known as an architecture of the component. You must specify a language version.

Options and Arguments

`implementation_name`

Specifies the name of the implementation that will be created.

`hdl_comp` `|` *`implementation`*

Specifies the pathname of the component that owns this implementation.

`[-v1995 | -v2001 | -vhdl87 | -vhdl93]`

Specifies the language version for the RTL code.

Example

- Both of the following examples create the `krystal` implementation in VHDL 1993 format for the `CW_sweet_div` component. However, the first example will save run-time over the second by `cd`'ing into the component name directory and issuing the command:

```
rc:/hdl_libraries/CW/components/CW_sweet_div> hdl_create implementation \
    krystal -vhdl93
```

```
rc:/hdl_libraries/CW/components/CW_sweet_div/implementations> ls
```

```
./      krystal
```

- This example also creates an implementation named `krystal` for the same component. However, the command is issued from the root directory and therefore assumes a run-time penalty:

```
rc:/> hdl_create implementation krystal -vhdl93 \
    /hdl_libraries/CW/components/CW_sweet_div
```

```
rc:/> ls /hdl_libraries/CW/components/CW_sweet_div/implementations/
```

```
./      krystal
```

Related Information

Affects this attribute:	priority
Related commands:	hdl_create binding on page 138
	hdl_create component on page 140
	hdl_create library on page 144
	hdl_create operator on page 145
	hdl_create package on page 146
	hdl_create parameter on page 148
	hdl_create pin on page 150

hdl_create library

```
hdl_create library library_name
```

Creates an HDL library. An HDL library can be a library of ChipWare components, a library of synthetic operators, or a VHDL library.

Options and Arguments

library_name	Specifies the name of the library that will be created.

Related Information

Related commands:	hdl_create binding on page 138
	hdl_create component on page 140
	hdl_create implementation on page 142
	hdl_create operator on page 145
	hdl_create package on page 146
	hdl_create parameter on page 148
	hdl_create pin on page 150

hdl_create operator

```
hdl_create operator operator_name
     [-signed | -unsigned]
```

Creates a synthetic operator. The default operator type is unsigned.

Options and Arguments

operator_name	Specifies the name of the synthetic operator that will be created.
`-signed`	Specifies the created operator to be a signed operator.
`-unsigned`	Specifies the created operator to be an unsigned operator. This is the default setting.

Related Information

Related commands:	hdl_create binding on page 138
	hdl_create component on page 140
	hdl_create implementation on page 142
	hdl_create library on page 144
	hdl_create package on page 146
	hdl_create parameter on page 148
	hdl_create pin on page 150

hdl_create package

```
hdl_create package pkg_name
     -path path_to_pkg
     [hdl_lib | packages]
```

Registers a VHDL package in the ChipWare Developer framework. Packages that are not registered are deleted after elaboration. However, registered packages are never deleted and their information can be further considered during synthesis as opposed to just during elaboration.

Registered packages are in the same location within RTL Compiler as non-registered packages:

`/hdl_libraries/library_name/packages/`

Options and Arguments

hdl_lib | *packages*	
	Specifies the pathname of the library that holds this package.
`-path` *path_to_pkg*	Specifies the UNIX pathname of the package to register.
pkg_name	Specifies the name of the package that will be created.

Examples

- Both of the following examples create the `numeric_std` package for the `ieee` library. However, the first example will save run-time over the second by `cd`'ing into the library name directory and issuing the command:

  ```
  rc:/hdl_libraries/ieee/packages> hdl_create package numeric_std -path \
      /home/krystal/vhdl/packages/numeric_std.vhdl
  rc:/hdl_libraries/ieee/packages> ls
  ./      numeric_std
  ```

- This example also creates a package named `numeric_std` for the same library. However, the command is issued from the root directory and therefore assumes a run-time penalty:

  ```
  rc:/> hdl_create package numeric_std -path /home/krystal/vhdl/packages \
      /hdl_libraries/ieee/packages/numeric_std
  rc:/> ls /hdl_libraries/ieee/packages/
  ./      numeric_std
  ```

Related Information

Related commands:	hdl_create binding on page 138
	hdl_create component on page 140
	hdl_create implementation on page 142
	hdl_create library on page 144
	hdl_create parameter on page 148
	hdl_create pin on page 150

hdl_create parameter

```
hdl_create parameter parameter_name
     [-hdl_invisible]
     [hdl_comp | parameters]
```

Creates a parameter for a ChipWare component. The created parameter will be a `hdl_param` object type and located under `../`*`component_name/`*`parameters`. The default `hdl_parameter` attribute value for parameters created with this command will be `true`. However, if the `-hdl_invisible` option is specified, the default value becomes `false`.

Options and Arguments

`hdl_comp | parameters`	
	Specifies the pathname of the component that holds this parameter.
`-hdl_invisible`	Specifies that the parameter cannot be accessed from the HDL. The value of the `hdl_parameter` attribute for this parameter becomes `false` with this option.
`parameter_name`	Specifies the name of the parameter that will be created.

Examples

- Both of the following examples create the `WIDTH` parameter for the `CW_sweet_div` component. However, the first example will save run-time over the second by `cd`'ing into the component name directory and issuing the command:

```
rc:/hdl_libraries/CW/components/CW_sweet_div> hdl_create parameter WIDTH
rc:/hdl_libraries/CW/components/CW_sweet_div/parameter> ls
./      WIDTH
```

- This example also creates a parameter named `WIDTH` for the same component. However, the command is issued from the root directory and therefore assumes a run-time penalty:

```
rc:/> hdl_create parameter WIDTH /hdl_libraries/CW/components/CW_sweet_div
rc:/> ls /hdl_libraries/CW/components/CW_sweet_div/parameters/
./      WIDTH
```

Related Information

Affects this attribute:	hdl_parameter
Related commands:	hdl_create binding on page 138
	hdl_create component on page 140
	hdl_create implementation on page 142
	hdl_create library on page 144
	hdl_create operator on page 145
	hdl_create package on page 146
	hdl_create pin on page 150

hdl_create pin

```
hdl_create pin pin_name
     {-input | -output | -inout}
     [pins | hdl_oper | hdl_comp]
```

Creates a pin for either a ChipWare component or a synthetic operator. You must specify a pin direction.

Options and Arguments

`-inout`	Specifies that the created pin will be an bidirectional pin.
`-input`	Specifies that the created pin will be an input pin.
`-output`	Specifies that the created pin will be an output pin.
`pin_name`	Specifies the name of the pin that will be created.
`pins` `|` *`hdl_oper`* `|` *`hdl_comp`*	Specifies the pathname of the component or synthetic operator that holds the created pin.

Examples

- Both of the following examples create the `div_in` input pin for the `CW_sweet_div` component. However, the first example will save run-time over the second by `cd`'ing into the component name directory and issuing the command:

  ```
  rc:/hdl_libraries/CW/components/CW_sweet_div> hdl_create pin -input div_in
  rc:/hdl_libraries/CW/components/CW_sweet_div/pins/> ls
  ./      div_in
  ```

- This example also creates an input pin named `div_in` for the same component. However, the command is issued from the root directory and therefore assumes a run-time penalty:

  ```
  rc:/> hdl_create pin -input div_in
  rc:/> ls /hdl_libraries/CW/components/CW_sweet_div/pins/
  ./      div_in
  ```

Related Information

Related commands:	hdl_create binding on page 138
	hdl_create component on page 140
	hdl_create implementation on page 142
	hdl_create library on page 144
	hdl_create operator on page 145
	hdl_create package on page 146
	hdl_create parameter on page 148

5

Input and Output

decrypt

```
decrypt [-keydb path] file
```

Decrypts and evaluates a Tcl file that was encrypted with the `encrypt` command.

Options and Arguments

`file`	Specifies the name of the Tcl file to decrypted and evaluated.
`-keydb` *`path`*	Sets the `NCPROTECT_KEYDB` environment variable to the directory containing the public key needed to decrypt the file. If you omit this option, you need to set the `NCPROTECT_KEYDB` environment variable before you run the `decrypt` command.

Example

The first command encrypts the Tcl file `my_script.g`. The second command decrypts the `my_script_encr.g` file. These commands are normally executed in different RC sessions.

```
rc:/> encrypt -tcl my_script.g > my_script_encr.g
rc:/> decrypt my_script_encr.g
```

Related Information

Related command:	encrypt on page 156

encrypt

```
encrypt inputfile_name
     [-vlog | -vhdl | -tcl] [-pragma]
     [> file]
```

Uses the NC Protect protection scheme to encrypt the specified HDL or Tcl files.

Tip

To load encrypted HDL files, use the `read_hdl` command.

The command to source an encrypted Tcl file depends on the file extension of the encrypted Tcl file:

- ❑ If the encrypted file does not have the `.etf` extension, use the `decrypt` command.
- ❑ If the encrypted file has the `.etf` extension, you can use the `source` command.

Options and Arguments

`input_file_name`	Specifies the file to be encrypted.
`file`	Specifies the name of the encrypted file. By default, the encrypted file is printed to standard out.
`-pragma`	Specifies to only encrypt the text between the `protect begin` and `protect end` NC Protect pragmas.
`-tcl`	Specifies that the file to be encrypted is a Tcl file. To hide the body of the Tcl scripts, make sure to code procedures using `hidden_proc`.
`-vhdl`	Uses VHDL style comments for NC Protect pragmas.
`-vlog`	Uses Verilog style comments for NC Protect pragmas. This is the default option.

Example

- ■ The following example encrypts the `ksable.vhdl` VHDL file, with VHDL constructs, to a file named `ksable_encrypted.vhdl`. The encrypted file is then loaded.

```
rc:/> encrypt -vhdl ksable.vhdl > ksable_encrypted.vhdl
rc:/> read_hdl -vhdl ksable_encrypted.vhdl
```

- The following example illustrates Verilog code with Verilog style NC Protect pragmas. You must specify `//pragma protect` before specifying the beginning (`//pragma protect begin`) and ending (`//pragma protect end`) pragmas.

  ```
   module secret_func (y, a, b);
      parameter w = 4;
      input [w-1:0] a, b;
      output [w-1:0] y;
  // pragma protect
  // pragma protect begin
      assign y = a & b;
  // pragma protect end
  endmodule
  ```

  Specifying the `-vlog` and `-pragma` options together will only encrypt the text between the pragmas. The following command encrypts the original verilog file (`ori.v`) that contained the NC Protect pragmas. The encrypted file is called `enc.v`.

  ```
  rc:> encrypt -vlog -pragma org.v > enc.v
  ```

- The following example illustrates VHDL code with VHDL style NC Protect pragmas. You must specify `--pragma protect` before specifying the beginning (`--pragma protect begin`) and ending (`--pragma protect end`) pragmas.

  ```
  entity secret_func is
      generic (w : integer := 4);
      port (  y: out bit_vector (w-1 downto 0);
          a, b: in bit_vector (w-1 downto 0)      );
  end;

  -- pragma protect
  -- pragma protect begin
  architecture rtl of secret_func is
  begin
      y <= a and b;
  end;
  -- pragma protect end
  ```

  Specifying the `-vhdl` and `-pragma` options together will only encrypt the text between the pragmas. The following command encrypts the original VHDL file (`ori.vhdl`) that contained the NC Protect pragmas. The encrypted file is called `enc.vhdl`:

  ```
  rc:/> encrypt -vhdl -pragma org.vhdl > enc.vhdl
  ```

- The following example shows a Tcl script (`test.tcl`) with two procedures: the first procedure starting with `proc`, the second one starting with `hidden_proc`. When the script is encrypted, no info will be returned for the `im_hidden` procedure.

  ```
  # pragma protect
  # pragma protect begin
  proc im_visible {args} {
      # ’info’ command will return data for this proc
  }
  hidden_proc im_hidden {args} {
      # ’info’ command will not return data for this proc
  }
  # pragma protect end
  ```

```
rc:/> encrypt -tcl test.tcl > test_enc.tcl
rc:/> info body im_hidden
rc:/> info body im_visible

# ’info’ command will return data for this proc
```

Related Information

Related command:	decrypt on page 155

export_critical_endpoints

```
export_critical_endpoints
     -rc_file string -fe_file string
     [-group | -no_group] [-verbose]
     [-percentage_of_endpoints integer]
     [-no_of_bins integer]
     [-percentage_difference integer] [-rtl]
     [-design string] [> file]
```

Generates a path adjust file, which allows synthesis to provide better timing closure results to Encounter.

Options and Arguments

`-design` *`string`*	Specifies the module name.
`-fe_file` *`string`*	Specifies the First Encounter (FE) slack report that you want to compare.
`file`	Specifies the name of the file to write the report.
`[-group|-nogroup]`	Specifies whether to groups endpoints into bins for path_adjust or not. *Default*: `-group`
`-no_of_bins` *`integer`*	Specifies the number of bins to group the endpoints for compression. *Default:*10 bins each for tighten and relax
`-percentage_difference` *`integer`*	Specifies the percentage difference between the endpoints to be path adjusted (with the path_adjust command). *Default:* 70%
`-percentage_of_endpoints` *`integer`*	Specifies the percentage of endpoints to be constrained or relaxed. *Default:* 20%
`-rc_file` *`string`*	Specifies the RTL Compiler endpoint report that you want to compare.

`-rtl`	Writes a path adjust file that can be applied to the RTL.
`-verbose`	Specifies a verbose report.

Related Information

Path Adjust Flows in *Encounter RTL Compiler Synthesis Flows*

read_config_file

Refer to read_config_file in Chapter 10, “Quality Analyzer.”

read_cpf

Refer to read_cpf in Chapter 13, “Advanced Low Power Synthesis.”

read_def

Refer to read_def in Chapter 9, “Physical.”

read_dfm

`read_dfm` *`coefficients_filename`*

Loads the coefficients file. You can only load one file at a time. After the coefficients file is loaded, RTL Compiler will annotate the defect probability of any matching cells between the coefficients file and the timing library.

Options and Arguments

`coefficients_filename`

Specifies the name of the coefficients file.

Example

- A DFM file is described in XML format. The following example shows what a DFM file might look like:

```
<?xml version="1.0"?>

<yield_file>
<title> file with probabilities of failure of each library cell </title>

<cell_probability>

  <cell> inv1
    <instance>   0.000000026309750 </instance>
    <systematic> 0.000000000000000 </systematic>
  </cell>

  <cell> fflopd
    <instance>   0.000000153055338 </instance>
    <systematic> 0.000000000000000 </systematic>
  </cell>
  <cell> nand2
    <instance>   0.000000044800000 </instance>
    <systematic> 0.000000000000000 </systematic>
  </cell>

</cell_probability>

</yield_file>
```

- The following example loads two coefficient files:

```
rc:/> read_dfm test1.dfm
rc:/> read_dfm test2.dfm
```

Related Information

Design For Manufacturing Flow in *Encounter RTL Compiler Synthesis Flows*

Affects these commands:	report gates -yield
	report yield
Affects this attribute:	yield
Related attribute:	optimize_yield

read_dft_abstract_model

Refer to read_dft_abstract_model in Chapter 11, “Design for Test.”

read_encounter

Refer to read_encounter in Chapter 9, “Physical.”

read_hdl

```
read_hdl file_list
     [ {-v1995 | -v2001 | -sv | -vhdl }
       [-library library_name[=library_name2]...]
     | -netlist]
     [-define macro=value]... file_list
```

Loads one or more HDL files in the order given into memory. Files containing macro definitions should be loaded before the macros are used. Otherwise, there are no ordering of constraints.

If you do not specify either the `-v1995`, `-v2001`, `-sv` or the `-vhdl` option, the default language format is that specified by the `hdl_language` attribute. The *default* value for the `hdl_language` attribute is `-v1995`.

The HDL files can contain structural code for combining lower level modules, behavioral design specifications, or RTL implementations.

You can automatically read in or write out a compressed HDL file in gzip format. For example:

```
read_hdl sample.v.gz
write_hdl -g sample.v.gz
```

When you load a parameterized Verilog module or VHDL architecture, each parameter in the module or architecture will be identified as an `hdl_param` object and located under `../architecture_name/parameters`. The default `hdl_parameter` attribute value for these parameters will be `true`.

Use the `rc -E -f <`*`your script`*`>` command to specify that RTL Compiler automatically quit if a script error is detected when reading in HDL files instead of holding at the `rc>` prompt.

Options and Arguments

`-define` *`macro`*`=`*`value`*	Defines a Verilog macro with the specified *`value`*, which is equivalent to the `‘define` *`macro value`*. **Note:** You can also define a macro definition list.

`file_list`	Specifies the name of the HDL files to load. If several files must be loaded, specify them in a string. **Note:** The files can be encrypted. The host directory where the HDL files are looked for is specified via the `hdl_search_path` root attribute.
`-library` *`library_name`*`[=`*`library_name2`*`]...`	Specifies the name of the Verilog or VHDL library in which the definitions will be stored. A virtual directory with this name will be created in the `hdl_libraries` directory of the design hierarchy if it does not already exist. If you specify multiple libraries, they become multiple names (aliases) of one library. In this case, separate the names with the equal sign (=). Only one of the library names in the list becomes a virtual directory in the `hdl_libraries` directory. The library definitions remain in effect until elaboration, after which all library definitions are deleted. By specifying Verilog and VHDL library names, you can read in multiple Verilog modules and VHDL entities (and VHDL packages) with the same name without overwriting each other. See Examples. **Note:** You can type `-lib` or `-library`.
`-netlist`	Reads structural input files when parts of the input design is in the form of a structural netlist. You can read partially structural files provided the structural part of the input design is in the form of structural Verilog-1995 constructs and is contained in separate files from the non-structural (RTL) input. See Reading a Partially Structural Design in *Using Encounter RTL Compiler* for detailed information on using the `-netlist` option to read and elaborate a partially structural design. **Note:** If this option is specified, all the following options are ignored: `-v1995`,`-v2001`,`-vhdl`, `-sv.`
`-sv`	Specifies that the HDL files conform to SystemVerilog 3.1.a.
`-v1995`	Specifies that the HDL files conform to Verilog-1995.

`-v2001`	Specifies that the HDL files conform to Verilog-2001.
`-vhdl`	Specifies that the HDL files are VHDL files. The `hdl_vhdl_read_version` root attribute value specifies the standard to which the VHDL files conform. *Default*: VHDL-1993

Examples

- The following example first loads the `example1.v` file, then the `example2.v` file:

  ```
  rc:/> read_hdl {example1.v example2.v}
  ```

- The following commands with macro definitions are equivalent:
  - `read_hdl -define "A B=4 C"`
  - `read_hdl -define A -define B=4 -define C ...`

- The following command loads a single VHDL file and specifies a single VHDL library.

  ```
  read_hdl -vhdl -library lib1 test1.vhdl
  ```

- The following commands read structural Verilog files when the design includes RTL (VHDL or Verilog) files:

  ```
  read_hdl file1_bhv.vhdl
  read_hdl file2_bhv.v
  read_hdl -netlist file3_str.v
  elaborate
  ```

- In the following command, the `-v1995` option is ignored. Both `rtl.v` and `struct.v` are parsed in the structural mode.

  ```
  read_hdl -v1995 rtl.v -netlist struct.v
  ```

- The following command defines VHDL libraries `lib1`, `lib2` as aliases for `lib3.`

  ```
  read_hdl -vhdl -library lib1=lib2=lib3 test1.vhdl
  ```

- The following commands read in two Verilog files that each contain a Verilog module with the same name (`compute`) but with different functionality. To store both definitions, the `-lib` option indicates in which library to store the definition.

  ```
  read_hdl -v2001 -library lib1 test_01_1.v
  read_hdl -v2001 -library lib2 test_01_2.v
  ```

  Inspection of the design hierarchy shows:

  ```
  rc:/> ls /hdl_libraries/
  ```

```
/hdl_libraries:
./              DP/             DW04/           GB/               STD/            lib2/
AMBIT/          DW01/           DW05/           GTECH/            SYNERGY/        synthetic/
CADENCE/        DW02/           DW06/           IEEE/             SYNOPSYS/
```

```
CW/               DW03/             DWARE/            IEEE_SYNERGY/      lib1/
```

```
rc:/> ls /hdl_libraries/lib1/architectures/
/hdl_libraries/lib1/architectures:
./                  compute/
rc:/> ls /hdl_libraries/lib2/architectures/
/hdl_libraries/lib2/architectures:
./                  compute/
```

Related Information

Reading a Partially Structural Design in *Using Encounter RTL Compiler.*

Affects this command:	elaborate on page 284
Related command:	read_netlist
Affects this attribute:	hdl_parameter
Affected by these attributes:	hdl_search_path
	hdl_language
	hdl_preserve_dangling_output_nets
	hdl_verilog_defines

read_io_speclist

Refer to read_io_speclist in Chapter 11, “Design for Test.”

read_netlist

```
read_netlist file_list
     [-top top_module_name]
     [-define macro=value]...
```

Reads and elaborates a Verilog 1995 structural netlist when the design does not include behavioral (VHDL or Verilog) modules.

A structural Verilog file contains only structural Verilog-1995 constructs, such as module and gate instances, concurrent assignment statements, references to nets, bit-selects, part-selects, concatenations, and the unary ~ operator. Using the `read_hdl -netlist` command uses less memory and runtime to load a structural file than the `read_hdl` command.

Use the `read_netlist` command to read and elaborate a design and create a generic netlist that is ready to be synthesized. You do *not* need to use the `elaborate` command

Use the `read_hdl -netlist` command to read in a design that *includes* behavioral (VHDL or Verilog) files.

Options and Arguments

`-define` *`macro=value`*

Defines a Verilog macro with the specified *`value`*, which is equivalent to the `‘define` *`macro value`*.

`file_list`

Specifies the name of the HDL files to load. If several files must be loaded, specify them in a string.

The host directory where the HDL files are looked for is specified via the `hdl_search_path` root attribute.

`-top` *`top_module_name`*

Specifies the top-level structural Verilog module to be read and elaborated.

If you do not specify this option and multiple top-level modules are found in the loaded netlist, the tool randomly selects one of them and deletes the remaining top-level modules.

Related Information

Reading and Elaborating a Structural Netlist Design and Reading a Partially Structural Design in *Using Encounter RTL Compiler.*

Related Commands:	read_hdl `-netlist`
Affected by these attributes:	hdl_preserve_dangling_output_nets

read_saif

Refer to read_saif in Chapter 12, “Low Power Synthesis.”

read_sdc

```
read_sdc file
     [-stop_on_errors] [-no_compress]
     [-mode mode_name]
```

Reads a constraints file in Synopsys Design Constraint (SDC) format into RTL Compiler. RTL Compiler creates a cost group for each clock defined in the file. It does not create false paths between these clocks.

You must first elaborate the design before you can read the design constraints.

If you use the `read_sdc` command for loading a subset of timing constraints that includes native RTL Compiler commands, such as adding exceptions (`path_delays`), the `write_sdc` command will write these exceptions out in the SDC file.

After using the `read_sdc` command, if you make hierarchy changes in RTL Compiler using the `ungroup` or `group` commands, and there are pin specific constraints, then the `write_sdc` command will reflect the change in the hierarchy. For example, if you have constraints on the hierarchy pins you have ungrouped, then the constraints are moved to buffers (that are automatically inserted by RTL Compiler when ungrouping). The SDC file will have constraints reflecting these buffers.

Unsupported Constraints

Not all SDCs are supported. For those that are not supported, RTL Compiler will issue a warning message but store them for output for the `write_sdc` command. RTL Compiler will only store the SDCs and not manipulate any data with them.

The following SDCs are not supported:

```
set_max_area
set_propagated_clock
set_scan_style
set_signal_type
set_test_methodology
set_wire_load_min_block_size
get_references
set_data_check
get_reference
set_connection_class
set_critical_range
set_fix_multiple_port_nets
set_local_link_library
```

Options and Arguments

`file`	Specifies the name constraints file to read. You can also specify a file that was compressed with gzip (`.gz` extension).
`-mode` *`mode_name`*	Reads mode specific constraints for a design.
`-no_compress`	Turns off advanced compression.
`-stop_on_errors`	Stops reading the remainder of the script if an error is encountered during reading of the SDC file.

Related Information

Applying Design Constraints in *Using Encounter RTL Compiler*

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affects this command:	synthesize on page 294
	create_mode on page 237

read_spef

Refer to read_spef in Chapter 9, “Physical.”

read_tcf

Refer to read_tcf in Chapter 12, “Low Power Synthesis.”

read_vcd

Refer to read_vcd in Chapter 12, “Low Power Synthesis.”

restore_design

```
restore_design
     -db_dir string -design_name design
     [-def file] [-worst_corner string]
```

Loads the database written out by the Encounter® tool in the RTL Compiler tool.

Options and Arguments

`-db_dir` *`string`*	Specifies the path to the Encounter database directory.
`-def` *`file`*	Specifies the path to the DEF file. If this option is not specified, the tool searches for a *`design`*`.def` or *`design`*`.def.gz` file in the Encounter database directory. If neither file is found, an error message is issued.
`-design_name` *`design`*	Specifies the name of the design. The design name is the base filename for the output files generated by Encounter tool.
`-worst_corner` *`string`*	Specifies the worst case delay corner of all corners defined in the `viewdefinitions.tcl` file. This option is required for multi-mode multi-corner designs. **Note:** The libraries and captable for the worst corner will be loaded in the RTL Compiler tool.

Example

The following command reads the Encounter database from the `fe_db_dat` directory, specifies that the name of the design is `test`. The assumptions are that a `test.def` or `test.def.gz` file is part of the `fe_db_dat` directory and that the design is a non-MMMC design.

```
restore_design -db_dir fe_db_1.dat -design test
```

Related Information

Affects this command:	synthesize on page 294

write_atpg

Refer to `write_atpg` in Chapter 11, "Design for Test."

write_bsdl

Refer to `write_bsdl` in Chapter 11, “Design for Test.”

write_compression_macro

Refer to write_compression_macro in Chapter 11, “Design for Test.”

write_config_template

Refer to write_config_template in Chapter 10, “Quality Analyzer.”

write_def

See write_def on page 462 in Chapter 9, “Physical.”

write_design

```
write_design
     [-basename string] [-gzip_files]
     [-encounter] [design]
```

Generates all the files needed to reload the session in RTL Compiler (for example, `.g`, `.v`. and `.tcl` files). If you want to generate all the files that are need to loaded in both a RTL Compiler and Encounter® session, use the `-encounter` option.

Options and Arguments

`-basename` *`string`*	Specifies the path and basename for the generated files.
`design`	Specifies the top-level design in RTL Compiler.
`-encounter`	Generates the additional files needed for Encounter.
`-gzip_files`	Compresses the generated files in gzip format.

Example

- The following example writes both the RTL Compiler and Encounter files as well as specifies the path and basename to be `test/top`:

```
rc:/> write_design -encounter -basename test/top

unix> ls /home/mydir/test
top.conf
top.g
top.rc_setup.tcl
top.v
top.enc_setup.tcl
top.mode
top.sdc
```

Related Information

Generating Design and Session Information in *Using Encounter RTL Compiler*

Saving and Restoring a Session in RTL Compiler in *Using Encounter RTL Compiler*

write_dft_abstract_model

Refer to write_dft_abstract_model in Chapter 11, “Design for Test.”

write_do_ccd

```
write_do_ccd {compare_sdc | generate | propagate | validate}
```

Translates RTL Compiler settings to Conformal's Constraint Designer (CCD) commands for the *Validate* and *Generate* flows. In the *Validate* flow, by default the command compares the SDC to the RTL

For more information, see Validating and Generating Constraints in *Interfacing Between RTL Compiler and Conformal.*

Options and Arguments

`compare_sdc`	Generates a dofile to compare two SDC files.
`generate`	Generates a dofile for the *Generate* flow.
`propagate`	Generates a dofile to create a chip-level SDC file.
`validate`	Generates a dofile for the *Validate* flow.

Related Information

Related commands:	write_do_ccd compare_sdc on page 191
	write_do_ccd generate on page 192
	write_do_ccd propagate on page 193
	write_do_ccd validate on page 195

write_do_ccd compare_sdc

```
write_do_ccd compare_sdc
     [-design string] [-logfile file
     -golden_sdc file -revised_sdc file
     [-detail ] [-netlist file] [-no_exit] [> file]
```

Writes a dofile for the Encounter® Conformal® Constraint Designer (CCD) tool to compare two SDC files and report any differences between the two files.

Options and Arguments

`-design` *`string`*	Specifies the top-level design in RTL Compiler.
`-detail`	Requests a detailed comparison report.
`file`	Specifies the file to which the report must be written.
`-golden_sdc` *`file`*	Specifies the UNIX path to the golden SDC file.
`-logfile` *`file`*	Specifies the name of the CCD logfile. You must specify the UNIX path to the file.
`-netlist` *`file`*	Specifies the UNIX path to the netlist. By default, the tool uses the RTL.
`-no_exit`	Suppresses the `exit` command at the end of the dofile.
`-revised_sdc` *`file`*	Specifies the UNIX path to the revised SDC file.

Example

The following command compares the `test.sdc` and `revised.sdc` files.

```
write_do_ccd compare_sdc -golden_sdc test.sdc -revised_sdc revised.sdc
```

Related Information

Comparing SDC Constraint Files in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

Related command: compare_sdc on page 322

write_do_ccd generate

```
write_do_ccd generate [-design string] [-logfile string]
     [-netlist string] [-slack integer] [-report string]
     -in_sdc string [-out_sdc string] [-trv] [-fpgen]
     [-dfpgen] [-no_exit] [> file]
```

Writes a dofile for the Encounter ® Conformal ® Constraint Designer (CCD) tool for the *Generate* flow, which generates additional false paths based on critical path timing reports.

Options and Arguments

`-design` *`string`*	Specifies the top-level design in RTL Compiler.
`file`	Redirects all the output to the specified file.
`-dfpgen`	Generates a dofile for the directed false path flow.
`-fpgen`	Generates a dofile for the false path flow.
`-in_sdc` *`string`*	Specifies the list of SDC files to load.
`-logfile` *`string`*	Specifies the name of the CCD logfile.
`-netlist` *`string`*	Specifies the UNIX path to the netlist. This option compares the SDC to the specified netlist instead of the RTL.
`-out_sdc` *`string`*	Specifies the filename to which the identified false paths will be written.
`-report` *`string`*	Specifies the name of the timing report file to be generated. The report will be in CCD format.
`-sdc` *`string`*	Specifies the list of SDC files.
`-slack` *`integer`*	Specifies the slack value in picoseconds. Only paths below this slack value will be used to generate the timing report. This option should be used with the `-report` option.
`-trv`	Generates a dofile for the timing report validation flow.

Related Information

Affected by this attribute:	wccd_threshold_percentage

write_do_ccd propagate

```
write_do_ccd propagate
     -block_sdc string
     [-glue_sdc string]
     [-partial_chip_sdc string]
     [-out_sdc string] [-netlist string]
     [-rule_instance_file string]
     [-rule_instance_template string]
     [-no_exit]
     [-design design] [-logfile string] [> file]
```

Generates a dofile for the Encounter® Conformal® Constraint Designer (CCD) tool to propagate block-level constraints to the top-level and integrate them with the glue constraints to generate a chip-level SDC file.

Options and Arguments

`-block_sdc` *`string`*	Specifies a list of block names with their associated block-level SDC files in the following format: `{{block_name block_sdc_file}...}`
`-design` *`string`*	Specifies the top-level design in RTL Compiler.
`file`	Specifies the file to which the report must be written.
`-glue_sdc` *`string`*	Specifies the name of a glue SDC file. This file contains a set of constraints for the top-level module only (without covering any block-level constraints).
`-logfile` *`string`*	Creates a separate CCD logfile. You must specify the UNIX path to the file.
`-netlist` *`string`*	Specifies the UNIX path to the netlist. By default, the tool uses the RTL.
`-no_exit`	Suppresses the `exit` command at the end of the dofile.
`-out_sdc` *`string`*	Specifies the name of the constraints file that is generated after propagation and integration of the block and glue constraints. *Default*: `chip.sdc`
`-partial_chip_sdc` *`string`*	Specifies the name of the partial SDC file that corresponds to the top-level of the design.

`-rule_instance_file` *`string`*

Specifies the name of the file which defines the rule instances for the Encounter ® Conformal ® Constraint Designer (CCD) tool.

`-rule_instance_template` *`string`*

Creates a template integration rule instances file.

You can modify this file according to the design. To use this file as input for the Encounter ® Conformal ® Constraint Designer (CCD) tool, specify the file as value of the `-rule_instance_file` option.

Example

- The following command creates a do file to propagate the block-level SDC files `i1.sdc` and `i2.sdc` to the top level.

```
rc:/> write_do_ccd propagate -block_sdc {{i1 i1.sdc} {i2 i2.sdc}} \
-out_ sdc ./my_chip.sdc  rule_instance_file my_rules -logfile ccd.log
```

  The generated dofile will be similar to:

```
read library -statetable -liberty ./slow.lib

add search path -design .

read design -verilog ./ti1.v -lastmod -noelab

elaborate design

dofile ./my_rules

read hierarchical sdc \
-sdc_design i1 i1.sdc \
-sdc_design i2 i2.sdc

set system mode verify

integrate -all ./chip.sdc -replace

report rule check

report environment
```

write_do_ccd validate

```
write_do_ccd validate [-design string] [-logfile string]
     [-netlist string] -sdc string
     [-init_sequence_file string] [-no_exit]
     [> file]
```

Writes a Conformal Constraint Designer (CCD) dofile for the *Validate* flow, which validates the constraints and false path exceptions.

Options and Arguments

`-design` *`string`*	Specifies the top-level design in RTL Compiler.
`file`	Redirects all the output to the specified file.
`-init_sequence_file` *`string`*	Specifies the UNIX path to the initialization sequence file for multi-cycle path validation.
`-logfile` *`string`*	Specifies the name of the CCD logfile.
`-netlist` *`string`*	Specifies the UNIX path to the netlist. This option compares the SDC to the specified netlist instead of the RTL.
`-no_exit`	Suppresses the `exit` command at the end of the dofile.
`-report` *`string`*	Specifies the name of the timing report file to be generated. The report will be in CCD format.
`-sdc` *`string`*	Specifies the list of SDC files.

write_do_clp

```
write_do_clp
     [-design design] [-netlist string ]
     [-logfile file] [-env_var string]
     [-add_iso_cell string] [-clp_out_report string]
     [-ignore_ls_htol] [-no_exit] [-verbose]
     [-tmp_dir string] [> file]
```

Writes the required dofile for Conformal Low Power (CLP). This command will issue an error and will not proceed if you have multiple Common Power Format (CPF) files.

Options and Arguments

`-add_iso_cell` *`string`*	
	Specifies the standard cells that CLP should recognize as isolation cells.
`-clp_out_report` *`string`*	
	Writes the output of the `report rule check` Encounter® Conformal® Equivalence Checking command to this specified file.
`-design` *`design`*	Specifies the top-level design in RTL Compiler.
`-env_var` *`string`*	Specifies the names and values of UNIX environment variables to be used in the library, design, and logfile names in the generated dofile.
`file`	Redirects all the output to the specified file.
`-ignore_ls_htol`	Indicates whether to ignore the high to low level shifter check. If this option is specified, the following CLP directive will be added to the dofile: `set lowpower option -ignore_high_to_low`
`-logfile` *`file`*	Specifies the name of the CLP logfile.
`-netlist` *`path`*	Specifies the UNIX path that contains the netlist containing all the design's low power features (for example, level shifters, isolation cells and SRPG flops). *Default*: RTL

`-no_exit`	Indicates whether to skip the `exit` command at the end of the dofile. If this option is specified, the following command will be omitted from the dofile: `exit -force`
`-tmp_dir` *`string`*	Specifies the name of the directory to which the generated files must be written. Its contents will be CLP native commands. *Default*: `RC_CLP_`*`design_name`*`_out.do`
`-verbose`	Indicates whether the generated dofile will be verbose. If this option is specified, the `report rule check` command should write out the intermediate dofile for CLP and then include that file.

Example

- The following Encounter Conformal Equivalence Checking commands is an example of a CLP dofile:

```
set log file log_file_name -replace
set lowpower option -netlist_style logical

read library -statetable -liberty bn65lplvt_121a/tcbn65lplvtwcl0d90d72.lib \
read design -verilog -sensitive netlist.v

read cpf file cpf_file_name

analyze power domain
rep rule check ISO* LSH* RET* -verbose
exit -force
```

Related Information

Affected by these attributes: wclp_lib_statetable

write_do_lec

```
write_do_lec [-top string]
     [-golden_design string] [-revised_design string]
     [-sim_lib string] [-sim_plus_liberty]
     [-logfile string] [-env_var string]
     [-hier] [-flat] [-no_exit]
     [-save_session string] [-tmp_dir string]
     [-verbose] [> file]
```

Translates RTL Compiler settings to Encounter® Conformal® Equivalence Checking commands.

This command works with the Common Power Format (CPF) flow: if it detects a CPF file then it will output this information to the Conformal LEC dofile.

Options and Arguments

`-env_var` *`string`*	Specifies the names and values of UNIX environment variables to be used in library, design, and log filenames in the dofile.
`file`	Redirects all the output to the specified file.
`-flat`	Performs a flattened Conformal LEC comparison.
`-golden_design` *`string`*	
	Specifies the UNIX path to an alternative golden design.
	If the file was loaded into RTL Compiler (using either `read_hdl` or `read_netlist`), the tool knows the language format of the file.
	Otherwise, the tool assumes that the format of the file is Verilog-1995.
`-hier`	Performs a hierarchical Conformal LEC comparison.
`-logfile` *`string`*	Specifies the name of the Conformal LEC logfile.
`-no_exit`	Does not add the `exit` command to the end of the dofile.
`-revised_design` *`string`*	
	Specifies the UNIX path to the revised design.

`-save_session` *`string`*	
	Specifies the filename to save the LEC session.
`-sim_lib` *`string`*	Specifies the simulation library in Verilog 1995.
`-sim_plus_liberty`	Specifies that the simulation library is an addition to the synthesis library.
`-tmp_dir` *`string`*	Specifies the name of the directory to which the generated files must be written.
`-top` *`string`*	Specifies the name of the top-level design in RTL Compiler.
`-verbose`	Generates a dofile with verbose reporting.

Related Information

Interfacing with Encounter Conformal Logical Equivalence Checker in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

Affected by these attributes:	boundary_optimize_invert_hier_pins
	wlec_add_noblack_box_retime_subdesign
	wlec_analyze_abort
	wlec_analyze_setup
	wlec_auto_analyze
	wlec_compare_threads
	wlec_cut_point
	wlec_hier_comp_threshold
	wlec_lib_statetable
	wlec_set_cdn_synth_root
	wlec_skip_iso_check_hier_compare
	wlec_skip_lvl_check_hier_compare
	wlec_uniquify
	wlec_use_lec_model

write_do_verify cdc

```
write_do_verify cdc -sdc string
     [-design string]
     [-categorize | -validate] [-no_exit]
     [-logfile string] [> file]
```

Generates a dofile for Conformal Extended Clock Domain Crossing Checks. Use the dofile for either the Categorization or Validate flows. In the Categorization flow, no synchronization rules are defined. RTL Compiler automatically identifies and categorizes the clock domain crossing paths. In the Validate flow, you define the synchronization rules that specify the valid synchronization structures in the design.

Options and Arguments

`[-categorize	-validate]`	
	Specifies whether to generate a dofile for the *Categorization* flow or the *Validate* flow.
`-design` *`string`*	Specifies the name of the top-level design in RTL Compiler.
`file`	Specifies a specific dofile filename.
`-logfile` *`string`*	Specifies a specific logfile name.
`-no_exit`	Suppresses the `exit` command in the dofile.
`-sdc` *`string`*	Specifies a list of the SDC files.

Examples

- The following example writes a dofile for the Categorization flow:

```
rc:/> write_do_verify cdc -sdc /home/test/general.sdc -logfile my.log \
    -categorize
```

- The following example writes a dofile for the Validate flow:

```
write_do_verify cdc -sdc /home/test/general.sdc -logfile my.log \
    -validate
```

Related Information

Interfacing with Encounter Conformal Extended Checks in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

Affected by these attributes:	wcdc_clock_dom_comb_propagation
	wcdc_synchronizer_type

write_encounter

```
write_encounter design [-basename string]
     [-gzip_files] [-reference_config_file config_file]
     [-ignore_scan_chains] [-ignore_msv]
     [-floorplan string] [-lef lef_files] [design]
```

Writes Encounter® input files to a single directory. The command will only convert library domains into power domains for Encounter if power domains exist in RTL Compiler. If power domains do not exist, the `-ignore_msv` option is implied. The command also supports the Common Power Format (CPF) files by directly passing them to Encounter.

The generated files are all required Encounter input files and include the following files:

- Netlist (`.v`)
- Encounter configuration file (`.conf`),
- SDC constraints (`.sdc`)
- Tcl script (`.enc_setup.tcl`)
- Mode file (`.mode`)
- Scan DEF file (`.scan.def`)
- MSV-related files (`.msv.tcl`, `.msv.vsf`)
- Multiple timing mode (`.mmode.tcl`)

The `.enc_setup.tcl` file can simultaneously load all the necessary Encounter data in an Encounter session. This eliminates the need to load each of the necessary files sequentially.

The `.mode` file contains all the Encounter `setMode` settings. For example, the file would contain the `setAnalysisMode` and `setPlaceMode` settings.

The full DEF file that is outputted is the exact same DEF file that was loaded or generated by `predict_qos`. However, RTL Compiler generates the information for the Scan DEF file (`.scan.def`).

Note: The MSV (multiple supply voltage) library domain setup commands require Encounter version 4.2 or later. Multiple timing mode is supported in Encounter version 5.2 or later.

Options and Arguments

`-basename` *`string`*	Specifies the directory pathname and base filename for the output data.The default directory is `./rc_enc_des` and the default filename without the extension is `rc`.
`design`	Specifies a particular design for which to write out information. Only one design can be specified at a time.
`-floorplan` *`string`*	Specifies the extension for the file containing the floorplan. The valid extensions are: `.def`—DEF `.pde`—PDEF `.fp`—Encounter floorplan
`-gzip_files`	Compresses the netlist and constraints in `.gz` format. The floorplan, if one was read, will be untouched. That is, if it was read in uncompressed, it will be outputted uncompressed and vice versa.
`-ignore_scan_chains`	If specified, the scan DEF file will not be written and the scan reorder directives will not be included in the setup file.
`-ignore_msv`	If specified, the MSV setup file and the shifter table file will not be written out. This option is useful if the library domains in RTL Compiler are not being used for modeling power domains.
`-lef` *`lef_files`*	Specifies a particular physical library or libraries to use. The physical libraries will have the `.lef` extension. The contents of the LEF library that was specified with the `lef_library` attribute will be used if this option is not specified.
`-reference_config_file` *`config_file`*	Specifies a reference Encounter configuration file to use as a template for the generated configuration file.

Examples

- The following example writes all the Encounter input files for the `test07` design to the directory TEST. The basename for the files are specified to be `test1`.

  ```
  rc:/> write_encounter design test07 -basename TEST/test1

  unix> ls TEST/
  test1.conf   test1.enc_setup.tcl   test1.mode   test1.sdc   test1.v
  ```

- The following example compresses the netlist and constraints with the `-gzip_files` option:

  ```
  rc:/> write_encounter design -gzip_files

  unix> ls rc_enc_des
  rc.conf  rc.def  rc.enc_setup.tcl  rc.mode  rc.sdc.gz  rc.v.gz
  ```

Related Information

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Export to Place and Route in The Multiple Supply Voltage Flow in *Low Power in Encounter RTL Compiler*.

Related commands:	create_mode on page 237
	read_encounter on page 167

write_et_atpg

Refer to write_et_atpg in Chapter 11, “Design for Test.”

write_et_bsv

Refer to write_et_bsv in Chapter 11, “Design for Test.”

write_et_mbist

Refer to write_et_mbist in Chapter 11, “Design for Test.”

write_et_rrfa

Refer to write_et_rrfa in Chapter 11, “Design for Test.”

write_ets

```
write_ets [-default] [-ocv]
     [-pre_include string] [-post_include string]
     [-netlist string]
     [-sdc string] [-sdf string] [-spef string]
     [> file]
```

Generates an Encounter® Timing System (ETS) run script.

Options and Arguments

`-default`	Generates a simple ETS run script. The script will contain the following ETS commands: `read_lib`, `read_verilog`, `set_top_module`, `read_sdc`, and `report_timing`.
`file`	Redirects the output to the specified file.
`-netlist` *`string`*	Specifies the UNIX path of the file containing the gate-level netlist.
`-ocv`	Adds an extra `set_timing_derate` command into the ETS run script. This option can only be specified with the `-sdf` option.
`-post_include` *`string`*	Specifies the UNIX path of the include file that contains ETS commands that need to be added after the `report_timing` command in the run file generated by `write_ets`.
`-pre_include` *`string`*	Specifies the UNIX path of the include file that contains ETS commands that need to be added before the `report_timing` command in the run file generated by `write_ets`.
`-sdc` *`string`*	Specifies the UNIX path of the SDC file.
`-sdf` *`string`*	Specifies the UNIX path of the SDF file. If this option is specified, the `read_sdf`, `set_analysis_mode`, and `set_op_cond` commands will be added to the ETS run script.
`-spef` *`string`*	Specifies the UNIX path of the SPEF file. If this option is specified, the `read_spef`, `set_analysis_mode`, and `set_op_cond` commands will be added to the ETS run script.

write_ett

```
write_ett
     [-strict | -dc | -ett]
     [-version {1.1|1.3|1.4}]
     [design] [> file]
```

Generates constraints for Encounter® True Time.

Some constraints (such as `set_input_delay`, `set_output_delay`, and so on) are written out in SDC (Synopsys Design Constraints) format while others (such as `set_false_path`, `set_disable_timing`) are written out in Encounter test format.

Options and Arguments

design	Specifies the design for which the constraints must be generated.
file	Specifies the name of the file to which the constraints must be written.
`-dc`	Writes the constraints that are DC and PT compatible, which means that commands not listed in the SDC specification may be written out.
`-ett`	Writes an Encounter True Time Clock Constraints file.
`-strict`	Writes only constraints in SDC format.
`-version {1.1|1.3|1.4}`	Specifies the SDC version to use to write SDC constraints.

write_forward_saif

Refer to write_forward_saif in Chapter 12, “Low Power Synthesis.”

write_hdl

```
write_hdl  {design|subdesign}...
     [-suffix string]
     [-abstract] [-generic] [-depth integer]
     [-equation] [-lec] [ > file]
```

Generates one of the following design implementations in Verilog format:

- A structural netlist using generic logic
- A structural netlist using mapped logic

You can automatically read in or write out a gzip compressed Verilog file. For example:

```
read_hdl sample.v.gz
write_hdl > sample.v.gz
```

Options and Arguments

`-abstract`	Generates an empty top-level Verilog module definition of the specified design or subdesign that defines the I/O pins and bit-width for all top-level functional and scan-related ports in the design or subdesign. This empty module description is further referred to as *logic abstract model*.
`-depth` *`integer`*	Specifies the number of hierarchy levels to be written out, starting from the top level. A value of 0, writes out only the top-level module. *Default*: infinite
{*`design`* | *`subdesign`*}	Specifies the design or subdesign for which the design implementation must be generated.
`-equation`	Writes out a logic equation in an assign statement for each Verilog primitive gate.
`file`	Specifies the file to which the output must be written. *Default*: Output is written to the screen.

`-generic`	Generates an unoptimized generic logic implementation of the design that uses the generic logic gates specified within the Verilog language. Any parts of the design that are mapped will be unmapped for the `write_hdl` command without affecting the design in memory. Once the `synthesize` command has been run, you cannot recover the version of the design that was generated using this option prior to synthesis.
`-lec`	Generates additional information to facilitate formal verification with Encounter® Conformal® Equivalence Checking.
`-suffix`	Specifies the string to be appended to the name of all defined modules in the generated netlist.

Examples

- The following example writes out the logic abstract model definition of design `test`:

```
rc:/> write_hdl -abstract

// Generated by Cadence RTL Compiler-D (RC) version

module test(in1, in2, out1, out2, clk1, clk2, clk3, se1, se2);
  input [3:0] in1;
  input [7:0] in2;
  input clk1, clk2, clk3, se1, se2;
  output [3:0] out1;
  output [7:0] out2;
endmodule
```

- The following example writes out the design as generic logic regardless of its current mapped state:

```
rc:/> write_hdl -generic > design_rtl.v
```

- You can write out a netlist for a specific module. For example, the following commands writes out the `middle` module:

```
rc:/> set_attr unresolved true [get_attr instance [get_attr subdesign bottom]]
rc:/> write_hdl [find / -subdesign middle]
```

- The following example writes out a design that instantiates cells from the target technology library reflecting the current state of the design (mapped state):

```
rc:/> read_hdl design.v
rc:/> ...
rc:/> synthesize -to_mapped
rc:/> ...
rc:/> write_hdl
```

- The following example replaces each Verilog primitive gate by an equivalent Verilog assign statement:

  ```
  rc:/> write_hdl -equation design.v
  ```

- The following example writes out the design as generic logic regardless of its current mapped state:

  ```
  rc:/> write_hdl -generic > design_rtl.v
  ```

Related Information

Writing Out the Design Netlist in *Using Encounter RTL Compiler*.

Affects this command:	synthesize on page 294
Affected by these commands:	elaborate on page 284
	synthesize on page 294
Affected by these attributes:	write_sv_port_wrapper
	write_vlog_bit_blast_bus_connections
	write_vlog_bit_blast_constants
	write_vlog_bit_blast_mapped_ports
	write_vlog_bit_blast_tech_cell
	write_vlog_convert_onebit_vector_to_scalar
	write_vlog_declare_wires
	write_vlog_empty_module_for_logic_abstract
	write_vlog_line_wrap_limit
	write_vlog_no_negative_index
	write_vlog_port_association_style
	write_vlog_preserve_net_name
	write_vlog_top_module_first
	write_vlog_unconnected_port_style
	write_vlog_wor_wand

write_io_speclist

Refer to write_io_speclist in Chapter 11, “Design for Test.”

write_saif

Refer to write_saif in Chapter 12, “Low Power Synthesis.”

write_scandef

Refer to `write_scandef` in Chapter 11, “Design for Test.”

write_script

```
write_script [-hdl]
     [-analyze_all_scan_chains [-dont_overlay_segments]]
     [design] [> file]
```

Generates a script that contains the timing for all modes and the design rule constraints of the design. If you used DFT functionality, the script will also contain any test constraints that were applied, as well as any objects that were created in the design as a result of inserting DFT logic such as boundary scan, building the fullscan chains, and inserting scan chain compression logic.

The resulting script can subsequently be used to examine the current design constraints, or it can be read back into RTL Compiler to perform analysis or optimization at a later time.

The `write_script` command can also compress the output using the gzip (`.gz` extension).

The script contains the following:

- The attributes connected with the `wire_load` models
- Clock objects and their reference to the pins of the design blocks
- `External_delay` on all inputs and outputs
- Timing exceptions
- `max_fanout` / `max_capacitance` and similar design rule constraints applied
- All user defined attributes that were created with the `define_attribute` command

The script can also include DFT constraints or commands, such as:

- DFT constraints created (or tool inferred) using any of the following:
  `define_dft shift_enable, define_dft test_mode, define_dft test_clock, define_dft scan_chain`
- DFT constraints created with `set_attribute dft_dont_scan`
- DFT objects created by the user or by the tool with `set_attribute user_defined` and `set_attribute dft_auto_created`
- `check_dft_rules`

Note: The `write` command writes out only the design itself while the `write_script` command writes out the constraints for the design.

Options and Arguments

`-analyze_all_scan_chains`

Writes out all chains in the `dft/report/actual_scan_chains` directory using the following notation:

`define_dft scan_chain -name` *`name`*`... -sdo` *`sdo`* `-analyze`

When running the script in a new RTL Compiler session, the RC-DFT engine analyzes the existing scan chains (traces the connectivity of the chains) and restores this information into the `dft/report/actual_scan_chains` directory.

`design` Specifies the name of the design for which to write a script.

`-dont_overlay_segments`

Adds the `-dont_overlay` option to the scan chains it is writing out with the `-analyze` option.

Note: You can only specify this option when you specify the `-analyze_all_scan_chains` option.

`file` Specifies the name of the file to which to write the constraints.

`-hdl` Writes out the architecture/entity filename information to the output file.

Examples

- The following example saves the design and its constraints:

```
rc:/> write_hdl > mapped.v
rc:/> write_script > mapped.g
```

The design and script is subsequently read into another RTL Compiler session. You must specify any .lib, LEF, or cap table files: these files are process specific as opposed to design specific and therefore are not automatically loaded.

```
rc:/> set_attribute library areid.lib
rc:/> set_attribute lef_library areid.lef
rc:/> set_attribute cap_table_file areid.cap
rc:/> read mapped.v
rc:/> elaborate
rc:/> source mapped.g
```

- The following example automatically compresses the output file using the .gz extension:

```
rc:/> write_script > foo.g.gz
```

Related Information

Affected by these commands:	create_mode on page 237
	define_clock on page 240
	define_cost_group on page 245
	define_dft scan_chain on page 560
	external_delay on page 248
	multi_cycle on page 254
	path_adjust on page 259
	path_delay on page 263
	path_disable on page 266
	path_group on page 269

write_sdc

```
write_sdc [-version {1.1|1.3|1.4|1.5|1.5rc}]
     [-strict] [-mode mode_name] [-no_split] [design] [> file]
```

Writes out the current design constraints in Synopsys Design Constraint (SDC) format. The `write_sdc` command can also compress the SDC constraints with gzip (`.gz` extension).

When using the `write_sdc` command, RTL Compiler replaces the / character with the @ character when the `/` character is used in the name of objects. This could happen when the design is ungrouped or when the / character is used as the `ungroup_separator`.
To prevent this problem, write out the constraints using an SDC version less than 1.3 to avoid the hsc specification. For example: `rc:/> write_sdc -version 1.1`.

For those SDCs that are not supported, RTL Compiler will issue a warning message but store them for output for the `write_sdc` command. RTL Compiler will only store the SDCs and not manipulate any data with them.

Note: Using the `write_sdc` command may not capture all the design information necessary to recreate the image of a design's constraints.

Options and Arguments

`design`	Specifies the name of the design for which to write the SDC constraints.
`file`	Specifies the name of the file to which to write the SDC constraints.
`-mode` *`mode`*	Writes out mode specific constraints for a design.
`-no_split`	Prevents printing the SDC commands over several lines.
`-strict`	Writes out commands that are specifically listed in the SDC specification. If you do not use this option, the `write_sdc` command outputs commands that are DC and PT compatible, which means that commands not listed in the SDC specification may be written out. See Examples for the difference in results when using the `-strict` option.
`-version {1.1|1.3|1.4|1.5|1.5rc}`	Specifies the SDC version to use. Version `1.5rc` includes the `set_time_unit` and `set_load_unit` commands.

Examples

- The following example writes out the SDC constraints to the `my_des.sdc` file:

  ```
  rc:/> write_sdc /designs/my_des > my_des.sdc
  ```

- The following example shows the results you may get if you do not specify the `-strict` option with the `write_sdc` command.

  ```
  ######################################################################
   ...
  ######################################################################

  set sdc_version 1.4
  # Set the current design
  current_design add

  set_wire_load_mode "enclosed"
  set_wire_load_selection_group "ALUMINUM" -library "tutorial"
  set_dont_touch [get_designs Madd_addinc]
  set_dont_touch [get_cells flop1]
  ```

- The following example shows the results you may get if you do specify the `-strict` option with the `write_sdc` command.

  ```
  ####################################################################
    ...
  #####################################################################

  set sdc_version 1.4

  # Set the current design
  current_design add

  set_wire_load_mode "enclosed"
  set_wire_load_selection_group "ALUMINUM" -library "tutorial"
  ```

- The following example shows how to write out mode-specific constraints:

  ```
  write_sdc -mode mode1 mode1.sdc
  ```

- The following examples show the effect of the `-no_split` option.

  Assume that the `write_sdc` command would write out the following command in the SDC file:

  ```
  set_false_path -from [list \
      [get_clocks in1]  \
      [get_clocks in2] ] -to [get_clocks in3]
  ```

  If you specify the `write_sdc` command with the `-no_split` option, the SDC command would be written as:

  ```
  set_false_path -from [list [get_clocks in1] [get_clocks in2] ] -to [get_clocks in3]
  ```

Related Information

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affected by this command:	create_mode on page 237
	read_sdc on page 176

write_sdf

```
write_sdf [-version {OVI 3.0 | OVI 2.1}]
     [-precision non_negative_integer]
     [-timescale {ps | ns}] [-delimiter character]
     [-celltiming {all | none | nochecks}]
     [-interconn {port | interconnect [-no_empty_cells]}]
     [-edges {edged | check_edge}] [-condelse]
     [-nonegchecks] [-no_escape] [-nosplit_timing_check]
     [-no_input_port_nets] [-no_output_port_nets]
     [-design] [> file}
```

The command generates a Standard Delay Format (SDF) file that analysis and verification tools or timing simulation tools can use for delay annotation. The SDF file specifies the delay of all the cells and interconnects in the design in the Standard Delay Format. Specifically, it includes the delay values for all the timing arcs of a given cell in the design.

Note: Use the `write_sdf` command after technology mapping (after the `synthesize -to_mapped` command).

Options and Arguments

`-celltiming {all | none | nochecks}`	
	Specifies which cells delays and timing checks to write out. `all`—Writes all cell delays and timing checks to the SDF file. `none`—Excludes cell delays and timing checks from being written into the SDF file. `nochecks`—Only excludes the timing checks. *Default*: `all`
`-condelse`	Writes `CONDELSE` constructs with the default value when a `COND` construct is written.
`-delimiter` *`character`*	
	Specifies the hierarchy divider character to be used in the SDF file. The valid options are the “`/`” and “`.`” characters.
`-design`	Specifies the design name for which the SDF file has to be generated.

`-edges {edged | check_edge}`

Specifies the edges values.

- `check_edge`—Keeps edge specifiers on timing check arcs but does not add edge specifiers on combinational arcs.
- `edged`—Keeps edge specifiers on timing check arcs as well as combinational arcs.

Default: `edged`

`file` Specify the SDF file name.

`-interconn {port | interconnect}`

Specifies the construct to use for writing out net delays.

`port`—Writes out the net delays using the `PORT` construct.

`interconnect`—Writes out the net delays using the `INTERCONNECT` construct.

Default: `port`

`-no_escape` Writes out object names without escaping the special characters such as “`[`“ or “`]`”.

`-nonegchecks` Converts all negative timing check values to `0.0`.

`-no_empty_cells`

Suppresses writing out empty cell descriptions.

`-no_input_port_nets`

Suppresses writing out nets connected to input ports.

`-no_output_port_nets`

Suppresses writing out nets connected to output ports.

`-nosplit_timing_check`

Does not split the `TIMINGCHECK` delays (`SETUP/HOLD/RECOVERY/REMOVAL` delays). Instead, the maximum delay values are used.

`-precision` *`non_negative_integer`*

Specifies the number of digits appearing after the decimal point in the output SDF file.

`-timescale {ps | ns}`

Specifies the timescale setting of the SDF file in either nanoseconds or picoseconds.

Default: `ps`.

`-version {OVI 3.0 | OVI 2.1}`

Specifies whether to generate SDF version 2.1 or 3.0.

Default: OVI 3.0.

Examples

- The following example writes out the SDF file, `areid.sdf`, with the "/" delimiting character:

```
rc:/> synthesize -to_mapped
rc:/> write_sdf -delimiter "/" > areid.sdf
```

- The following report illustrates two `TIMINGCHECK examples: the first only writes out the maximum delays while the second writes out all the delays.`

```
(TIMINGCHECK
(HOLD D (posedge CK) (::0.0))
(SETUP D (posedge CK) (::0.452))
)

(TIMINGCHECK

(HOLD (negedge D) (posedge CK) (::0.0))
(HOLD (posedge D) (posedge CK) (::0.0))
(SETUP (negedge D) (posedge CK) (::0.452))
(SETUP (posedge D) (posedge CK) (::0.181))
)
```

To write out only the maximum `TIMINGCHECK` delays (the first report above), use the `-nosplit_timing_check` option:

```
rc:/> write_sdf -nosplit_timing_check > areid.sdf
```

write_set_load

`write_set_load [`*`design`*`] [> `*`file`*`]`

Generates a set of load values, which were obtained from the physical layout estimator (PLE) or wire-load model, for all the nets in the specified design. This command is useful when performing timing correlation across various synthesis and timing tools. The `write_set_load` command can be used to reproduce PLE based timing in external timing analyzers.

Options and Arguments

`design`	Generates the load values for the specified design.
`file`	Specifies the file to which to write the load values.

Example

- The following example shows that the load values on all the nets is .0103:

```
rc:/> write_set_load
set_load  0.0103 [get_nets inst1/out1[3]]
set_load  0.0103 [get_nets inst1/out1[2]]
set_load  0.0103 [get_nets inst1/out1[1]]
set_load  0.0103 [get_nets inst1/out1[0]]
```

write_spef

See write_spef in Chapter 9, “Physical.”

write_tcf

Refer to write_tcf in Chapter 12, “Low Power Synthesis.”

write_template

```
write_template
     [-dft] [-power] [-cpf] [-retime] [-physical]
     [-area] [-no_sdc] [-n2n] [-yield]
     [-full] [-simple] [-split]
     [-multimode] -outfile string
```

Generates a template script with the commands and attributes needed to run RTL Compiler. Use the command options to include specific commands and attributes in the script.

Options and Arguments

`-area`	Writes out a template script for area-critical designs.
`-checkpoint`	Writes out a template for the (old) checkpoint flow.
	Note: The basic template created with the `write_template` command contains the necessary commands and attribute settings to verify the synthesized netlist with the Encounter® Conformal® Logical Equivalence Checker (LEC) tool. Use the `-checkpoint` option *only* if you need to revert to the *old* flow.
`-cpf`	Writes out a template script for the Common Power Format (CPF) based flow and a template CPF file (`template.cpf`). The template CPF file is read in the template script with the `read_cpf` command.
`-dft`	Writes out a template script for the test synthesis (DFT) flow.
	The template contains commands and attributes needed for basic and advanced DFT features.
`-full`	Writes out DFT, power, and retiming commands and attributes along with the basic template.
`-multimode`	Writes out a template script for multi-mode analysis.
`-n2n`	Writes out the template script for netlist to netlist optimization in RTL Compiler. Use with the `-dft` and `-power` options to include the DFT and power attributes and commands.
`-no_sdc`	Writes out clock delays and input and output delays in the RTL Compiler format using the `define_clock` and the `external_delay` commands.

`-outfile` *`string`*	Specifies the name of the file to which the template script is to be written.
`-physical`	Writes out a template script for the physical flow.
`-power`	Writes out power attributes and commands.
`-retime`	Writes out retiming attributes and commands.
`-simple`	Writes out a simple template script. You cannot use this option with the `-split`, `-area`, `-dft`, `-cpf`, `-multimode`, `-physical`, `-power`, `-retime`, or `-full` options.
`-split`	Writes out a template script with a separate setup file which contains the root attributes and setup variables. If you specified the `-dft` option with the `-split` option, an additional file is created which contains the DFT design attributes, test clock and scan chain information. If you specified the `-power` option with the `-split` option, an additional file is created which contains the leakage and dynamic power, and clock-gating setup information.
`-yield`	Writes out a template script for yield.

Examples

- The following example writes out the basic template file with the constraints in SDC format:

```
rc:/> write_template -outfile template.g
```

- The following example writes out the basic template file with the constraints in the RTL Compiler format using the `define_clock` and the `external_delay` commands:

```
rc:/> write_template -no_sdc -outfile template.g
```

- The following example adds both the DFT and power related attributes to the template.g file written out:

```
rc:/> write_template -dft -power -outfile template.g
```

- The following example writes out the template script with the DFT attributes and commands for netlist to netlist optimization:

```
rc:/> write_template -dft -n2n -outfile template.g
```

- The following example writes out the template script *`template.g`* and the setup file *`setup_template.g`* that contains all the root attributes and setup variables and includes it in the *`template.g`* file:

  ```
  rc:/> write_template -split -outfile template.g
  ```

- The following example writes out the *`template.g`* template script, a `setup_`*`template.g`* setup file, a `dft_`*`template.g`* file, and a `power_`*`template.g`* file and includes them in the appropriate *`template.g`* file:

  ```
  write_template -split -dft -power -outfile template.g
  ```

- The following example writes out a simple template script with no path and cost groups and without any variables, power, or DFT related attributes:

  ```
  write_template -simple -outfile template.g
  ```

- The following example writes out a template script for area critical designs:

  ```
  write_template -area -outfile template.g
  ```

6

Constraints

clock_uncertainty

```
clock_uncertainty
     [-fall | -rise]
     [-from_edge <string>] [-to_edge <string>]
     [-from clock_list | -fall_from clock_list |
       -rise_from clock_list]
     [-to clock_list | -fall_to clock_list |
       -rise_to clock_list]
     [-hold | -setup]
     [-clock clock_list] uncertainty
     [clock|port|instance|pin]...
```

Specifies the uncertainty on the clock network. You specify either a simple or an inter-clock uncertainty.

- Simple uncertainties are defined directly on a clock, port, pin, or instance.These uncertainty values are stored in attributes.
- The inter-clock uncertainties (defined using options such as `-from`, `-to`, `-rise_from`, `-rise_to`) are modeled as `path_adjust` exceptions. These uncertainties take precedence over the simple uncertainty values.

Options and Arguments

`{`*`clock`*` | `*`pin`*` | `*`port`*` | `*`instance`*`}`

Specifies the clocks, pins, ports and instances to which the specified uncertainty value applies. In case of instances, the uncertainty is applied on the input pins of the instance.

Note: These arguments only apply to simple uncertainties.

`-clock` *`clock_list`* Specifies the clock(s) to which the uncertainty value applies.

If this option is not specified, it applies to all clocks propagating to that pin or port.

Note: This option only applies to simple uncertainties.

`-fall | -rise` Specifies to apply the uncertainty to the falling or rising edge of the capture clock pin.

If neither option is specified, the uncertainty value applies to both edges.

Note: These options only apply to inter-clock uncertainties.

`-from` *`clock_list`* `| -fall_from` *`clock_list`* `| -rise_from` *`clock_list`*

Applies the uncertainty value to the specified list of launching clocks.

Note: These options only apply to inter-clock uncertainties.

`-from_edge {rise | fall | both}`

Specifies the edge type of the launch clock.

Note: This option only applies to inter-clock uncertainties and cannot be specified with either `-fall_from` or `-rise_from`.

`-hold | -setup` Specifies that the clock uncertainty applies to hold or setup checks.

If neither option is specified, the uncertainty value applies to both checks.

Note: These options apply to both types of uncertainties.

`-to` *`clock_list`* `| -fall_to` *`clock_list`* `| -rise_to` *`clock_list`*

Applies the uncertainty value to the specified list of capture clocks.

Note: These options only apply to inter-clock uncertainties.

`-to_edge {rise | fall | both}`

Specifies the edge type of the capture clock.

Note: This option only applies to inter-clock uncertainties and cannot be specified with either `-fall_to` or `-rise_to`.

`uncertainty` Specifies the uncertainty value for the clock(s). Use a floating value.

Examples

- The following command sets a (simple) setup uncertainty of 0.4 sdc units for all paths ending at `reg1` and captured by the rising transition of all clocks propagating to `reg1/CK`.

```
clock_uncertainty -setup -rise 0.4 [find / -pin reg1/CK]
```

- The following command sets a (simple) setup uncertainty of 0.4 sdc units for all paths ending at `reg1` and captured by the rising transition of `clk1`.

```
clock_uncertainty -setup -rise -clock clk1 0.4 [find / -pin reg1/CK]
```

- The following command sets a (inter clock) setup uncertainty of 0.5 sdc units for all paths launched by `clk2` and captured by `clk1`.

  ```
  clock_uncertainty -setup -from clk2 -to clk1 0.5
  ```

Related Information

Affects this command:	path_adjust
Related command:	dc::set_clock_uncertainty
Sets these attributes:	clock_hold_uncertainty
	clock_setup_uncertainty
	hold_uncertainty_by_clock
	setup_uncertainty_by_clock

create_mode

```
create_mode -name mode_list
     [-default] [-design design]
```

Specifies the mode for power and timing analysis and optimization. Use this command after loading and elaborating the design, before reading in an SDC file, and before using any of the following constraints: define_clock, external_delay, multi_cycle, path_adjust, path_delay, path_disable, path_group, and specify_paths.

After creating modes, use the `-mode` option with the `read_sdc` command.

The command returns the directory path to the `mode` object that it creates. You can find the objects created by the `create_mode` command in:

```
/designs/design/modes
```

Options and Arguments

`-default`	Designates the specified mode as the default mode.
	In this case, you can specify only mode with the `-name` option.
`-design` *design*	Specifies the name of the design for which you want to create a mode.
`-name` *mode_list*	Specifies the names of the modes to be created. Unless a mode is assigned to be the default mode using the `-default` option, the first mode specified will be the default.

Examples

- The following example creates multiple modes for one design using one `create_mode` command:

  ```
  rc:/>create_mode -name "mode1 mode2" -design design_name
  ```

- The following example creates two modes using multiple `create_mode` commands, and declares one of them as the default mode.

  ```
  rc:/> create_mode -name a
  /designs/design_name/modes/a
  rc:/>create_mode -name b -default
  /designs/design_name/modes/b
  ```

- The following example illustrates the recommended flow.

```
read_hdl test.v
elaborate
create_mode -name {a b}
read_sdc -mode a a.sdc
read_sdc -mode b b.sdc
```

Related Information

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler* for detailed information.

Affects these commands:	define_clock on page 240
	external_delay on page 248
	multi_cycle on page 254
	path_adjust on page 259
	path_delay on page 263
	path_disable on page 266
	path_group on page 269
	specify_paths on page 274
	read_sdc on page 176
	report clocks on page 347
	report timing on page 437
	write_sdc on page 221
Related commands:	derive_environment on page 246
	report summary on page 435
	write_encounter on page 202
	write_script on page 218
Sets this attribute:	default

Related attributes:	(instance) disabled_arcs_by_mode
	(pin/port) external_delays_by_mode
	(design/instance) latch_borrow_by_mode
	(design/instance/pin) latch_max_borrow_by_mode
	(pin/port) propagated_clocks_by_mode
	(design/pin/port/cost group) slack_by_mode
	(pin/port) timing_case_computed_value_by_mode
	(instance) timing_case_disabled_arcs_by_mode
	(pin/port) timing_case_logic_value_by_mode

define_clock

```
define_clock -name string
     -period integer [-divide_period integer]
     [-rise integer] [-divide_rise integer]
     [-fall integer] [-divide_fall integer]
     [-domain string] [-mode mode_name]
     [-design design] [pin|port]...
```

Defines a clock waveform. A clock waveform is a periodic signal with one rising edge and one falling edge per period. The command returns the directory path to the clock object that it creates.

Note: Clock waveforms that are not applied to objects in your design are referred to as “external” clocks and are only used as references for external delay values (see the external_delay command).

Options and Arguments

`-design` *design*

Specifies the name of the top module for which you want to define a clock waveform.

This option is required for external clocks when there are multiple top designs or user netlists.

`-divide_fall` *integer*

Determines together with the `-fall` option the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a fraction of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: `100`

`-divide_period` *integer*

Determines together with the `-period` option the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: `1`

`-divide_rise` *`integer`*	Determines, together with the `-divide_rise` option, the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a fraction of the period and is derived by dividing `-rise` by `-divide_rise`. *Default*: `100`
`-domain` *`string`*	Specifies the name of the clock domain. A clock domain groups clocks that are synchronously related to each other, allowing timing analysis to be performed between these clocks. RTL Compiler only computes timing constraints between clocks in the same clock domain. Paths between clocks in different domains are unconstrained by default. To constrain these paths, use the `path_delay` command. *Default*: `domain_1`
`-fall` *`integer`*	Determines, together with the `-divide_fall` option, the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a fraction of the period and is derived by dividing `-fall` by `-divide_fall`. *Default*: `50` (falling edge is halfway through the period)
`-mode` *`mode_name`*	Defines a clock waveform for a mode.
`-name` *`string`*	Specifies the name of the clock that is being defined. Each clock object in your design must have a unique name. If you define a new clock with the same name as an existing clock, then the new clock replaces the old one. The clock name allows you to search for the clock later (through the `find` command) or to recognize it in reports. *Default*: Name of the first pin or port object specified.
`-period` *`integer`*	Determines, together with the `-divide_period` option, the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

`{`*`pin`*`	`*`port`*`}`	Specifies the clock input pin or port. You can apply clock waveforms to input ports of your design, hierarchical pins, clock pins of sequential cells in your design, a combination, or to no objects at all.
`-rise` *`integer`*	Determines together with the `-divide_rise` option the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a fraction of the period and is derived by dividing `-rise` by `-divide_rise`. *Default*: `0` (rising edge is at the start of the period)

Examples

- The following example defines a clock for a design with top module `alu`.

  ```
  rc:/> define_clock -period 10000 -name 100MHz -design /designs/alu
  ```

  The clock period is 10,000 picoseconds.

- The following example defines a 300 MHz clock that applies to all sequential logic within a design:

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

  The clock period is 10,000/3 picoseconds. This allows RTL Compiler to compute that there are exactly 3 periods of clock `300MHz` to every period of `100MHz`.

  If you had specified a period of 3,333 picoseconds for the 300 MHz clock, RTL Compiler would compute a different relationship between the clocks (3333 periods of one clock to 10000 periods of the other) and the timing analysis of the design would be different.

- The following example defines clock `100MHz` with a rising edge after 20 percent of the period and a falling edge after 80 percent:

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

- The following example defines clock `100MHz` with the falling edge1/3 and the rising edge 2/3 of the way through the period:

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

  Note: The `-divide_rise` and `-divide_fall` options allow you to precisely define when the clock transitions occur. In some cases RTL Compiler needs this precise definition to compute the correct timing constraints for paths that are launched by one clock and captured by another.

- The following examples create a clock domain `system` and assign the original 100 MHz and 300 MHz clocks to it:

  ```
  rc:/> define_clock -domain "system" -period 10000 -name 100MHz
  rc:/> define_clock -domain "system" -period 10000 -name 300MHz  \
  ==> -divide_period 3 [clock_ports]
  ```

- The following example saves the directory path to the clock that is defined in variable `clock1`:

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

  Alternatively, the `find` command can be used to perform a search at a later time.

- The following example removes the clock whose definition you saved in variable `clock1`:

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

  Note: When you remove a clock object, any external delays that reference it are also removed. Timing exceptions referring to the clock object are also removed if they can't be satisfied without the clock.

- The following example searches for a clock object by name:

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

- The following example examines the attributes of a clock object:

  ```
  rc:/> ls -a [find / -clock 100MHz]
  ```

  Note: The clock object is identified by its directory path.

Related Information

Defining the Clock Period in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

Affects these commands:	clock_ports on page 321
	external_delay on page 248
	multi_cycle on page 254
	path_adjust on page 259
	path_delay on page 263
	path_disable on page 266
	path_group on page 269
	report clocks on page 347

	report qor on page 419
	specify_paths on page 274
	read_sdc on page 176
	report clocks on page 347
	report summary on page 435
	report timing on page 437
	synthesize on page 294
	write_encounter on page 202
	write_script on page 218
	write_sdc on page 221
Related commands:	create_mode on page 237
Affects these attributes:	divide_fall
	divide_period
	divide_rise
	fall
	period
	rise
	propagated_clocks_by_mode

define_cost_group

```
define_cost_group -name string
     [-weight integer]
     [-design design]
```

Defines a cost group. The command returns the directory path to the object that it creates.

Options and Arguments

`-design` *`design`*	Specifies the name of the design for which you want to define the cost group.
`-name` *`string`*	Specifies the name of the cost group. *Default*: `grp_`*`x`*
`-weight` *`integer`*	Specifies the weight of the cost group. The higher the weight factor, the higher the effort used to optimize the paths in this group. *Default*: `1`

Examples

The following example assigns a weight factor of 1 to cost group `I2O` for the top-level design.

```
rc:/> define_cost_group -name I2O -weight 1
/designs/.../timing/exceptions/path_groups/I2O
```

Related Information

Creating Path Groups and Cost Groups in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

Affects these commands:	path_group on page 269
	report timing on page 437
	synthesize on page 294
	write_script on page 218
Sets this attribute:	weight

derive_environment

```
derive_environment [-name string] [-sdc_only] instance
```

Creates a new design from the specified instance and creates timing constraints for all modes for this design based on the timing constraints that the instance had in the original design.

This command does not perform time-budgeting. The slack at each pin in the new design matches the slack at the corresponding pin in the original design.

Note: By default, the `derive_environment` command uses RTL Compiler's more powerful constraint language that produces a more accurate timing view, but is not understood by other tools. Use the `-sdc_only` option to specify that RTL Compiler only apply constraints to the new design that can be expressed in SDC.

If you use the `derive_environment` command to generate constraints for a subdesign then try to write them out, often the constraints cannot be expressed in SDC and you will get the following error message:

```
Error   : The design contains constraints which have no SDC equivalent.[SDC-19]
        : The design is /designs/Madd_addinc.
        : If the design constraints were created using the ’derive_environment’
command, use the ’-sdc_only’ option so that only constraints that can be expressed
in SDC are generated. By default the ’derive_environment’command uses the more
powerful RC constaints, which cannot always be converted to SDC.
```

Options and Arguments

instance	Specifies the name of the instance whose environment (constraints) you want to derive.
-name *string*	Specifies the name of the target design.
-sdc_only	Specifies that the tool only apply constraints to the new design that can be expressed in SDC. Using this option makes the new constraints less accurate, but makes it possible to use the constraints in flows between different tools that support SDC. By default, the `derive_environment` command uses RTL Compiler's more powerful constraint language that produces a more accurate timing view, but is not understood by other tools.

Related Information

Affected by this command:	path_adjust on page 259
	create_mode on page 237
Affects these commands:	report timing on page 437
	synthesize on page 294
Sets this attribute:	precluded_path_adjusts

external_delay

```
external_delay
     {-input min_rise min_fall max_rise max_fall
     |-output min_rise min_fall max_rise max_fall}
     [-clock  object] [-edge_fall | -edge_rise]
     [-level_sensitive] [-accumulate]
     [-mode mode_name] [-name string] {port|pin}...
```

Constrains ports and pins within your design. Timing is specified as either an input or output delay, and is specified relative to a clock edge.

External delays are most often specified on top-level ports of your design.

Options and Arguments

`-accumulate`	Indicates that more than one external delay can be specified per clock per phase.
`-clock` *`object`*	Specifies the reference clock. Input and output delays are defined relative to this clock. For an input delay, the reference clock is called the *launching* clock. For an output delay, the reference clock is called the *capturing* clock. The reference clock must have been defined with the `define_clock` command.
`[-edge_rise | -edge_fall]`	Specifies to use the rising or falling edge of the reference clock as reference edge. *Default*: `-edge_rise`

`-input` *`min_rise min_fall max_rise max_fall`*

Specifies an input delay. That is the time between the reference edge of the launching clock and the time when the input signal at the specified ports or pins becomes stable.

You can specify one, two, or four integers. If you specify one value, that value is used for all four delay values. If you specify two values, the first value defines the (minimum and maximum) rise delays, while the second value defines the (minimum and maximum) fall delays.

The *minimum* rise (fall) delays are used for hold analysis.The *maximum* rise (fall) delays are used for setup analysis.

Note: RTL Compiler does not support hold analysis.

The (minimum and maximum) *rise* delays are measured with respect to the rising edge of the input signal.

The (minimum and maximum) *fall* delays are measured with respect to the falling edge of the input signal.

`-level_sensitive`

Used with the `-input` option, it specifies the constraint coming from a level-sensitive latch.

Used with the `-output` option, it specifies the constraint to a level-sensitive latch.

`-mode` *`mode_name`*

Constrains ports and pins by mode in a design.

`-name` *`string`*

Associates a name with the specified timing constraint. If another timing constraint already exists with that name, the existing timing constraint is replaced with the new one.

The name may be useful for later finding the constraint (with the `find` command) and for recognizing the constraint in reports.

Default: xx_*n*

`-output` *`min_rise min_fall max_rise max_fall`*

Specifies an external output delay. That is the delay between the time when the output signal at the specified ports or pins becomes stable and the reference edge of the capturing clock.

You can specify one, two, or four integers. If you specify one value, that value is used for all four delay values. If you specify two values, the first value defines the (minimum and maximum) rise delays, while the second value defines the fall delays.

The *minimum* rise (fall) delays are used for hold analysis.The *maximum* rise (fall) delays are used for setup analysis.

Note: RTL Compiler does not support hold analysis.

The (minimum and maximum) *rise* delays are measured with respect to the rising edge of the output signal.

The (minimum and maximum) *fall* delays are measured with respect to the falling edge of the output signal.

`{`*`pin`*` | `*`port`*`}`

Specifies delay for timing startpoints and endpoints, which can be primary ports, pins of sequential instances, and pins of unresolved references.

Use the `break_timing_paths` attribute to make a non-startpoint/endpoint a startpoint/endpoint, which then can be used with the `external_delay` command.

Examples

- The following example specifies an input delay of 300 picoseconds on all bits of port a relative to the falling edge of clock `clock1`:

```
rc:/> external_delay -input 300 -edge_fall -clock [find / -clock clock1] \
[find / -port a*]
```

  RTL Compiler interprets this as a worst-case upper bound constraint, that is, the latest time to set up the data on a D pin of an edge-triggered flop.

  Applying delays to individual bits of multibit ports or busses is possible since each bit of a port is accessible individually within the directory structure of the design.

- The following example specifies an output delay of 1300 picoseconds on all bits of port a relative to the rising edge (default) of clock `clock1`:

```
rc:/> external_delay -output 1300 -clock [find / -clock clock1] \
[find / -port a*]
```

  RTL Compiler interprets this as a minimum setup time for external logic.

Related Information

Setting Output Delays in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler.*

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affected by this command:	define_clock on page 240
Affects these commands:	report qor on page 419
	report timing on page 437
	synthesize on page 294
	write_encounter on page 202
	write_sdc on page 221
	write_script on page 218
Related commands:	create_mode on page 237
	define_clock on page 240
	derive_environment on page 246
	multi_cycle on page 254
	path_adjust on page 259
	path_delay on page 263
	path_disable on page 266
	path_group on page 269
	specify_paths on page 274
Affects these attributes:	External Delay Attributes
Related attributes:	(pin/port) external_delays_by_mode

generate_constraints

```
generate_constraints [-rtl] [-netlist string]
     [-slack integer] [-report string]
     [-in_sdc string] [-out_sdc string]
     [-trv] [-fpgen] [-dfpgen] [> file]
```

Verifies the false paths and multi-cycle paths in the SDC files against the RTL or netlist and then generates any missing functional false paths or multi-cycle paths. The command can be used any time after elaboration. If you do not specify either the `-rtl`, `-netlist`, or `-in_sdc` options, RTL Compiler will internally generate the SDC constraints and verify them against the design at its current state. RTL Compiler will generate any missing constraints.

Use this command in the False Path Generation, Directed False Path Generation, and Timing Report Validation flows. Refer to *The Generate Flow without Dofiles* section in the *Validating and Generating Constraints* chapter in the *Interfacing Between Encounter RTL Compiler and Conformal* guide for more information on these flows.

Options and Arguments

`-dfpgen`	Generates the false paths from Directed False Path Generation flow.
`file`	Specifies the name of the file to write the report.
`-fpgen`	Generates the false paths from False Path Generation flow.
`-in_sdc` *`string`*	Specifies the UNIX path to the SDC files.
`-logfile` *`string`*	Creates a separate CCD logfile. You must specify the UNIX path to the file.
`-netlist` *`string`*	Specifies the UNIX path to the netlist.
`-out_sdc` *`string`*	Specifies the name for the RTL Compiler generated SDC file. *Default:* `cfp.sdc`
`-report` *`string`*	Specifies the name of the timing report file to be generated for the design. The file will be in the CCD format.
`-rtl`	Indicates that the verification should happen against the RTL.
`-slack` *`integer`*	Specifies a slack value in picoseconds. Only paths below this slack value will be used for generating the timing report. This must be used with the `-report` option.

`-trv`	Generates the false paths from the Timing Report Validation flow.

Examples

- The following command generates the false path from the False Path Generation flow, verifies against the RTL, specifies the input SDC file `top.sdc`, and specifies the output SDC file `top_out.sdc`:

```
rc:/> generate_constraints -fpgen -rtl -in_sdc /home/test/top.sdc \
    -out_sdc /home/cody/top_out.sdc
```

- The following command generates the false path from the Directed False Path Generation flow, verifies against the netlist `generic.nl.v`, specifies the input SDC file `top.sdc`, specifies the slack (2 picoseconds), specifies the report name, and specifies the output SDC file `top_out.sdc`:

```
rc:/> generate_constraints -dfpgen -netlist generic.nl.v -in_sdc top.sdc \
    -report top_out.rep -slack 2 -out_sdc top_out.sdc
```

Related Information

Affected by this attribute:	wccd_threshold_percentage
Related command:	validate_constraints on page 280

multi_cycle

```
multi_cycle
     { {-from {instance|external_delay|clock|port|pin}...
       |-through {instance|port|pin}...[-through...]...
       |-to {instance|external_delay|clock|port|pin}...}...
       |-paths string}
     [-launch_shift integer] [-capture_shift integer]
     [-setup] [-hold]
     [-lenient] [-mode mode_name] [-name string]
```

Creates a timing exception object that overrides the default clock edge relationship for paths that meet the path selection criteria. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options. The command returns the directory path to the object that it creates.

RTL Compiler normally computes timing constraints for paths based on the launching clock waveforms and capturing clock waveforms. The default timing constraint is the smallest positive difference that exists between a launching clock edge and a capturing clock edge.

Options and Arguments

`-capture_shift` *`integer`*

Specifies the capture clock shift value.

An ordinary two-cycle path would be specified using `-capture_shift 2`. Incrementing the `-capture_shift` value adds a cycle to the path by shifting the capture edge one period later.

Default: `1`

`-hold` Specifies that the exception is for hold timing analysis only.

Default: `setup`

`-from` {*`instance`*|*`external_delay`*|*`clock`*|*`port`*|*`pin`*}

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which the specified external delay timing exception applies.

Only paths that start at one of the ports or pins, or paths that are launched by one of the clock objects will have the timing exception applied to them.

`-launch_shift` *`integer`*	
	Specifies the launch clock shift value. Incrementing the `-launch_shift` value adds a cycle to the path by shifting the launch edge one period earlier.
	Adjusting the launch edge is only useful if the launch and capture clocks have different periods. Otherwise an equivalent timing relationship can be achieved by shifting the capture clock instead.
	Default: `0`
`-lenient`	Converts an invalid start or endpoint into a through point and issues a warning message. Without this option, an invalid start or endpoint results in an error message.
`-mode` *`mode_name`*	Creates a timing exception object for a mode that overrides the default clock edge relationship for paths that meet the path selection criteria.
`-name` *`string`*	Associates a name with the specified timing exception. If another timing exception already exists with that name, the existing timing exception is replaced with the new one.
	The name can be useful for later finding the exception (with the `find` command) and for recognizing the exception in reports.
	Default: `mc_`*`n`*
`-paths` *`string`*	Specifies the paths to which the exception should be applied. The string argument should be created using the `specify_paths` command. The `-paths` option cannot be used in conjunction with any of the other three path selection options (`-from`, `-through`, `-to`).
`-setup`	Specifies that the exception is for setup timing analysis only.
	Default: `setup`

`-through {`*`instance`*`|`*`port`*`|`*`pin`*`}`

Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/ mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-to {`*`instance`*` |`*`external_delay`*`|`*`clock`*`|`*`port`*`|`*`pin`*`}`

Specifies a Tcl list of endpoints for the paths. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies.

Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects, have the exception applied to them.

Examples

- The following example requires a path to first pass through object a, then end on object b:

```
rc:/> multi_cycle -through a -to b
```

- The following example requires a path to pass through either object a or b.

```
rc:/> multi_cycle -through {a b}
```

- The following example requires a path to first pass through object a or `b,` then pass through object c or d.

```
multi_cycle -through {a b} -through {c d}
```

  The candidate paths would be:

```
ac      ad      bc      bd
```

- The following example requires a path that goes through object a, b, and c in that order:

```
rc:/> multi_cycle -through a -through b -through c
```

- The following example uses a Tcl variable to save the timing exception for future reference:

```
rc:/> set two_cycle [multi_cycle -capture_shift 2 -from [find / -port a]]
```

The following example removes the timing exception that you saved in the `two_cycle` variable:

```
rc:/> rm $two_cycle
```

- The following example searches for a timing exception that you defined earlier:

```
rc:/> multi_cycle -capture_shift 2 -from [find / -port a] -name two_cycle
/designs/alu/timing/exceptions/multi_cycles/two_cycle
...
...
rc:/> find / -exception two_cycle
/designs/alu/timing/exceptions/multi_cycles/two_cycle
```

- The following example lists multi cycle timing exception objects using the `ls` command.

```
rc:/> ls -l timing/exceptions/multi_cycles
```

- The following examples are equivalent and apply to paths starting from (clock) pin `clk1`:

```
multi_cycle -from clk1
```

```
multi_cycle -paths [specify_paths -from clk1]
```

Related Information

Setting Timing Exceptions in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affects these commands:	report qor on page 419
	report timing on page 437
	specify_paths on page 274
	synthesize on page 294
	write_encounter on page 202
	write_sdc on page 221
	write_script on page 218

Related commands:	create_mode on page 237
	define_clock on page 240
	external_delay on page 248
	path_adjust on page 259
	path_delay on page 263
	path_disable on page 266
	path_group on page 269
	specify_paths on page 274
Sets these attributes:	Exception Attributes

path_adjust

```
path_adjust -delay integer
     { {-from {instance|external_delay|clock|port|pin}...
       |-through {instance|port|pin}...[-through...]...
       |-to {instance|external_delay|clock|port|pin}...}...
       |-paths string }
     [-lenient] [-mode mode_name] [-name string]
```

Modifies path constraints. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options.

These path constraints could have been

- Computed by the timing engine using the launching and capturing waveforms
- Set explicitly with the `path_delay` command

The command returns the directory path to the object that it creates.

The constraints specified with the `path_adjust` command can co-exist with other timing exceptions, such as `path_delay`, `multi_cycle`, as well as `path_adjust` (it is possible to have multiple `path_adjust` constraints on a path).

The constraints created by the `path_adjust` commands can be found in:

```
/designs/../timing/exceptions/path_adjusts
```

Options and Arguments

`-delay` *`integer`*
: Specifies the delay constraint value, in picoseconds, by which the path has to be adjusted. A positive adjustment relaxes the clock constraint and a negative adjustment tightens it.

`-from {`*`instance`*`|`*`external_delay`*` |`*`clock`*`|`*`port`*`|`*`pin`*`}`
: Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which the specified external delay timing exception applies.

: Only paths that start at one of the ports or pins, or paths that are launched by one of the clock objects will have the timing exception applied to them.

`-lenient`	Converts an invalid start or endpoint into a through point and issues a warning message. Without this option, an invalid start or endpoint results in an error message.
`-mode` *`mode_name`*	Modifies path constraints for a specified mode.
`-name` *`string`*	Associates a name with the specified timing exception. If another timing exception already exists with that name, the existing timing exception is replaced with the new one. The name may be useful for later finding the exception (with the `find` command) and for recognizing the exception in reports. *Default*: `adj_`*`n`*
`-paths` *`string`*	Specifies the paths to which the exception should be applied. The string argument should be created using the `specify_paths` command. The `-paths` option cannot be used in conjunction with any of the other three path selection options (`-from`, `-through`, `-to`).
`-through {`*`instance`*`|`*`port`*`|`*`pin`*`}`	
	Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/ mapped combinational instances. You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.
`-to {`*`instance`*` |`*`external_delay`*`|`*`clock`*`|`*`port`*`|`*`pin`*`}`	
	Specifies a Tcl list of endpoints for the paths. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies. Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects, have the exception applied to them.

Examples

- The following example removes the timing exception that you saved in the `override` variable:

  ```
  rc:/> set override [path_adjust -to $clock -delay -500]
  rc:/> rm $override
  ```

- The following example searches for a timing exception that you defined earlier:

  ```
  rc:/> path_adjust -to $clock -delay -500 -name override
  /designs/alu/timing/exceptions/path_adjusts/override
  ...
  rc:/> find / -exception override
  /designs/alu/timing/exceptions/path_adjusts/override
  ```

- The following examples are equivalent and apply to paths starting from (clock) pin `clk1`:

  ```
  path_adjust -from clk1
  path_adjust -paths [specify_paths -from clk1]
  ```

Related Information

Modifying Path Constraints in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affects these commands:	report timing on page 437
	report qor on page 419
	report timing on page 437
	specify_paths on page 274
	synthesize on page 294
	write_script on page 218
	write_encounter on page 202
	write_sdc on page 221
	write_script on page 218

Related commands	define_clock on page 240
	external_delay on page 248
	multi_cycle on page 254
	path_delay on page 263
	path_disable on page 266
	path_group on page 269
	specify_paths on page 274
Sets these attributes:	Exception Attributes

path_delay

```
path_delay -delay integer
     { {-from {instance|external_delay|clock|port|pin}...
       |-through {instance|port|pin}...[-through...]...
       |-to {instance|external_delay|clock|port|pin}...}...
       |-paths string}
     [-lenient] [-mode mode_name] [-name string]
```

Creates a timing exception object that allows you to specify the timing constraint for paths that meet the path selection criteria. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options. The command returns the directory path to the object that it creates.

RTL Compiler normally computes timing constraints for paths based on the launching clock waveforms and capturing clock waveforms. The default timing constraint is the smallest positive difference that exists between a launching clock edge and a capturing clock edge.

The constraints created by the `path_delay` commands can be found in:

```
/designs/../timing/exceptions/path_delays
```

Options and Arguments

`-delay` *`integer`*

Specifies the delay constraint value, in picoseconds, for paths that meet the path selection criteria.

`-from {`*`instance`*` | `*`external_delay`*` | `*`clock`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which the specified external delay timing exception applies.

Only paths that start at one of the ports or pins, or paths that are launched by one of the clock objects will have the timing exception applied to them.

`-lenient`

Converts an invalid start or endpoint into a through point and issues a warning message. Without this option, an invalid start or endpoint results in an error message.

`-mode` *`mode_name`*

Creates a timing exception object for the specified mode that lets you specify the timing constraint for paths that meet the path selection criteria.

`-name` *`string`*

Associates a name with the specified timing exception. If another timing exception already exists with that name, the existing timing exception is replaced with the new one.

The name may be useful for later finding the exception (with the `find` command) and for recognizing the exception in reports.

Default: `del_`*`n`*

`-paths` *`string`*

Specifies the paths to which the exception should be applied. The string argument should be created using the `specify_paths` command. The `-paths` option cannot be used in conjunction with any of the other three path selection options (`-from`, `-through`, `-to`).

`-through {`*`instance`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/ mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-to {`*`instance`*` | `*`external_delay`*` | `*`clock`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of endpoints for the paths. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies.

Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects, have the exception applied to them.

Examples

■ The following example specifies a path delay of 5000 ps for all paths starting from port `a`.

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

- The following command defines path delay `my_delay` of `4000ps` for all paths ending at port `b.`

  ```
  rc:/> path_delay -delay 4000 -to [find / -port b] -name my_delay
  /designs/alu/timing/exceptions/path_delays/my_delay
  ```

  The following command searches for the timing exception `my_delay` defined earlier:

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

- The following examples are equivalent and define a path delay for all paths starting from (clock) pin `clk1`:

  ```
  path_delay -from clk1
  ```

  ```
  path_delay -paths [specify_paths -from clk1]
  ```

Related Information

Modifying Path Constraints in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affects these commands:	report qor on page 419
	report timing on page 437
	specify_paths on page 274
	synthesize on page 294
	write_script on page 218
	write_sdc on page 221
Related commands:	create_mode on page 237
	define_clock on page 240
	external_delay on page 248
	multi_cycle on page 254
	path_adjust on page 259
	path_disable on page 266
	path_group on page 269
	specify_paths on page 274
Sets these attributes:	Exception Attributes

path_disable

```
path_disable
     { {-from {instance|external_delay|clock|port|pin}...
       |-through {instance|port|pin}...[-through...]...
       |-to {instance|external_delay|clock|port|pin}...}...
       |-paths string}
     [-lenient] [-mode mode_name] [-name string]
```

Creates a timing exception object that allows you to unconstrain paths. The command returns the directory path to the object that it creates. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options.

RTL Compiler computes timing constraints for paths based on the launching clock waveforms and the capturing clock waveforms. The default timing constraint is the smallest positive difference that exists between a launching clock edge and a capturing clock edge.

Options and Arguments

`-from {`*`instance`*` | `*`external_delay`*` | `*`clock`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which the specified external delay timing exception applies.

Only paths that start at one of the ports or pins, or paths that are launched by one of the clock objects will have the timing exception applied to them.

`-lenient`	Converts an invalid start or endpoint into a through point and issues a warning message. Without this option, an invalid start or endpoint results in an error message.
`-mode` *`mode_name`*	Creates a timing exception object for the specified mode that lets you unconstrain paths.

`-name` *`string`*	Associates a name with the specified timing exception. If another timing exception already exists with that name, the existing timing exception is replaced with the new one. The name may be useful for later finding the exception (with the `find` command) and for recognizing the exception in reports. *Default*: `dis_`*`n`*
`-paths` *`string`*	Specifies the paths to which the exception should be applied. The string argument should be created using the `specify_paths` command. The `-paths` option cannot be used in conjunction with any of the other three path selection options (`-from`, `-through`, `-to`).

`-through {`*`instance`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/ mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-to {`*`instance`*` | `*`external_delay`*` | `*`clock`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of endpoints for the paths. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies.

Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects, have the exception applied to them.

Examples

- The following example uses a Tcl variable to save the timing exception for future reference:

```
rc:/> set false_path [path_disable -from [find / -port a]]
```

- The following example removes the timing exception that you saved in variable `false_path`:

```
rc:/> rm $false_path
```

- The following example lists timing exception objects using the `ls` command.

```
rc:/> ls -l timing/exceptions/path_disables
```

- The following examples are equivalent and apply to paths starting from (clock) pin `clk1`:

```
path_disable -from clk1
path_disable -paths [specify_paths -from clk1]
```

Related Information

Specifying a False Path in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affects these commands:	report qor on page 419
	report timing on page 437
	specify_paths on page 274
	synthesize on page 294
	write_encounter
	write_script on page 218
	write_sdc on page 221
Affected by these commands:	define_clock on page 240
	multi_cycle on page 254
Related commands:	create_mode on page 237
	define_clock on page 240
	external_delay on page 248
	path_adjust on page 259
	path_delay on page 263
	path_group on page 269
	specify_paths on page 274
Sets these attributes:	Exception Attributes

path_group

```
path_group
     {{-from {instance|external_delay|clock|port|pin}...
      |-through {instance|port|pin}...[-through...]...
      |-to {instance|external_delay|clock|port|pin}...}...
      |-paths string}
     [-group string]
     [-lenient] [-mode mode_name] [-name string]
```

Assigns paths that meet the path selection criteria to a cost group. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options.

Options and Arguments

`-from {`*`instance`*` | `*`external_delay`*` | `*`clock`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports.

`-group` *`string`*	Specifies the name of the cost group (defined with `define_cost_group`) to which the path is added.
`-lenient`	Converts an invalid start or endpoint into a through point and issues a warning message. Without this option, an invalid start or endpoint results in an error message.
`-mode` *`mode_name`*	Assigns paths for a specified mode that meet the path selection criteria to a cost group.
`-name` *`string`*	Associates a name with the specified timing exception.
`-paths` *`string`*	Specifies the paths to which the exception should be applied. The string argument should be created using the `specify_paths` command. The `-paths` option cannot be used in conjunction with any of the other three path selection options (`-from`, `-through`, `-to`).

`-through {`*`instance`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/ mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-to {`*`instance`*` | `*`external_delay`*` | `*`clock`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of endpoints for the paths. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports.

Examples

- The following example assigns the paths from all inputs to all outputs to the group called I2O, which was previously defined by a `define_cost_group` command:

```
path_group -from /designs/*/ports_in/* -to /designs/*/ports_out/* -group I2O
```

- The following examples are equivalent and apply to paths starting from (clock) pin `clk1`:

```
path_group -from clk1
path_group -paths [specify_paths -from clk1]
```

Related Information

Creating Path Groups and Cost Groups in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affects these commands:	define_cost_group on page 245
	report qor on page 419
	report timing on page 437
	specify_paths on page 274
	synthesize on page 294

	write_encounter on page 202
	write_script on page 218
	write_sdc on page 221
Related commands:	create_mode on page 237
	define_clock on page 240
	external_delay on page 248
	multi_cycle on page 254
	path_adjust on page 259
	path_delay on page 263
	path_disable on page 266
	specify_paths on page 274
Sets these attributes:	Exception Attributes

propagate_constraints

```
propagate_constraints
     -block_sdc string
     [-glue_sdc string]
     [-partial_chip_sdc string]
     [-out_sdc string] [-netlist string]
     [-rule_instance_file string]
     [-rule_instance_template string]
     [-logfile string] [> file]
```

Propagates the block-level constraints to the top-level and integrates them to generate a chip-level constraints. Propagating and integrating of the design constraints requires to analyze the provided block-level and glue constraints, and to check them against any existing chip-level constraints to determine whether to ignore or promote them to the top-level.

Options and Arguments

`-block_sdc` *`string`*	Specifies a list of block names with their associated block-level SDC files in the following format: `{{block_name` *`block_sdc_file`*`}...}` **Note:** You need to specify the path to the SDC files. If the paths contain Tcl variables, use the format shown in the Examples on page 273.
`file`	Specifies the file to which the report must be written.
`-glue_sdc` *`string`*	Specifies the name of a glue SDC file. This file contains a set of constraints for the top-level module only (without covering any block-level constraints).
`-logfile` *`string`*	Creates a separate CCD logfile. You must specify the UNIX path to the file.
`-netlist` *`string`*	Specifies the UNIX path to the netlist. By default, the tool uses the RTL.
`-out_sdc` *`string`*	Specifies the name of the constraints file that is generated after propagation and integration of the block and glue constraints. *Default*: `chip.sdc`
`-partial_chip_sdc` *`string`*	Specifies the name of the partial SDC file that corresponds to the top-level of the design.

`-rule_instance_file` *`string`*

Specifies the name of the file which defines the rule instances for the Encounter ® Conformal ® Constraint Designer (CCD) tool.

`-rule_instance_template` *`string`*

Creates a template with the default rules. You can modify this file according to the design. To use this file as input for the Encounter ® Conformal ® Constraint Designer (CCD) tool, specify the file as value of the `-rule_instance_file` option.

Examples

- The following command creates a top-level SDC file, `chip.sdc`, which integrates the block-level SDC files `i1.sdc` and `i2.sdc`.

  ```
  rc:/> propagate_constraints -block_sdc {{i1 i1.sdc} {i2 i2.sdc}}
  ```

  The internally generated CCD dofile will be similar to:

  ```
  read library -statetable -liberty ./tech/slow.lib
  add search path -design .
  read design -verilog ./ti1.v -lastmod -noelab
  elaborate design
  dofile ./default_rule_instances.do
  read hierarchical sdc \
  -sdc_design i1 i1.sdc \
  -sdc_design i2 i2.sdc
  set system mode verify
  integrate -all ./chip.sdc -replace
  report rule check
  report environment
  ```

- The following example shows the use of Tcl variables to specify the paths to the SDC files:

  ```
  propagate_constraints -block_sdc "{block1 $WORKINGDIR/block1.sdc} \
  {block2 $WORKINGDIR/block2.sdc} ..."
  ```

  or

  ```
  propagate_constraints -block_sdc [list \
          [list block1 ${WORKINGDIR}/block1.sdc] \
          [list block2 ${WORKINGDIR}/block2.sdc] \] ...
  ```

specify_paths

```
specify_paths [-mode mode_name] [-domain clock_domain]
     [ -from {pin|port|clock|external_delay|instance}...
     | -from_rise_clock {pin|port|clock|external_delay|instance}...
     | -from_fall_clock {pin|port|clock|external_delay|instance}...
     | -from_rise_pin {pin|port|clock|external_delay|instance}...
     | -from_fall_pin {pin|port|clock|external_delay|instance}...]
     [ -through {pin|port|instance}...
     | -through_rise_pin {pin|port|instance}...
     | -through_fall_pin {pin|port|instance}...] ...
     [ -to {pin|port|clock|external_delay|instance}... |
     | -to_rise_clock {pin|port|clock|external_delay|instance}...
     | -to_fall_clock {pin|port|clock|external_delay|instance}...
     | -to_rise_pin {pin|port|clock|external_delay|instance}...
     | -to_fall_pin {pin|port|clock|external_delay|instance}...]
     [ -capture_clock_pins pin...
     | -capture_clock_pins_rise_clock pin...
     | -capture_clock_pins_fall_clock pin...
     | -capture_clock_pins_rise_pin pin...
     | -capture_clock_pins_fall_pin pin... ]
     [-lenient]
```

Creates a string that indicates the path of a particular timing exception. You must use the `specify_paths` command as an argument to the `-paths` option in any of the timing exception commands. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these three options.

The `specify_paths` command gives you more detailed control when specifying timing exceptions than the `-from`, `-through`, `-to` options in these timing exceptions could provide.

Options and Arguments

`-capture_clock_pins` *pin*

Specifies a Tcl list of sequential clock pins that capture data. This option can be useful with complex sequential cells such as RAMs that have multiple clock pins.

`-capture_clock_pins_fall_clock` *pin*

Specifies a Tcl list of sequential clock pins that capture data at the falling edge of the ideal clock waveform. This option can be useful with complex sequential cells such as RAMs that have multiple clock pins.

`-capture_clock_pins_fall_pin` *`pin`*

Specifies a Tcl list of sequential clock pins that capture data at the fall transitions of the specified sequential clock pin. This option can be useful with complex sequential cells such as RAMs that have multiple clock pins.

`-capture_clock_pins_rise_clock` *`pin`*

Specifies a Tcl list of sequential clock pins that capture data at the rising edge of the ideal clock waveform.

`-capture_clock_pins_rise_pin` *`pin`*

Specifies a Tcl list of sequential clock pins that capture data at the rise transitions of the specified sequential clock pin.

`-domain` *`clock_domain`*

Specifies a clock domain the paths should be restricted to.

`-from {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

Also, if you specify both the `-from` option on an enable pin of a latch and the `-lenient` option, the paths starting on the D pin will also be included in the timing exception.

`-from_fall_clock {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of start points for the paths. The paths are restricted to launch at the falling edge of an ideal clock waveform. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

`-from_fall_pin {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of start points for the paths. The paths are restricted to fall transitions at the start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

`-from_rise_clock {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of start points for the paths. The paths are restricted to launch at the rising edge of an ideal clock waveform. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

`-from_rise_pin {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of start points for the paths. The paths are restricted to rise transitions at the start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

`-lenient`

Also, if you specify both the `-from` option on an enable pin of a latch and the `-lenient` option, the paths starting on the D pin will also be included in the timing exception.

If a `-from` option is provided that is not a valid startpoint or a `-to` option is provided that is not a valid endpoint, then a warning is issued but the rest of the command succeeds. These invalid points are stored in the exception, but will not affect timing analysis results. A warning is also issued when using the `report timing -lint` command in this situation.

`-mode` *`mode_name`*

Indicates the path of a particular timing exception for a specified mode.

`-through {`*`pin`*` | `*`port`*` | `*`instance`*`}`

Specifies a Tcl list of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/ mapped combinational cells, or sequential/mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-through_fall_pin {`*`pin`*` | `*`port`*` | `*`instance`*`}`

Specifies a Tcl list of points that a path must traverse. The path is restricted to fall transitions at the indicated points. Points to traverse can be ports, hierarchical pins, pins on a sequential/ mapped combinational cells, or sequential/mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on

`-through_rise_pin {`*`pin`*` | `*`port`*` | `*`instance`*`}`

Specifies a Tcl list of points that a path must traverse. The path is restricted to rise transitions at the indicated points. Points to traverse can be ports, hierarchical pins, pins on a sequential/ mapped combinational cells, or sequential/mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on

`-to {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of endpoints for the paths. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

`-to_fall_clock {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of endpoints for the paths. The paths are restricted to capture at the falling edge of an ideal clock waveform. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

`-to_fall_pin {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of endpoints for the paths. The paths are restricted to fall transitions at the endpoints. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

`-to_rise_clock {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of endpoints for the paths. The paths are restricted to capture at the rising edge of an ideal clock waveform. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

`-to_rise_pin {`*`pin`*` | `*`port`*` | `*`clock`*` | `*`external_delay`*` | `*`instance`*`}`

Specifies a Tcl list of endpoints for the paths. The paths are restricted to rise transitions at the endpoints. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

Examples

- The following example specifies the paths for the `path_disable` timing exception, by specifying a clock pin as startpoint without any other restrictions:

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

- The following example creates a group containing paths that are launched by clock `clk1` and which start with a rise pin transition at their startpoint:

```
rc:/> path_group -paths [specify_paths -from_rise_pin clk1] -group I20
```

- The following example uses the `-to_fall_clock` option with a pin object:

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

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

- Consider an input pin of a RAM that has setup arcs from both pin `CK1` and `CK2`. The following example applies a timing exception to paths using the setup arcs from `CK1`, but not to paths using the setup arcs from `CK2`:

```
rc:/> path_disable -paths [specify_paths -capture_clock_pins RAM/CK1]
```

Related Information

Specifying Timing Exceptions in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affects these commands:	multi_cycle on page 254
	path_adjust on page 259
	path_delay on page 263
	path_disable on page 266
	path_group on page 269
	report qor on page 419
	report timing on page 437
	write_encounter on page 202
	write_script on page 218
	write_sdc on page 221
Related commands:	create_mode on page 237
	define_clock on page 240
	external_delay on page 248
Affects this attribute:	paths

validate_constraints

```
validate_constraints [-rtl] [-netlist string]
     [-init_sequence_file string] [-sdc string]
     [-logfile file] [> file]
```

Validates the SDCs specified in the SDC files against the RTL or netlist. The command can be used any time after elaborating the design. If you do not specify either the `-rtl`, `-netlist`, or `-sdc` options, RTL Compiler will validate the internally generated constraints against the design at its current state.

Use this command in the False Path and Multi-cycle Path validation flows. Refer to *The Validate Flow with Dofiles* section in the *Validating and Generating Constraints* chapter in the *Interfacing Between Encounter RTL Compiler and Conformal* guide for more information on these flows.

Options and Arguments

`file`	Specifies the name of the file to write the report.
`-init_sequence_file` *`string`*	Specifies the UNIX path to the initialization sequence file for MCP validation.
`-logfile` *`string`*	Creates a separate CCD logfile. You must specify the UNIX path to the file.
`-netlist` *`string`*	Specifies the UNIX path to the netlist. This option also indicates that the validation should happen against the netlist (as opposed to the RTL).
`-rtl`	Indicates that the validation should happen against the RTL (as opposed to the netlist).
`-sdc` *`string`*	Specifies the UNIX path to the SDC files.

Examples

- The following command validates the SDCs in the `top.sdc` file against the RTL:

```
rc:/> validate_constraints -rtl -sdc /home/test/top.sdc
```

- The following command validates the `generic.nl.v` netlist against the `lane.sdc`:

```
rc:/> validate_constraints -netlist /home/cody/design/netlist.nl.v \
    -sdc /home/cody/lane.sdc
```

Related Information

Related command:	generate_constraints on page 252

7

Elaboration and Synthesis

elaborate

```
elaborate [-parameters string] [top_module_name]...
     [-libpath path]... [-libext extension]...
```

Creates a design from a Verilog module or from a VHDL entity and architecture. Undefined modules and VHDL entities are labeled “unresolved” and treated as blackboxes.

The command returns the directory path to the top-level design(s) that it creates.

Note: Before elaborating a design, load your library using the `library` attribute and load your design using the `read_hdl` command into the rc shell.

Options and Arguments

`-libext` *`extension`*	Specifies the extension of the Verilog library files. **Note:** User-specified extensions overwrite the default extensions. *Default*: `.v` and `.V`
`-libpath` *`path`*	Specifies the search path to be used to look for unresolved Verilog module instances. You can specify a relative path with respect to the working directory (.) or an absolute path. **Note:** ~ is currently not supported.
`module`	Specifies the name of the top-level module to elaborate. If *`module`* is not specified, all top-level modules are elaborated. **Note:** When reading in a structural file using the `read_hdl -netlist` command, use this option to specify the top module name.
`-parameters` *`string`*	Changes the values of Verilog parameters or VHDL generics that are used within the top level code to the specified values. Specify the new values in the same sequence as the parameter definitions appear in the code to ensure proper substitution during elaboration.

You can specify a parameter positionally, as an integer in the parameter list, or by name, as a two-element Tcl list. The first element is the name and the second element is the value. For example, you can do either:

- `elaborate -parameters {5 10}`

`elaborate -parameters {{width 5} {depth 10}}`

Examples

- In the following example, the top-level code contains the following parameters in this order:

  ```
  parameter data_width1 =  8  ;
  parameter data_width2 = 12  ;
  parameter averg_period = 4 ;
  ```

  The following command changes `data_width1` to 14, `data_width2` to 12, and `averg_period` to 8 during elaboration:

  ```
  rc:/>elaborate -parameters {14 12 8} TOP
  ```

- The following example reads in file `top.v` which has an instance of module `sub`, but file `top.v` does not contain a description of module `sub`.

  ```
  set_attr hdl_search_path { ../src ../incl }
  ```

  ```
  elaborate -libpath ../mylibs -libpath /home/verilog/libs -libext ".h"
  -libext ".v" top.v
  ```

  The latter command is equivalent to

  ```
  elaborate -libpath { ../mylibs /home/verilog/libs } -libext { ".h" ".v" } top.v
  ```

  First, `elaborate` looks for the `top.v` file in the directories specified through the `hdl_search_path` attribute. After `top.v` is parsed, `elaborate` looks for undefined modules (such as `sub`) in the directories specified through the `-libpath` option. First, the tool looks for a file that corresponds to the name of the module appended by the first specified file extension (`sub.`h). Next, it looks for a file that corresponds to the name of the module appended by the next specified file extension (sub.v), and so on.

Related Information

Affected by this command:	read_hdl on page 168
Affected by this attribute:	library

remove_assigns_without_optimization

```
remove_assigns_without_optimization
     [-buffer_or_inverter <libcell>]
       [-ignore_preserve_map_size_ok]
     [-ignore_preserve_setting]
     [-respect_boundary_optimization]
     [-preserve_dangling_nets]
     [-skip_unconstrained_paths]
     [-verbose]
     [-design {subdesign|design}]
```

Controls the aspects of the replacement of `assign` statements in the design with buffers or inverters without performing incremental optimization.

To avoid incremental optimization when removing assign statements, make sure that the `remove_assigns` attribute is set to `false`, and specify this command after synthesis.

Options and Arguments

`-buffer_or_inverter` *`libcell`*

Specifies the buffer or inverter to be used to replace the `assign` statements. The specified cell must be part of a library that was loaded.

If this option is omitted, RTL Compiler will determine which buffers or inverters to insert. If a particular library domain does not have any buffers or inverters, no buffers or inverters will be added in that domain. However, buffers or inverters will be added in other domains depending on availability.

`-design {`*`design`*` |`*`subdesign`*`}`

Indicates that the specified command options apply to the specified design or subdesign. If neither is specified, the options apply to the entire design.

`-ignore_preserve_map_size_ok`

Allows assign statements in the module to be removed if the `preserve` attribute on the module is set to `map_size_ok`.

`-ignore_preserve_setting`

Allows assign statements in the module to be removed independent of the setting of the `preserve` attribute on the module.

`-no_buffers_use_inverters`

Specifies to search for inverters and to use them in case the library or library domain does not have any buffers. If buffers exist they will be used, irrespective of whether the library has inverters or not.

`-preserve_dangling_nets`

Specifies to preserve a net that is driven by a buffer or inverter that is inserted when assign statements are replaced and if that net does not have a fanout in the next level of hierarchy and was not driven by a constant before adding the buffer or inverter.

`-respect_boundary_optimization`

Prevents RTL Compiler from looking beyond the boundary of the subdesign, when considering subports driven by constants, if the `boundary_opto` attribute on the subdesign was set to `false`.

`-skip_unconstrained_paths`

Prevents RTL Compiler from removing assigns on unconstrained paths. These paths can be paths driven by a constant, false paths with assigns, or paths without any timing constraints.

`-verbose` Displays all messages during assign removal.

Related Information

Affected by this attribute:	remove_assigns

remove_inserted_sync_enable_logic

```
remove_inserted_sync_enable_logic [> file]
```

Removes timing critical synchronous enable logic from flops.

Tools like PowerPro™ from Calypto™ can insert synchronous enable logic for sequential clock-gating in the RTL code and write out optimized RTL. However if the synchronous enable logic is inserted on a timing critical path, RTL Compiler can remove the synchronous enable logic to improve overall timing of the design.

Use the `remove_inserted_sync_enable_logic` command after mapping and before incremental or after incremental optimization.

By disconnecting the synchronous enable, some sequential instances can become unloaded. Run incremental optimization (`synthesize -incremental`) to remove these unloaded sequential instances.

Options and Arguments

`file`	Specifies the name of the file to which RTL Compiler must write the list of flops from which the synchronous enable pins were removed.

Related Information

Related command:	synthesize on page 294

retime

```
retime [-prepare] [-min_area] [-min_delay]
     [-effort {medium | low | high}]
     [design | subdesign]...
```

Improves the performance of the design by either optimizing the area or the clock period (timing) of the design. If no option is specified, the `-min_delay` option is implied. Optimization is realized through appropriately moving registers. The area optimization will not be done at the expense of timing. That is, optimizing the area will not degrade the timing.

Options and Arguments

design|*subdesign*	Specifies the name of the design or subdesign you want to retime.
`-effort {medium	low	high}`	The effort levels are only available for the `min_delay` option. ■ `high` — RTL Compiler consumes as much time as necessary to provide the optimum timing solution. ■ `low` — Provides a rough retiming estimate. This effort level provides the quickest results among the three effort levels. ■ `medium` — Provides results that are approximately within 1% of the optimum solution. *Default*: `medium`
`-min_area`	Optimizes the design for area by minimizing the number of registers without degrading the critical path in the design.
`-min_delay`	Optimizes the design for timing. For the best results, you should first issue the `retime -prepare` command separately before issuing the `retime -min_delay` command.
`-prepare`	Prepares the design for retiming and then synthesizes the design. Specifically, the `retime -prepare` command prepares the design by constraining paths according to the path delays through registers (from inputs to outputs) as opposed to register to register. There is no need to separately issue the `synthesize` command after issuing the `retime -prepare` command.

Examples

- The following example retimes the design to optimize for timing:

```
...
rc:/> retime -prepare
rc:/> retime -min_delay
...
```

Related Information

Related command:	synthesize on page 294
Affected by these attributes:	dont_retime
	retime
	retime_async_reset
	retime_effort_level
	retime_hard_region
	retime_move_mux_loop_with_reg
	retime_optimize_reset
Related attributes:	retime_original_registers
	retime_reg_naming_suffix
	retime_verification_flow
	trace_retime

set_remove_assign_options

```
set_remove_assign_options
     [ [-buffer_or_inverter libcell]
       [-no_buffers_use_inverters]
       [-ignore_preserve_setting]
       [-ignore_preserve_map_size_ok]
       [-preserve_dangling_nets]
       [-respect_boundary_optimization]
       [-skip_unconstrained_paths] [-verbose]
     | -reset]
     [-design {design | subdesign}]
```

Controls the aspects of the replacement of `assign` statements in the design with buffers or inverters, which is controlled by the `remove_assigns` root attribute. The actual replacement happens during the next incremental optimization run.

The replacement of `assign` statements is performed

- Only on the objects (design or subdesigns) specified with the `-design` option
- Only once during each incremental optimization run operation

  If your script contains multiple occurrences of the `set_remove_assign_options` command with different settings for the same object, the last settings before the next synthesize command will prevail.
- On the objects in the order of their corresponding `set_remove_assign_options` commands.

The specified options apply to all subsequent incremental optimization runs unless you reset the options with the `-reset` option.

Options and Arguments

`-buffer_or_inverter` *`libcell`*

Specifies the buffer or inverter to be used to replace the `assign` statements. The specified cell must be part of a library that was loaded.

If this option is omitted, RTL Compiler will determine which buffers or inverters to insert. If a particular library domain does not have any buffers or inverters, no buffers or inverters will be added in that domain. However, buffers or inverters will be added in other domains depending on availability.

`-design {`*`design`*` |`*`subdesign`*`}`

Indicates that the specified command options apply to the specified design or subdesign. If neither is specified, the options apply to the entire design.

`-ignore_preserve_map_size_ok`

Allows `assign` statements in the module to be removed if the `preserve` attribute on the module is set to `map_size_ok`.

`-ignore_preserve_setting`

Allows `assign` statements in the module to be removed independent of the setting of the `preserve` attribute on the module.

`-no_buffers_use_inverters`

Specifies to search for inverters and to use them in case the library or library domain does not have any buffers. If buffers exist they will be used, irrespective of whether the library has inverters or not.

`-preserve_dangling_nets`

Specifies to preserve a net that is driven by a buffer or inverter that is inserted when `assign` statements are replaced and if that net does not have a fanout in the next level of hierarchy and was not driven by a constant before adding the buffer or inverter.

`-respect_boundary_optimization`

Prevents RTL Compiler from looking beyond the boundary of the subdesign, when considering subports driven by constants, if the `boundary_opto` attribute on the subdesign was set to `false`.

`-reset` Specifies to use the command with the default settings.

`-skip_unconstrained_paths`

Prevents RTL Compiler from removing `assign` statements on unconstrained paths. These paths can be paths driven by a constant, false paths with assigns, or paths without any timing constraints.

`-verbose` Displays all messages during `assign` removal.

Examples

- The following command specifies to use buffer BUFX2 to replace `assign` statements in subdesign `med`.

  ```
  rc:/> set_remove_assign_options -buffer BUFX2 [find / -subdesign med]
  ```

- In the following example, the subdesigns `bottom` and `top` represent two different library domains. Two different kinds of buffers are specified for each domain.

  ```
  rc:/> set_remove_assign_options -buffer BUFX2 [find / -subdesign bottom]
  rc:/> set_remove_assign_options -buffer BUFX7 [find / -subdesign top]
  ```

- In the following example, several settings are specified for subdesign `sub` before the synthesize command.

  ```
  set_remove_assign_options -buffer_or_inverter [ find / -libcell buf1] \
  -design {find / -subdesign sub]
  set_remove_assign_options -buffer_or_inverter [ find / -libcell inv] \
  -design sub
  synthesize -incr
  ```

  Only the second `set_remove_assign_options` command is taken into account during the next optimization run and no `assign` removals are performed at the top level.

- In the following example, several incremental optimization runs are performed.

  ```
  set_remove_assign_options -skip_unconstrained_paths -design top
  synthesize -incr
  ```

  ```
  set_remove_assign_options -reset -design top
  synthesize -incr
  ```

  During the first opimization run, unconstrained paths remain untouched in design `top`. However, before the second optimization run starts the options have been reset for design `top` and `assign` statement can now be removed from the unconstrained paths.

Related Information

Affects this command:	synthesize
Affected by this attribute:	remove_assigns

synthesize

```
synthesize [-effort {medium | low | high}]
     [-to_generic] [-to_mapped] [-to_placed] [-spatial]
     [[-auto_identify_shift_register]
       [-shift_register_min_length integer]
       [-shift_register_max_length integer]]
     [-incremental | -no_incremental] [design]...
```

Determines the most suitable design implementation using the given design constraints (clock cycle, input delays, output delays, technology library, and so on).

The `synthesize` command takes a list of top-level designs, synthesizes the RTL blocks, optimizes the logic, and performs technology mapping.

Options and Arguments

`-auto_identify_shift_register`

Automatically identifies functional shift register segments.

`design` Specifies the name of the design to synthesize.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used. If multiple top-level designs exist, all are synthesized.

`-effort {medium | low | high}`

Specifies the effort to use during optimization.

Default: `medium`

`-incremental` Incrementally optimizes mapped gates. Allows the mapper to preserve the current implementation of the design and perform incremental optimizations if and only if the procedure guarantees an improvement in the overall cost of the design.

Note: This option only works with an already mapped design.

`-no_incremental` Disables incremental optimization.

`-shift_register_max_length` *`integer`*

Specifies the maximum sequential length of the shift register for auto-identification.

Note: This option applies only when you specify `-auto_identify_shift_register`.

Default: Longest detected shift register segment

`-shift_register_min_length` *`integer`*

Specifies the minimum sequential length of the shift register for auto-identification.

Note: This option applies only when you specify `-auto_identify_shift_register`.

Default: 2

`-to_generic`
Performs RTL optimization.

This is the default option if the RTL design has not been optimized yet.

`-to_mapped`
Maps the specified design(s) to the cells described in the supplied technology library and performs logic optimization. The aim of the optimization is to provide the smallest possible implementation of the synthesized design that still meets the supplied timing goal.

This is the default option when the design is in the generic or mapped state.

`-to_placed`
Invokes and runs the `predict_qos` command and performs post-placement optimizations.

`-spatial`
Performs a quick placement to optimize the mapped gates.

Note: You must have access to the Encounter® place and route tool to run this command option.

Examples

- The following example synthesizes and optimizes all of the top-level designs below the current position in the design hierarchy into generic logic.

  ```
  rc:/> synthesize
  ```

  The following table shows which actions are performed, depending on the state of the design.

  Table 7-1 Actions Performed with No Option Specified

Current Design State		
RTL	**Generic**	**Mapped**
■ RTL Optimization	■ Map ■ Incremental Optimizations	■ Unmap ■ Remap ■ Incremental Optimizations
(same as `-to_generic`)	(same as `-to_mapped`)	(same as `-to_mapped`)

- The following example limits synthesis to a single design `main`:

  ```
  rc:/> synthesize main
  ```

- The following example maps multiple designs at the same time:

  ```
  rc:/> synthesize -to_mapped design1 design2
  ```

- After the following example, the design in memory will be at the Boolean (generic) level of abstraction:

  ```
  rc:/> synthesize -to_generic
  ```

  The following table shows the actions that are performed for the `-to_generic` option depending on the state of the design.

  Table 7-2 Actions Performed with -to_generic Option Specified

Current Design State		
RTL	**Generic**	**Mapped**
■ RTL Optimization	■ nothing	■ Unmap design

- The following example requests mapping of the design:

  ```
  rc:/> synthesize -to_mapped
  ```

  The following table illustrates how the `-to_mapped` option affects the design:

  Table 7-3 Actions Performed with -to_mapped Option Specified

Current Design State		
RTL	**Generic**	**Mapped**
■ RTL Optimization ■ Mapping ■ Incremental Optimizations	■ Mapping ■ Incremental Optimizations	■ Unmap ■ Remap ■ Incremental Optimizations

- Table 7-4 on page 297 illustrates how the `-incremental` option affects the design:

  Table 7-4 Actions Performed with -incremental Option Specified

Options	**Current Design State**	
	RTL/Generic	**Mapped**
`-to_generic -incremental`	Disable `-incremental` and proceed	Unmap design
`-to_mapped -incremental`	Disable `-incremental` and proceed	Incremental Optimization
`-incremental`	Disable `-incremental` and proceed	Incremental Optimization

- The following example predicts and optimizes the design for silicon:

  ```
  rc:/> synthesize -to_placed
  ```

  The following table illustrates how the `-to_placed` option affects the design:

Table 7-5 Actions Performed with -to_placed Option Specified

Options	Current Design State		
	RTL	**Generic**	**Mapped**
`-to_placed`	RTL Optimization Mapping Incremental Optimization `predict_qos` & post-placement incremental optimizations	Mapping Incremental Optimization `predict_qos` & post-placement incremental optimizations	`predict_qos` & post-placement incremental optimizations
`-to_placed` `-incremental`	N/A	N/A	post-placement incremental optimizations

Related Information

Affects these commands:	report area on page 332
	report clock_gating on page 342
	report datapath on page 350
	report design_rules on page 355
	report gates on page 374
	report power on page 404
	report summary on page 435
	report timing on page 437
Related Command	remove_assigns_without_optimization on page 286
Affected by these attributes:	iopt_force_constant_removal
	iopt_sequential_resynthesis
	iopt_sequential_resynthesis_min_effort

8

Analysis and Report

all_connected

```
all_connected {net | pin | port}...
```

If the specified object is a net, the command returns the list of all the pins connected to the net. If the object is a pin or a port, the command returns the net connected to the pin or the port.

Options and Arguments

`net`	Specifies a net. The ensuing list will return a list of all the pins connected to this net.
`pin`	Specifies a pin. The ensuing list will return the net connected to the pin.
`port`	Specifies a port. The ensuing list will return the net connected to the port.

all des

```
all des {inps | insts | outs | seqs}
```

Generates a Tcl list based on the specified object. For more information on specific `all des` commands, see Related Information.

Options and Arguments

`inps`	Generates a list of all input ports of the specified clock or clock domain.
`insts`	Generates a list of all the instances in the design.
`outs`	Generates a list of all output ports of the specified clock or clock domain.
`seqs`	Generates a list of all the sequential instances in the design.

Related Information

Related commands:	all des inps on page 306
	all des insts on page 307
	all des outs on page 308
	all des seqs on page 309

all des inps

```
all des inps
     [-clock clock...] [-clock_domains clock_domain...]
     [design]
```

Generates a Tcl list of all input ports of the specified clock or clock domain.

Options and Arguments

`-clock` *`clock`*	Returns a list of input ports for the specified clock or clocks.
`-clock_domains` *`clock_domain`*	Returns a list of input ports for the specified clock domain or domains.
`design`	Returns a list of input ports for the specified design. If a design is not specified, the input ports for the current design will be returned.

Example

- The following example returns all the input ports for the clock named `clock1`:

```
rc:/> all des inps -clock clock1
```

```
{/designs/PENNY/ports_in/in1[3]} {/designs/PENNY/ports_in/in1[2]}
{/designs/PENNY/ports_in/in1[1]} {/designs/PENNY/ports_in/in1[0]}
{/designs/PENNY/ports_in/in2[3]} {/designs/PENNY/ports_in/in2[2]}
{/designs/PENNY/ports_in/in2[1]} {/designs/PENNY/ports_in/in2[0]}
```

all des insts

```
all des insts [-unresolved] [design]
```

Generates a Tcl list of all instances in the specified design. You can return a list of only unresolved instances by specifying the `-unresolved` option.

Options and Arguments

`design`	Returns a list of instances for the specified design. If a design is not specified, the instances for the current design will be returned.
`-unresolved`	List only unresolved instances and omit everything else.

Example

- The following example returns a list of all instances in the design named `STONE`:

  ```
  rc:/> all des insts STONE
  ```

  ```
  /designs/STONE/instances_hier/inst1
  /designs/STONE/instances_hier/inst1/instances_comb/g1
  /designs/STONE/instances_hier/inst1/instances_comb/g2
  /designs/STONE/instances_hier/inst1/instances_comb/g3
  /designs/STONE/instances_hier/inst1/instances_comb/g4
  /designs/STONE/instances_hier/inst2
  /designs/STONE/instances_hier/inst2/instances_comb/g1
  /designs/STONE/instances_hier/inst2/instances_comb/g2
  /designs/STONE/instances_hier/inst2/instances_comb/g3
  /designs/STONE/instances_hier/inst2/instances_comb/g4
  ```

all des outs

```
all des outs
     [-clock clock...] [-clock_domains clock_domain...]
     [design]
```

Generates a Tcl list of all output ports of the specified clock or clock domain.

Options and Arguments

`-clock` *`clock`*	Returns a list of output ports for the specified clock or clocks. This option is to find the ports with respect to a clock waveform, not a clock input port. It is valid only after you constrain the input and output ports.
`-clock_domains` *`clock_domain`*	Returns a list of output ports for the specified clock domain or domains.
`design`	Returns a list of output ports for the specified design. If a design is not specified, the output ports for the current design will be returned.

Example

- The following example returns all the output ports for the clock named `clock1`:

  ```
  rc:/> all des outs -clock clock1
  ```

  ```
  {/designs/STONE/ports_out/out1[1]} {/designs/STONE/ports_out/out1[0]}
  {/designs/STONE/ports_out/out2[3]} {/designs/STONE/ports_out/out2[2]}
  {/designs/STONE/ports_out/out2[1]} {/designs/STONE/ports_out/out2[0]}
  /designs/STONE/ports_out/out3 /designs/STONE/ports_out/out4
  ```

all des seqs

```
all des seqs
     [-clock clock...] [-clock_domains clock_domain...]
     [-exclude instance...]
     [-level_sensitive | -edge_triggered]
     [-no_hierarchy]
     [-data_pins] [-output_pins] [-clock_pins]
     [-master_slave] [-slave_clock_pins]
     [-inverted_output] [design...]
```

Generates a Tcl list of all the flip-flops and latches in the design. Use the `-level_sensitive` option to return only the latches in the design.

Options and Arguments

`-clock` *`clock`*	Returns a list of sequential instances for the specified clock or clocks.
`-clock_domains` *`clock_domain`*	Returns a list of sequential instances for the specified clock domain or domains.
`-clock_pins`	Returns the clock pins in the design.
`-data_pins`	Only returns a list of data pins.
`design`	Returns a list of sequential instances for the specified design. If a design is not specified, the sequential instances for the current design will be returned.
`-edge_triggered`	Only returns a list of flip-flops in the design.
`-exclude` *`instance`*	Specifies a list of instances to be excluded.
`-inverted_output`	Returns sequential instances that have an inverted output (Qbar)
`-level_sensitive`	Only returns a list of latches in the design.
`-master_slave`	Returns the master slave flops.
`-no_hierarchy`	Returns sequential elements only at the top-level.
`-output_pins`	Returns the output pins in the design.
`-slave_clock_pins`	Returns the slave clock pins of master slave flops.

Examples

- The following example returns all the sequential instances for the design named `REID`:

  ```
  rc:/> all des seqs REID
  ```

  ```
  {/designs/REID/instances_hier/accum1/instances_seq/r_reg[1]}
  {/designs/REID/instances_hier/accum1/instances_seq/r_reg[2]}
  {/designs/REID/instances_hier/accum1/instances_seq/r_reg[3]}
  ```

- The following example returns all the data pins for the design named `REID`:

  ```
  rc:/> all des seqs REID -data_pins
  ```

  ```
  {/designs/cpu/instances_hier/accum1/instances_seq/r_reg[1]/pins_in/d}
  {/designs/cpu/instances_hier/accum1/instances_seq/r_reg[2]/pins_in/d}
  {/designs/cpu/instances_hier/accum1/instances_seq/r_reg[3]/pins_in/d}
  ```

- The following example returns the master slave clock with the `-master_slave` option and then the slave clock pins of that master slave clock with the `-slave_clock_pins` option. If you are using the `map_to_master_slave_lssd` attribute, you must specify it before loading any libraries.

  ```
  rc:/> set_attribute map_to_master_slave_lssd true
  rc:/> set_attribute library jess.lib
  ...
  rc:/> all des seqs -master_slave

  /designs/summers/instances_seq/flop

  rc:/> all des seqs -slave_clock_pins

  /designs/summers/instances_seq/flop/pins_in/t0
  ```

Related Information

Related attributes:	lssd_master_clock
	map_to_master_slave_lssd

all lib

```
all lib {bufs | ties}
```

Generates a Tcl list of library cell objects based on the specified object. For more information on specific `all lib` commands, see Related Information.

Options and Arguments

`bufs`	Generates a list of all the buffers in the loaded library.
`ties`	Generates a list of all the tie-cells in the loaded library.

Related Information

Related commands:	all lib bufs on page 312
	all lib ties on page 313

all lib bufs

```
all lib bufs
```

Generates a Tcl list of all the buffers in the loaded library or libraries.

Example

- The following example returns all the buffers that were defined in the loaded library:

  ```
  rc:/> all lib bufs
  /libraries/slow/libcells/BUFX1 /libraries/slow/libcells/BUFX12
  /libraries/slow/libcells/BUFX16 /libraries/slow/libcells/BUFX2
  ```

all lib ties

```
all lib ties {-lo | -hi | -hilo}
```

Generates a Tcl list of all the tie-cells in the loaded library.

Options and Arguments

`-hi`	Returns only a list of tie-hi cells (1 value tie cells).
`-hilo`	Returns only a list of tie-hi-lo cells (0 and 1 value tie cells).
`-lo`	Returns only a list of tie-lo cells (0 value tie cells).

Example

- The following example returns a list of all tie-hi cells in the library named `LUX`:

  ```
  rc:/> all lib ties -hi
  /libraries/LUX/libcells/TIEHI /libraries/LUX/libcells/ANTENNA
  ```

analyze_library_corners

```
analyze_library_corners
     {-libraries list | -cpf file}
     [-buffer_libcell libcell]
     [-fanout integer] [-fanin integer]
      [> file]
```

Reads in the specified multi-corner libraries and determines the slowest corner. Multi-corner libraries have the same libcells but are each characterized for a specific set of operating conditions resulting in different delay and slew values.

For each libcell the tool takes into account a given load at the input pins (specified through the number of buffers at the input) and a given load at the output pins (specified through the number of buffers at the output). Given that configuration, the tool calculates all timing arcs for the libcells for each corner and reports the average delay per corner. In addition, it reports the list of libcells whose delay exceeds the delay of the corresponding cell in the slowest library.

Important

This command should be the only command run in the synthesis session.

Options and Arguments

`-buffer_libcell` *`cell`*	Specifies the libcell to be used as buffer.
`-cpf` *`file`*	Specifies the name of the CPF file that has the libraries for the multi corners.
`-fanin` *`integer`*	Specifies the number of buffers to consider in the fanin of the input pins of each libcell. *Default*: `2`
`-fanout` *`integer`*	Specifies the number of buffers to consider in the fanout of the output pins of each libcell *Default*: `10`
`-libraries` *`list`*	Specifies the list of multi-corner libraries.

Example

The following command reads in the libraries from the CPF file. There are 8 libraries. The report shows the average delay for each library (corner) and indicates that library XS in set4 has the largest delay.

```
analyze_library_corners -cpf test.cpf


             Library filename                Average delay
-------------------------------------------------------------
/libraries/library_domains/set8/MF           170
/libraries/library_domains/set7/MS           176
/libraries/library_domains/set6/XF           100
/libraries/library_domains/set4/XS           352
/libraries/library_domains/set2/F            162
/libraries/library_domains/set5/S            193
/libraries/library_domains/set3/VF           115
/libraries/library_domains/set1/VS           335


The slowest library : /libraries/library_domains/set4/XS
The average delay for this lib : 352

     Libcell              Library             Delay         Slowest Library          Delay
------------------------------------------------------------------------------------------
     AO22XA    /lib*/library_domains/set1/VS   354    /lib*/library_domains/set4/XS    346
------------------------------------------------------------------------------------------
    AND2CSXA   /lib*/library_domains/set1/VS   283    /lib*/library_domains/set4/XS    280
------------------------------------------------------------------------------------------
     BUFCSXA   /lib*/library_domains/set1/VS   241    /lib*/library_domains/set4/XS    239
------------------------------------------------------------------------------------------
    NAND3BXA   /lib*/library_domains/set1/VS   356    /lib*/library_domains/set4/XS    349
------------------------------------------------------------------------------------------
   NAND3BNXA   /lib*/library_domains/set1/VS   315    /lib*/library_domains/set4/XS    311
------------------------------------------------------------------------------------------
......
......
------------------------------------------------------------------------------------------
      BUFXA    /lib*/library_domains/set1/VS   241    /lib*/library_domains/set4/XS    239
------------------------------------------------------------------------------------------
     AND3XA    /lib*/library_domains/set1/VS   343    /lib*/library_domains/set4/XS    338
------------------------------------------------------------------------------------------
     OA22XA    /lib*/library_domains/set1/VS   348    /lib*/library_domains/set4/XS    340
------------------------------------------------------------------------------------------
```

Note: In the report, libraries was replaced with lib* to fit the report.

check_design

```
check_design [-lib_lef_consistency]
     [-undriven [-threshold_fanout]] [-multidriven]
     [-unloaded] [-unresolved]
     [-constant [-threshold_fanout]] [-assigns]
     [-preserved]  [-report_scan_pins]
     [-only_user_hierarchy]
     [-vname] [-all] [design] [> file]
```

Provides information on undriven and multi-driven ports and pins, unloaded sequential elements and ports, unresolved references, constants connected ports and pins, any assign statements and preserved instances in the given design. In addition, the command can report any cells that are not in both .lib and the physical libraries (LEF files).

By default, if you do not specify an option, the `check_design` command reports a summary table with this information.

Options and Arguments

`-all`	Reports all the information for the design with a summary at the end.
`-assigns`	Reports assign statements in the design.
`-constant`	Reports ports and pins in the design that are connected to a constant.
`design`	Specifies the name of the design to write the report.
`file`	Specifies the name of the file to write the report.
`-lib_lef_consistency`	Reports the cells that are present in .lib but not in the LEF file(s) and vice versa.
`-multidriven`	Reports ports and pins in the design that are multi-driven.
`-only_user_hierarchy`	Skips the tool generated internal hierarchies (such as Mux and Adders) when performing checks and only reports checks on user-created hierarchies.
`-preserved`	Reports all hierarchical and leaf instances in the design for which the `preserve` attribute is set to `true`.

`-report_scan_pins`	Includes the scan (DFT-related) pins in the checks and reports.
`-threshold_fanout`	Filters undriven or constant pins and ports with a fanout below the specified threshold.
`-undriven`	Reports ports and pins in the design that are undriven.
`-unloaded`	Reports ports and sequential elements in the design that are unloaded.
`-unresolved`	Reports unresolved references and empty modules in the design.
`-vname`	Uses the Verilog names instead of RTL Compiler design hierarchy path names in the report.

Examples

- The following example shows a sample report given when the command is executed without any options:

```
rc:/> check_design
  Checking the design.

         Check Design Report
         --------------------
 Summary
 -------
                Name                 Total
-------------------------------------------
Unresolved References                    0
Empty Modules                            0
Unloaded Port(s)                         0
Unloaded Sequential Pin(s)               0
Assigns                                  0
Undriven Port(s)                         0
Undriven Leaf Pin(s)                     0
Undriven hierarchical pin(s)             1
Multidriven Port(s)                      0
Multidriven Leaf Pin(s)                  0
Multidriven hierarchical Pin(s)          0
Multidriven unloaded net(s)              0
Constant Port(s)                         0
Constant Leaf Pin(s)                     0
Constant hierarchical Pin(s)             0
Preserved leaf instance(s)               5
Preserved hierarchical instance(s)       1
Libcells with no LEF cell              601
Physical (LEF) cells with no libcell   846

  Done Checking the design.
```

- The following example reports all the possible information on the design.

```
rc:\> check_design -all
Check Design Report
--------------------------------------
Unresolved References & Empty Modules
-------------------------------------
No unresolved references in design 'm1'
design 'm1' has the following empty module(s)
sub
Total number of empty modules in design 'm1' : 1
Unloaded Pin(s), Port(s)
-------------------------
design 'm1' has the following unloaded sequential elements:
/designs/m1/instances_seq/bx_reg
Total number of unloaded sequential elements in design 'm1' : 1
No unloaded port in 'm1'
Assigns
-------
Encountered an assign statement at subport '/designs/m1/instances_hier/sub/
subports_out/out[1]' in subdesign sub
Encountered an assign statement at subport '/designs/m1/instances_hier/sub/
subports_out/out[0]' in subdesign sub
Total number of assign statements in design 'm1' : 2
Undriven Port(s)/Pin(s)
------------------------
The following combinational pin(s) in design 'm1' are undriven
/designs/m1/instances_comb/g22/pins_in/A
/designs/m1/instances_comb/g24/pins_in/A
Total number of combinational undriven pins in design 'm1' : 2

The following sequential pin(s) in design 'm1' are undriven
/designs/m1/instances_seq/bx_reg5/pins_in/D
/designs/m1/instances_seq/bx_reg7/pins_in/D
Total number of sequential undriven pins in design 'm1' : 2

The following port(s) in design 'm1' are undriven
/designs/m1/ports_out/co
/designs/m1/ports_out/fo
Total number of undriven port(s) in design 'm1' : 2

 Multidriven Port(s)/Pin(s)
--------------------------
No multidriven combinational pin in 'm1'
No multidriven sequential pin in 'm1'
No multidriven ports in 'm1'
  Constant Pin(s)
  ----------------
No constant combinational pin(s) in design 'm1'
design 'm1' has the following constant input sequential pin(s)
/designs/m1/instances_seq/bx_reg3/pins_in/D
/designs/m1/instances_seq/bx_reg4/pins_in/D
Total number of constant sequential pins in design 'm1' : 2
No constant connected ports in design 'm1'

  Preserved instances(s)
  ----------------
No preserved combinational instance(s) in design ’m1’
No preserved sequential instance(s) in design ’m1’
No preserved hierarchical instance(s) in design ’m1’
```

```
Libcells with no corresponding LEF
----------------------------------
No libcell(s) found.

LEF cells with no corresponding libcell
---------------------------------------
No physical (LEF) cells found.

Summary
---------------------------
                Name                 Total
------------------------------------------
Unresolved References                    0
Empty Modules                            1
Unloaded Port(s)                         0
Unloaded Pin(s)                          1
Assigns                                  2
Undriven Port(s)                         2
Undriven Pin(s)                          4
Multidriven Port(s)                      0
Multidriven Pin(s)                       0
Multidriven unloaded net(s)              0
Constant Port(s)                         0
Constant Pin(s)                          2
Preserved leaf instance(s)               0
Preserved hierarchical instance(s)       0
Libcells with no LEF cell                0
Physical (LEF) cells with no libcell     0

Done Checking the design.
```

- The following example reports the preserved instances in your design.

```
rc:/> check_design -preserved
  Checking the design.

         Check Design Report
         --------------------

  Preserved instances(s)
  ----------------
design ’test1’ has the following preserved combinational instance(s)
/designs/test1/instances_comb/and1
/designs/test1/instances_comb/and2
/designs/test1/instances_comb/and3
/designs/test1/instances_hier/U1/instances_comb/and4
Total number of preserved combinational instances in design ’test1’ : 4
design ’test1’ has the following preserved sequential instance(s)
/designs/test1/instances_seq/dff1
Total number of preserved sequential instances in design ’test1’ : 1
design ’test1’ has the following preserved hierarchical instance(s)
/designs/test1/instances_hier/U1
Total number of preserved hierarchical instances in design ’test1’ : 1

  Done Checking the design.
```

- The following command reports the cells which are only in .lib or in the physical library (LEF files).

```
rc:/> check_design -lib_lef_consistency
  Checking the design.

         Check Design Report
         --------------------

Libcells with no corresponding LEF
---------------------------------
/libraries/mylib.slow/libcells/ACCSHCINX2
...
/libraries/mylib.slow/libcells/p_SMDFFHQX8

Total number of cell(s) with only library (.lib) info : 601

LEF cells with no corresponding libcell
--------------------------------------
/libraries/physical_cells/libcells/AN2D0
...
/libraries/physical_cells/libcells/XOR4D4
Total number cell(s) with only physical (LEF) Info : 846

  Done Checking the design.
```

- The following command reports all undriven pins including the DFT-related pins (highlighted in the report below).

```
rc:/> check_design -report_scan_pins -undriven
  Checking the design.

         Check Design Report
        --------------------

Undriven Port(s)/Pin(s)
 ------------------------
The following combinational pin(s) in design ’test1’ are undriven
/designs/test1/instances_comb/and3/pins_in/A
Total number of combinational undriven pins in design ’test1’ : 1

The following sequential pin(s) in design ’test1’ are undriven
/designs/test1/instances_seq/dff1/pins_in/D
/designs/test1/instances_seq/dff1/pins_in/SE
/designs/test1/instances_seq/dff1/pins_in/SI
Total number of sequential undriven pins in design ’test1’ : 3

The following hierarchical pin(s) in design ’test1’ are undriven
/designs/test1/instances_hier/U1/pins_in/A       (fanout : 0)
Total number of hierarchical undriven pins in design ’test1’ : 1

No undriven port in ’test1’

  Done Checking the design.
```

clock_ports

`clock_ports [`*`design`*`]`

Returns input ports of your design that are clock inputs.

Note: Only input ports at the top level are listed. Gated clocks and clock pins that are present in the hierarchical design internally (typical PLL outputs) will not be identified.

Note: This command is useful when you are working with an unfamiliar design.

Options and Arguments

`design`	Specifies the design for which you want to list the clock input ports.

Examples

- The following example finds all of the clock ports of a design:

  ```
  rc:/> clock_ports
  /designs/alu/ports_in/clock
  ```

- In the following example, the `clock_ports` command is embedded within a `define_clock` command to apply a clock waveform to all clock input ports of the design:

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

Related Information

Affected by this command:	define_clock on page 240

compare_sdc

```
compare_sdc [-design string] [-rtl | -netlist file]
     -golden_ sdc files [-revised_sdc files]
     [-logfile file] [-detail] [> file]
```

Compares two (or two sets of) SDC files for a design and generates a report containing differences.

To run this command you need to have access to the Encounter ® Conformal ® Constraint Designer (CCD) software.

Options and Arguments

`-design` *`string`*	Specifies the top-level design in RTL Compiler.
`-detail`	Requests a detailed comparison report.
`file`	Specifies the file to which the report must be written.
`-golden_sdc` *`files`*	Specifies the UNIX path to the golden (original) SDC files.
`-logfile` *`file`*	Specifies the name of the CCD logfile. You must specify the UNIX path to the file.
`-netlist` *`file`*	Specifies the UNIX path to the netlist.
	By default, the tool uses the RTL.
`-revised_sdc` *`files`*	Specifies the UNIX path to the revised SDC files.
	If this option is not specified, the tool internally generates an SDC file for the current state of the design and uses this file for the comparison.
`-rtl`	Specifies to use the RTL as input.

Related Information

Comparing SDC Constraint Files in *Interfacing between Encounter RTL Compiler and Encounter Conformal*

Related command:	write_do_ccd compare_sdc on page 191

fanin

```
fanin
     [-min_logic_depth integer] [-max_logic_depth integer]
     [-min_pin_depth integer] [-max_pin_depth integer]
     [-startpoints] [-structural] {pin | port | subport}...
```

Returns all the pins, ports, and subports in the fanin cone for the specified pins and ports.

Options and Arguments

`-max_logic_depth` *`integer`*

Specifies the maximum number of logic levels to go back to report the fanin cone information.

Default: Infinity

`-max_pin_depth` *`integer`*

Specifies the maximum number of pin levels to go back to report the fanin cone information.

Default: Infinity

`-min_logic_depth` *`integer`*

Specifies the minimum number of logic levels to go back to report the fanin cone information.

Default: `0`

`-min_pin_depth` *`integer`*

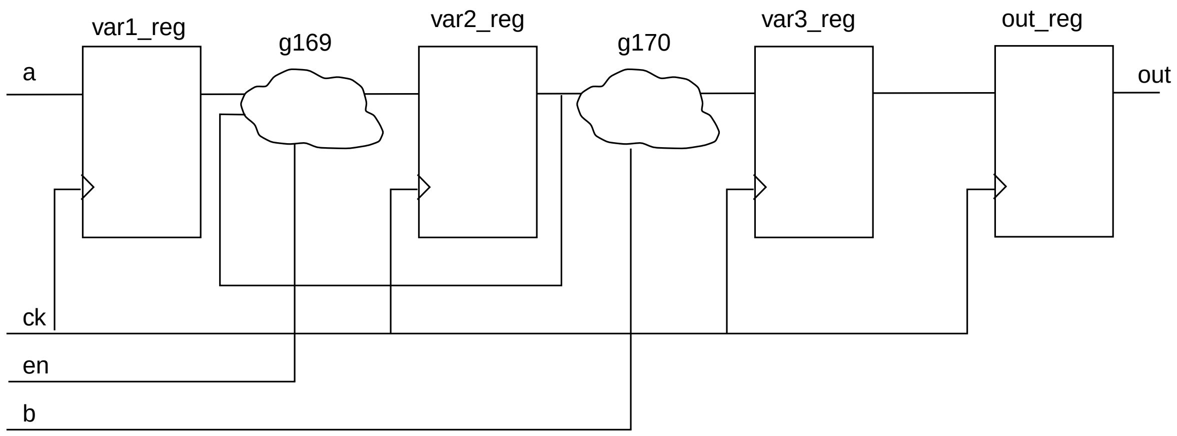
Specifies the minimum number of pin levels to go back to report the fanin cone information.

Default: `0`

`{`*`pin`* `|` *`port`*`}`	Specifies the name of a pin or port for which you want the fanin cone information.
`-startpoints`	Returns only timing start points in the fanin cone.
`-structural`	Specifies a structural trace based on connectivity instead of a timing trace, which is based on timing arcs. **Note:** If there are missing timing arcs, for example, when using the SDC `set_case_analysis` command, the traces may report different results.

Examples

- Consider the design below.



The following example returns all the pins in the fanin cone for port `out`:

```
rc:/> fanin out
/designs/test/instances_seq/out_reg/pins_out/Q
/designs/test/instances_seq/out_reg/pins_in/CK
```

- The following example specifies to only return the startpoint for port `out` shown in the design above:

```
rc:/> fanin out -startpoint out
/designs/test/instances_seq/out_reg/pins_in/CK
```

- The following example executes a path disable from all the start points that fan out to `reg1/D`:

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

- The following example queries the fanin of the output pin `S` of the combinational instance adder shown in Figure 8-1.

```
rc:/> fanin -startpoint S
```

In this case, the command returns input port `IN` and clock pin `CK`

- Use the `ls -dir` command to format the output:

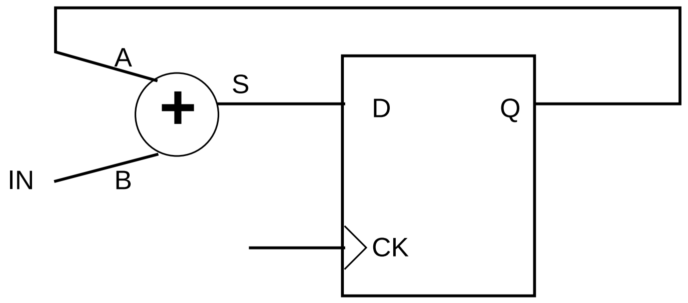
```
rc:/designs/malexander/ports_in> ls -dir [fanin in1[0]]
/designs/malexander/instances_hier/inst1/instances_comb/g43/pins_in/A
/designs/malexander/instances_hier/inst1/instances_comb/g43/pins_out/Y
/designs/malexander/ports_out/out1[1]
/designs/malexander/ports_out/out3
```

- Use the `ls -dir` command with the redirect arrow to redirect the output to the specified file:

  ```
  rc:/designs/malexander/ports_in> ls -dir [fanin in1[0]] > malexander.txt
  ```

  You can also use the append arrows (">>").

Figure 8-1 Example Design for fanin



- The following example queries the fanin of the output pin S of the combinational instance adder shown in Figure 8-1, but without using the `-startpoint` option.

  ```
  rc:/> fanin S
  ```

  In this case, the command returns in addition to the input port `IN` and clock pin `CK`, pins `A` and `Q`.

fanout

```
fanout
     [-min_logic_depth integer] [-max_logic_depth integer]
     [-min_pin_depth integer] [-max_pin_depth integer]
     [-endpoints] [-structural] {pin | port}...
```

Returns all the pins and ports in the fanout cone for the specified pins and ports.

Options and Arguments

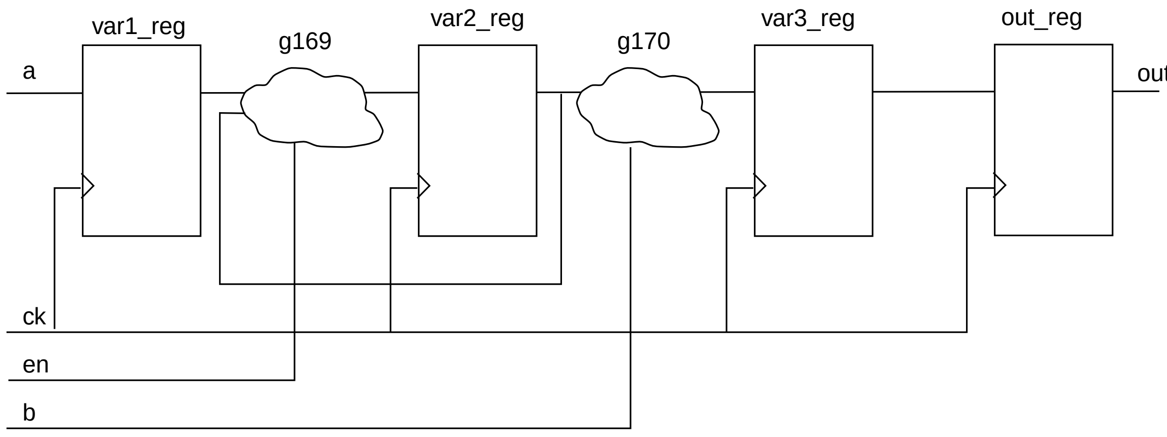
`-endpoints`	Returns only timing end points in the fanout cone.
`-max_logic_depth` *`integer`*	Specifies the maximum number of logic levels to go back to report the fanout cone information. *Default*: Infinity
`-max_pin_depth` *`integer`*	Specifies the maximum number of pin levels to go back to report the fanout cone information. *Default*: Infinity
`-min_logic_depth` *`integer`*	Specifies the minimum number of logic levels to go back to report the fanout cone information. *Default*: `0`
`-min_pin_depth` *`integer`*	Specifies the minimum number of pin levels to go back to report the fanout cone information. *Default*: `0`
`{`*`pin`* `|` *`port`*`}`	Specifies the name of a pin or port for which you want the fanout cone information.
`-structural`	Specifies a structural trace based on connectivity instead of a timing trace, which is based on timing arcs. **Note:** If there are missing timing arcs, for example, when using the SDC `set_case_analysis` command, the traces may report different results.

Examples

- The following example returns all the pins in the fanout cone of port `en` in the design shown in Figure 8-2 on page 327:

  ```
  rc:/> fanout en
  /designs/test/instances_comb/g170/pins_in/SL
  /designs/test/instances_comb/g170/pins_out/Z
  /designs/test/instances_seq/var2_reg/pins_in/D
  /designs/test/instances_comb/g169/pins_in/A1
  /designs/test/instances_comb/g169/pins_out/Z
  /designs/test/instances_seq/out_reg/pins_in/D
  ```

Figure 8-2 Example Design for fanout



- The following example executes a path disable on all the endpoints to which `reg1/CK` fans out:

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

- The following example returns all pins in the fanout cone up to two logic levels forward from the specified pin:

  ```
  rc:/> fanout -max_logic_depth 2
  /designs/top/instances_hier/m1/instances_comb/g2/pins_in/in_0
  ```

- Use the `ls -dir` command to format the output:

  ```
  rc:/designs/malexander/ports_in> ls -dir [fanout in1[0]]
  /designs/malexander/instances_hier/inst1/instances_comb/g43/pins_in/A
  /designs/malexander/instances_hier/inst1/instances_comb/g43/pins_out/Y
  /designs/malexander/ports_out/out1[1]
  /designs/malexander/ports_out/out3
  ```

- Use `ls -dir` with the redirect arrow to redirect the output to the specified file:

  ```
  rc:/designs/malexander/ports_in> ls -dir [fanout in1[0]] > malexander.txt
  ```

  You can also append arrows (">>").

report

```
report {area | boundary_opto | cdn_loop_breaker |
     | cell_delay_calculation | clock_gating | clocks
     | congestion | cwd | datapath | design_rules
     | dft_chains | dft_registers | dft_setup
     | dft_violations | disabled_transparent_latches
     | gates | hierarchy | instance | isolation
     | level_shifter | memory | memory_cells | messages
     | net_cap_calculation | net_delay_calculation
     | net_res_calculation | nets | operand_isolation
     | ple | port | power | power_doamin | qor | scan_power
     | sequential | slew_calculation | state_retention
     | summary | timing | yield}
```

Generates the specified report on synthesis results. For more information, see the Related Information.

You can automatically write out a gzip compressed report file. For example:

```
report port sample.gz
```

Note: The `report memory` command does not support the gzip compressed output.

Options and Arguments

`area`	Reports the area of the synthesized and mapped design.
`boundary_opto`	Reports a summary of the boundary optimization.
`cdn_loop_breaker`	Reports the loop breaker buffers added by the tool and breaks the combinational loops for timing analysis.
`cell_delay_calculation`	Reports how the cell delay of a libcell instance is calculated.
`clock_gating`	Reports clock-gating information for the design.
`clocks`	Generates a report on the clocks of the current design.
`congestion`	Reports the congestion summary.
`cwd`	Generates a ChipWare Developer report.
`datapath`	Reports datapath operators that were inferred from the design.
`design_rules`	Reports the design rule violations.
`dft_chains`	Reports the scan chain data for the design.

`dft_registers`	Reports the scan status of all flip-flops in the design.
`dft_setup`	Reports the DFT setup information.
`dft_violations`	Reports the DFT violations.
`disabled_transparent_latches`	Reports disabled transparent latches.
`gates`	Generates a gates report.
`hierarchy`	Reports the design hierarchy information.
`instance`	Generates a report on the instances of the current design.
`isolation`	Reports isolation cell information for the design.
`level_shifter`	Reports level-shifter information for the design.
`memory`	Reports the memory usage in the compilation environment.
`memory_cells`	Reports the memory cells in the library.
`messages`	Generates a summary of error messages that have been issued.
`net_cap_calculation`	Reports how the capacitance of the net is calculated.
`net_delay_calculation`	Reports how the net delay is calculated.
`net_res_calculation`	Reports how the resistance of the net is calculated.
`nets`	Generates a report on the nets of the current design.
`operand_isolation`	Reports operand-isolation information for the design.
`ple`	Reports physical layout estimation data
`port`	Generates reports on the ports of the current design.
`power`	Generates a power leakage report.
`power_domain`	Generates a report with power domain information.
`qor`	Generates a QoR report.
`scan_power`	Reports estimated power of design during scan test
`slew_calculation`	Reports how the slew on the driver pin of a libcell instance is calculated.
`sequential`	Generates a report on the sequential elements of the design.

`state_retention`	Reports state_retention information for the design.
`summary`	Generates an area, timing, and design rule violations report.
`timing`	Generates a timing report.
`yield`	Generates a yield report.

Related Information

Related commands:	report area on page 332
	report boundary_opto on page 334
	report cdn_loop_breaker on page 336
	report cell_delay_calculation on page 338
	report clock_gating on page 342
	report clocks on page 347
	report congestion on page 349
	report datapath on page 350
	report design_rules on page 355
	report dft_chains on page 356
	report dft_registers on page 361
	report dft_setup on page 365
	report dft_violations on page 370
	report disabled_transparent_latches on page 373
	report gates on page 374
	report hierarchy on page 377
	report instance on page 379
	report isolation on page 381
	report level_shifter on page 384
	report memory on page 388

report area

```
report area
     [-depth integer] [-min_count integer]
     [design]... [> file]
```

Reports the following information:

- The total count of cells mapped against the hierarchical blocks in the current design
- The combined cell area in each of the blocks and the top level design (hierarchical breakup)

  The `Cell Area` numbers are based on the cell implementations taken from the technology library after mapping. However, in PLE mode, the numbers are based on the information in the LEF libraries. It might be 0 if the information is missing in the LEF libraries.
- The `Net Area` refers to the estimated post-route net area and is only meaningful if you read in the LEF libraries. Net area is based on the minimum wire widths defined in the LEF and capacitance table files and the area of the design blocks.
- The wireload model adopted for each of the blocks

Note: The units used in the report depend on the units used in the library.

Options and Arguments

`-depth` *`integer`*	Specifies the number of levels of recursion.
`design`	Specifies the design for which you want to generate a report. You can also `cd` into the particular design directory and generate the report.
`file`	Specifies the name of the file to which to write the report.
`-min_count` *`integer`*	Specifies the minimum instance count per block.

Examples

- The following example generates the area report for the current design:

```
> report area
====================================================================
  Generated by:          RTL Compiler (RC) version
  Generated on:          Current date Current time
  Module:                alu
  Technology library:    tutorial 1.0
  Operating conditions:  typical_case (balanced_tree)
  Wireload mode:         enclosed
====================================================================

*************  Area  *************

Instance         Cells   Cell Area   Net Area     Wireload
---------------------------------------------------------------
alu                210         378          0    AL_MEDIUM (S)
  ops1_add_25       61         101          0    AL_MEDIUM (S)

 (S) = wireload was automatically selected
```

Related Information

Affected by this command:	synthesize on page 294
Affected by this attribute	shrink_factor

report boundary_opto

```
report boundary_opto
```

Reports a summary of the boundary optimization done on the design.

This is a summary of boundary changes on the hierarchical pins.

Example

Consider the following input RTL:

```
module top(in1,in2,out,out1,out2);
    input in1,in2;
    output out,out1,out2;
    child inst(in1,in2,out,out1,out2);
endmodule

module child(a,b,out,out1,out2);
    input a,b;
    output out,out1,out2;
    and u1(n_1,b,a);
    assign out = n_1;
    assign out1 = ~a;
    assign out2 = ~n_1;
endmodule
```

After the `synthesize -to_map` command, report the boundary optimization.

```
rc:/> report boundary_opto
============================================================
  Generated by:           Encounter(R) RTL Compiler 9.1.100
  Generated on:           Current date Current time
  Module:                 top
  Technology library:     tutorial 1.1
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

Instance  Pin                     Boundary Change
---------------------------------------------------------------------
inst     out1  routed opposite signal through ’inst/a’ around ’inst’
         out2  pushed opposite signal through ’inst/out’
```

Related Information

Related attribute:	boundary_change

report buskeepers

```
report buskeepers
     [> file]
```

Important

This command is only available in RCQA mode.

Reports the bus keeper cells in the library.

Bus keeper cells store the previous state (value) to avoid bus contentions, and are used mostly in PADS, tristates, other standard cells for bus inputs. These cells are defined in the library with the `driver_type` Liberty attribute with the `bus_hold` value.

Options and Arguments

`file`	Specifies the name of the report file.

Examples

- The following command reports the buskeeper cells to a file named `file.rpt`:

  ```
  rcqa:/> report buskeepers > file.rpt
  ```

report cdn_loop_breaker

```
report cdn_loop_breaker [-sdcfile string]
     [-version string] [design] [> file]
```

Reports the loop breaker buffers that were added by the timing engine to break the combinational loops during timing analysis.

The report lists for every loop breaker the instance name, and the driver and the load of the loop breaker.

Options and Arguments

design	Specifies the design for which you want the report.
file	Redirects the report to the specified file.
-sdcfile *string*	Creates an SDC file with the appropriate `set_disable_timing` and `set_false_path` settings.
-version *string*	Specifies a particular SDC version for the SDC file created with the `-sdcfile` option. The available versions are: `1.1, 1.3, 1.4, 1.5` or `1.7.` *Default*: `1.7`.

Example

The following report shows the loop breakers inserted in design `loop`.

```
rc:/> report cdn_loop_breaker
============================================================
  Generated by:           version
  Generated on:           date
  Module:                 loop
  Technology library:     tutorial 1.1
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

   CDN Loop breaker      Driver   Load
---------------------------------------
inst1/cdn_loop_breaker inst1/i1/Y i0/B
```

Related Information

Related command: remove_cdn_loop_breaker on page 817

report cell_delay_calculation

```
report cell_delay_calculation
     -from_pin pin  -to_pin pin
     [-from_rise] [-from_fall]
     [-to_rise] [-to_fall] [> file]
```

Reports how the cell delay is calculated from the look up table in the loaded technology library. Specify the cell by choosing its the driving and loading pins.The formula for calculating the delay is provided at the bottom of the report.

Options and Arguments

file	Redirects the report to the specified file.
-from_pin *pin*	Specifies the driving pin.
-from_fall	Uses the fall delay calculation from the driving pin.
-from_rise	Uses the rise delay calculation from the driving pin.
-to_fall	Uses the fall delay calculation of the loading pin.
-to_pin *pin*	Specifies the loading pin.
-to_rise	Uses the rise delay calculation of the loading pin.

report checks

```
report checks
     [-hdl_lint]
     [-clock_domain_crossing]
     [-constraints] [-library]
     [-power] [-dft] [-physical]
     [-error] [-warning] [-info]
     [-summary | -detail]
     [> file]
```

Important

This command is only available in RCQA mode.

Reports the logical design signoff checks that were performed when running the `signoff_checks` command. By default, this command reports all checks.

Options and Arguments

`-clock_domain_crossing`	Reports clock domain crossing checks
`-constraints`	Reports SDC lint checks
`-detail`	Details every message for every message ID, one message per line.
`-dft`	Reports DFT checks.
`-error`	Reports Error severity messages.
`file`	Specifies the name of the file to direct the report output.
`-hdl_lint`	Reports the HDL lint checks.
`-info`	Reports Info severity messages.
`-library`	Reports library checks.
`-physical`	Reports physical netlist checks.
`-power`	Reports power checks.
`-summary`	Prints the total number of messages for all severities. By default, the command reports on the total number per ID, followed by this summary.
`-warning`	Reports Warning severity messages.

Examples

- The following command reports a detailed table for HDL lint checks with a summary and includes only warning messages:

  ```
  rcqa:/> report checks -hdl_lint -warning
  ```

- The following command reports a detailed table for DFT checks including error, warning and info messages, and a summary:

  ```
  rcqa:/> report checks -dft
  ```

- The following shows a portion of the report when running the HDL lint checks for warning messages, and the summary the shows the total message count for all severities:

  ```
  rcqa:/> report checks -hdl_lint -warning
       HDL Lint Report
       ---------------

   File Formatting

  Message ID Count Severity                        Help
  ---------------------------------------------------------------------------
  CTLCHR     668   Warning  HDL source line contains one or more control
                            characters
  MAXLEN     1871  Warning  HDL source line is too long
  UCCONN     112   Warning  Lowercase characters used for identifier
  NOBLKN     42    Warning  Each block should be labeled with a meaningful
                            name
  SIGLEN     12    Warning  Signal name is not of appropriate length
  VERREP     29    Warning  Repeated usage of identifier or label name
  DIFFMN     6     Warning  Name differs from file name
  LCVARN     45    Warning  Name uses uppercase characters
  MULTMF     2     Warning  More than one module definition in file

  Total number warning messages : 2787
  ...
       Summary
       -------

  Total number of error messages : 360
  Total number of warning messages : 5897
  Total number of info messages : 161
  ```

- The following shows a portion of the report when running the SDC lint checks for error severity messages with the `-detail` option:

```
rcqa:/> report checks -constraints -error -detail
     Constraints Report
     ------------------
     SDC Lint

Message ID   Severity               Info                    File/Row/Column
----------------------------------------------------------------------------
CCD_SDC_USG1   Error     No object of the expected type(s)   ./firb.sdc 69
                         matches the given name(s) In line 69,
                         file ./firb.sdc (set_driving_cell):
                         The library 'slow' has not been read
                         using the read library command
CCD_SDC_USG1   Error     No object of the expected type(s)   ./firb.sdc 88
                         matches the given name(s) In line 88,
                         file ./firb.sdc (get_lib_cells):
                         slow/RFRDX2

......
......
```

- The following shows a portion of the report when running the library checks for info severity messages with the `-detail` option:

```
rcqa:/> report checks -library -info -detail
     Library Report
     --------------
     Library

Message ID   Severity               Info                    File/Row/Column
----------------------------------------------------------------------------
LBR-40         Info     An unsupported construct was detected s.lib.gz 6 19
                        in this library.
                        .lib.gz:6:18: Construct
                         'library_features' is not supported.
LBR-40         Info     An unsupported construct was detected s.lib.gz 34 39
                        in this library.
                        .lib.gz:34:31: Construct
                        'slew_lower_threshold_pct_fall' is not
                        supported.
```

report clock_gating

```
report clock_gating
     [ -preview [-gated_ff]
       [-clock clock_list | -clock_pin {pin|port|subport}...]
     | [-gated_ff] [-ungated_ff] [-no_hierarchical]
       [-summary] [-detail]
     | {-clock clock_list |-clock_pin {pin|port|subport}...}
       [-detail]
     | -cg_instance cg_instance...
     | -multi_stage ]  [> file]
```

Reports clock-gating information for the design.

Note: After importing third-party clock-gating logic, this clock-gating logic will be reported as "RC Clock Gating Instances."

Tip

If you use a user-defined clock-gating module, you need to change your current location in the hierarchy to the *`design`* directory, before you enter this command.

Options and Arguments

`-cg_instance`

Reports detailed clock-gating information for the specified clock-gating instances. Information includes the library cell used for the clock-gating cell, the clock-gating style, the signals connected to the inputs and outputs of the gating logic, and the flip-flops gated by this gating cell.

A clock-gating instance is the hierarchical instance with the clock-gating logic inside.

`-clock` *`clock_list`* Limits the report to the specified clocks. The specified clocks must that have been defined through either the `define_clock` command or through the SDC constraints.

`-clock_pin {`*`pin`* `|` *`port`*`|` *`subport`*`}`

Limits the report to the specified clock pins. Use this option if you did not define the clocks. You can specify clock pins, clock ports or clock subports.

Note: If both `-clock` and `-clock_pin` options are specified, the `-clock` option takes precedence.

`-detail`	Reports detailed clock-gating information. Lists all the clock-gating instances inserted, including the library cell used for the clock-gating cell, the clock-gating style, the signals connected to the inputs and outputs of the gating logic, and the flip-flops gated by this gating cell. If you specify only this option, the return value of this command is the total number of clock-gating logic inserted in the design.
`file`	Specifies the name of the file to which to write the report.
`-gated_ff`	Reports all the flip-flops that are clock gated and the clock-gating instances that gate the flip-flops. If you specify only this option, the return value of this command is the total number of flip-flops that are gated in the design. If you specify this option with the `-preview` option, the report adds for each combination of clock and enable inputs (listed in the report) the names of the flip-flops that can potentially be gated.
`-multi_stage`	Shows the clock-gating instance hierarchy. This option cannot be combined with any other options.
`-no_hierarchical`	Limits the clock-gating information to the current module. By default, the report command traverses the hierarchy starting from the current module and reports all the clock gating found in the current module and its children modules.
`-preview`	Shows the potential reduction in the clock toggling when clock-gating logic would be inserted—without making any modifications to the netlist. Use this option on an unmapped or partially mapped design. If you did not define the clocks, you must specify the `-clock_pin` option to identify a clock. **Note:** If both `-clock` and `-clock_pin` options are specified, the `-clock` option takes precedence.

`-summary`	Prints a summary report of the clock gating inserted in the design that includes the number of clock-gating instances, the number of gated flip-flops, and the number of ungated flip-flops. **Note:** The first two lines refer to the *leaf* clock-gating instances (RC and non-RC) that were added. If multi-stage clock gating is present, two more lines are added to the top of the summary reporting the multi-stage clock gating instances (RC and non-RC). If used with other options, a summary report is printed at the end. If you specify only this option, the return value of this command is the total number of clock-gating logic inserted in the design. If no other options specified, you will get a summary report by default.
`-ungated_ff`	Reports all the flip-flops that are not clock gated, and lists whether the flop was excluded for clock-gating or not. If you specify only this option, the return value of this command is the total number of flip-flops that are not gated in the design.

Examples

The following reports show output generated after clock gating has been inserted.

- The following command reports all the flip-flops that are clock gated. In this case, the return value of the command is 8.

```
rc:/> report clock_gating -gated_ff
============================================================
...
============================================================

Gated Flip-flops
----------------
 Module  Clock Gating Instance  Fanout  Gated Flip-flops
----------------------------------------------------------
 alu     RC_CG_HIER_INST           8    aluout_reg[0]
                                        aluout_reg[1]
 ...
                                        aluout_reg[7]
----------------------------------------------------------
 Total   1                         8
==========================================================

8
```

- The following command reports the number of flip-flops that are clock gated by the specified clock.

```
rc:/> report clock_gating -clock [find / -clock clock]
Info: Since -clock option is specified, options other than -detail are ignored.
Multi-stage clock gating for '/designs/alu/ports_in/clock'
====================================
Max stage:    1
Total FFs with 0 stage of CG:  1
Total FFs with 1 stage of CG:  8
====================================
Total FF:  9
```

- The following command reports all the flip-flops that are not clock gated. In this case, the return value of the command is 1.

```
rc:/> report clock_gating -ungated_ff
============================================================
...
============================================================

Ungated Flip-flops
------------------
 Flip-flop  Excluded  Module  Instance
----------------------------------------
 zero_reg     false   alu     alu
----------------------------------------
Total ungated flip-flops:    1
----------------------------------------


1
```

- The following command generates detailed clock-gating information for the specified clock-gating instance:

```
rc:/> report clock_gating -cg_instance RC_CG_HIER_INST
Info: Since -cg_instance option is specified, all other options are ignored.
...
============================================================

Detail
------
Clock Gating Instance : RC_CG_HIER_INST
---------------------
  Origin:               Inserted by RC
  Libcell:              TLATNTSCAX2 (slow)
  Style:                latch_posedge_precontrol
  Module:               alu (alu)
  Type:                 Leaf level CG Instance
  Inputs:
    ck_in       =       clock (/designs/alu/ports_in/clock)
                        TCF = (0.75000, 0.571429/ns)
    enable      =       ena (/designs/alu/ports_in/ena)
                        TCF = (0.50000, 0.020000/ns)
    test        =       LOGIC0
  Outputs:
    ck_out      =       rc_gclk_420
                        TCF = (0.37380, 0.310000/ns)
  Gated FFs:
```

```
 Module  Clock Gating Instance  Fanout  Gated Flip-flops
----------------------------------------------------------
 alu     RC_CG_HIER_INST           8    aluout_reg[0]
                                        aluout_reg[1]
                                        aluout_reg[2]
 ...
                                        aluout_reg[6]
                                        aluout_reg[7]
----------------------------------------------------------
 Total   1                         8
==========================================================

1
```

Related Information

Previewing the Effect of Clock Gating in *Low Power in Encounter RTL Compiler*

Reporting Clock-Gating Information in *Low Power in Encounter RTL Compiler*

Clock Gating Cell Specification in the *Library Guide for Encounter RTL Compiler.*

Affected by this command:	synthesize on page 294
Affected by these attributes:	lp_clock_gating_add_obs_port
	lp_clock_gating_add_reset
	lp_clock_gating_cell
	lp_clock_gating_control_point
	lp_clock_gating_exclude
	lp_clock_gating_style

report clocks

```
report clocks
     [-ideal] [-generated]
     [clock]... [-mode mode_name] [> file]
```

Generates a report on the clocks of the current design. Reports the clock period, rise, fall, domain, setup uncertainty, latency, clock ports or sources in the current design.

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

Options and Arguments

clock	Specifies the name of the clock for which you want to generate the report.
file	Specifies the name of the file to which to write the report.
`-generated`	Adds generated clock information to the description, uncertainty, and the relationship table.
`-ideal`	Reports a clock description with the ideal clock - clock relationship.
`-mode` *mode_name*	Generates a report by mode on the clocks of the current design.

Example

The following example generates a clock report with generated clock information added to the table:

```
rc:/> report clocks -generated
============================================================
  Generated by:           RTL Compiler (RC) Version
  Generated on:           Date
  Module:                 test
  Technology library:     tutorial 1.0
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
============================================================
 Clock Description
 -----------------
 Clock                                              No of
 Name    Period  Rise    Fall     Domain  Pin/Port  Registers
--------------------------------------------------------------
 CLK1    4000.0   0.0   2000.0   domain_1   Clk            5
 CLK2    2000.0   0.0   1000.0   domain_1   C              0
 CLK3    3000.0   0.0   1500.0   domain_2   Clk1           5
 CLK4    6000.0   0.0   3000.0   domain_2   C1             0
 Clock Network Latency / Setup Uncertainty
 -----------------------------------------
        Network   Network   Source   Source      Setup         Setup
Clock   Latency   Latency  Latency  Latency  Uncertainity  Uncertainity
 Name    Rise      Fall      Rise     Fall       Rise          Fall
-----------------------------------------------------------------------
CLK1      140.0     140.0    150.0    150.0         100.0         100.0
CLK2      120.0     120.0    120.0    120.0         110.0         110.0
CLK3      100.0     100.0    100.0    100.0         100.0         100.0
CLK4        0.0       0.0      0.0      0.0           0.0           0.0
 Clock Relationship (with uncertainity & latency)
 -----------------------------------------------
  From    To     R->R     R->F     F->R     F->F
--------------------------------------------------
  CLK1   CLK1   3900.0   1900.0   1900.0   3900.0
  CLK1   CLK2   1840.0    840.0   1840.0    840.0
  CLK2   CLK1   1950.0   1950.0    950.0    950.0
  CLK2   CLK2   1890.0    890.0    890.0   1890.0
  CLK3   CLK3   2900.0   1400.0   1400.0   2900.0
  CLK3   CLK4   2800.0   2800.0   1300.0   1300.0
  CLK4   CLK3   3100.0   1600.0   3100.0   1600.0
  CLK4   CLK4   6000.0   3000.0   3000.0   6000.0
```

Related Information

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affected by this command:	create_mode on page 237

report congestion

```
report congestion [ >file]
```

Reports the total number (and percentage) of gcells with overflow, the total overflow of the design as well as the maximum overflow and the associated gcell.

Options and Arguments

`file`	Specifies the name of the file to which to write the report.

Example

The following command shows the congestion summary for design `test`.

```
rc:/> report congestion
============================================================
  Generated by:           RTL Compiler (RC) Version
  Generated on:           Date
  Module:                 test
  Technology libraries:   lib1
                          lib2
  Operating conditions:   max
  Interconnect mode:      placement
  Area mode:              physical library
============================================================

GCELLS with H overflow: 99 (6.71%)
GCELLS with V overflow: 329 (22.29%)
Total number of GCELLS: 1476

Item                   Oflow/Avail   Gcell (Bounding-box location)
----------------------------------------------------------------------
Max. Overflow          -111/201      20,18 (480.0,432.0),(504.0,456.0)
Max. Overflow (H)      -52/118       31,29 (744.0,696.0),(768.0,720.0)
Max. Overflow (V)      -75/87        19,17 (456.0,408.0),(480.0,432.0)
Total Overflow         -7172
Total Overflow (H)     -1329
Total Overflow (V)     -5843
```

report datapath

```
report datapath [-full_path]
     [-no_header] [-no_area_statistics]
     [-mux] [-all] [-max_width string]
     [-print_instantiated] [-print_inferred]
     [-sort keys] [design] [> file]
```

Reports datapath operators that were inferred from the design. This command is available after elaboration. You must set the set the `hdl_track_filename_row_col` attribute to `true` to enable filename, column, and line number tracking in the datapath report; otherwise these headings will be hidden.

Options and Arguments

`-all`	Reports all datapath operators present in the design including muxes. **Note:** The mux operators are different from the MUX library cells that are picked by the mapper or are available in the technology library.
`design`	Specifies a particular design on which to report datapath operators. By default, RTL Compiler reports on the current design.
`file`	Specifies the name of the file to which to write the report.
`-full_path`	Reports the full UNIX path name of the filename, including the filename. By default, RTL Compiler only reports the design name. See Examples for more information.
`-max_width` *`string`*	Specifies the maximum width of individual column names. By default, the maximum width for a column is 20. If a name is more than 20, it will wrap to the next line. The valid column names are *Module*, *Instance*, *Operator*, *Signedness*, *Architecture*, *Inputs, Outputs*, *Cell Area*, *Line*, *Col*, and *Filename*.
`-mux`	Reports muxes present in the design. Muxes are not reported by default. Using the `-mux` option only displays the muxes in the design and suppresses the other datapath operators. To view both, use the `-all` option.

`-no_area_statistics`	Suppresses the table that only shows the total area and percentage information. The area and the percentage of the total area consumed by the datapath operators in the design are only available after issuing the `synthesize -to_mapped` command.
`-no_header`	Suppresses the header information.
`-print_inferred`	Reports only the inferred datapath components in the design.
`-print_instantiated`	Reports only the instantiated datapath components in the design.
`-sort` *`keys`*	Indicates how to sort the report. You can sort on the following keys: ■ `architectures` sorts the architectures in alphabetical order ■ `area` sorts by descending area ■ `inputs` sorts based on the number*width (number of bits) of the input signals—components with higher number of bits are printed first ■ `instances` sorts the instances in alphabetical order ■ `outputs` sorts based on the number*width (number of bits) of the output signals—components with higher number of bits are printed first ■ `operators` sorts by operator ■ `slack` sorts by ascending slack ■ `subdesigns` sorts the subdesigns in alphabetical order By default, the report does not contain the slack numbers. **Note:** You can sort on several keys.

Examples

- The following example generates a report of the datapath components for the `rpdp1_basic` design.

```
rc:/> report datapath rpdp1_basic
============================================================
...
  Module:                 rpdp1_basic
  Technology library:     tutorial 1.0
...
============================================================
Instantiated datapath components

              Operator Signedness  Inputs Outputs CellArea Line Col  Filename
===============================================================================
rpdp1_basic
 d1
  u2
  module:CW__CW_multadd__builtin_wA8_wB8_wC8_wZ16
   CW/CW_multadd/builtin
                 n/a   n/a        8x8x8x1 16       1239.75   38  52 impl_inf.v
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   mul_1_19
    very_fast/non_booth
                  *    signed     9x9     16        912.75
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   add_1_24
    very_fast     +    signed     16x9    16        322.50
===============================================================================

Inferred components

              Operator Signedness  Inputs Outputs CellArea Line Col  Filename
===============================================================================
rpdp1_basic
 mul_18_28
 module:mult_unsigned
  very_fast/non_booth
                  *    unsigned   16x8    16       1044.00   18  28 impl_inf.v
===============================================================================

      Type       CellArea Percentage
-------------------------------------
datapath modules  2283.75      20.55
external muxes       0.00       0.00
others            8829.00      79.45
-------------------------------------
total            11112.75     100.00
```

- By default, when using the `report datapath` command on a mapped netlist containing datapath operators, you will get the area statistics of the design, as shown in the following example:

```
------------------------------------
datapath modules 4938.00     100.00
mux modules         0.00       0.00
others              0.00       0.00
------------------------------------
total            4938.00     100.00
```

This information is useful in determining the percentage of the design that contains datapath operators. If you do not want to report this information, then use the `-no_area_statistics` option.

By default, the area report is suppressed for a netlist that contains only generic cells (no library cells).

- The following command provides a 30-character width to the *filename* column and provides a 0-character width to the *area* column:

  ```
  report datapath -max_width {{filename 30} {area 0}}
  ```

- The following command generates a report sorted by area.

  ```
  rc:/> report datapath -sort area
  ============================================================
  ...
  ============================================================


  Inferred components

              Operator Signedness Inputs Outputs CellArea Line Col Filename
  ==========================================================================
  lt_leq
   add_14_26
   module:add_signed
    very_fast     +    signed     4x4    4          31.50   14  26 lt_leq.v
  ==========================================================================
  lt_leq
   lt_13_25
   module:lt_signed
    very_fast     <    signed     4x4    1          26.25   13  25 lt_leq.v
  ==========================================================================
  lt_leq
   lt_16_25
   module:lt_unsigned
    very_fast     <    unsigned   4x4    1          26.25   16  25 lt_leq.v
  ==========================================================================
  lt_leq
   le_17_26
   module:leq_unsigned
    very_fast    <=    unsigned   4x4    1          25.50   17  26 lt_leq.v
  ==========================================================================

        Type       CellArea Percentage
  -------------------------------------
  datapath modules   109.50      75.26
  external muxes       0.00       0.00
  others              36.00      24.74
  -------------------------------------
  total              145.50     100.00
  ```

- The following command sorts the report first by slack, then by area:

```
rc:/> report datapath -sort {slack area}
============================================================
...
============================================================


Inferred components

          Operator Signedness Inputs Outputs CellArea Line Col Filename  Slack
==============================================================================
====
lt_leq
 lt_13_25
 module:lt_signed
  very_fast     <    signed     4x4    1         26.25   13  25 lt_leq.v -145.2
============================================================================
lt_leq
 lt_16_25
 module:lt_unsigned
  very_fast     <    unsigned   4x4    1         26.25   16  25 lt_leq.v -145.2
============================================================================
lt_leq
 le_17_26
 module:leq_unsigned
  very_fast    <=    unsigned   4x4    1         25.50   17  26 lt_leq.v -53.3
============================================================================
lt_leq
 add_14_26
 module:add_signed
  very_fast     +    signed     4x4    4         31.50   14  26 lt_leq.v 49.0
============================================================================

      Type       CellArea Percentage
-------------------------------------
datapath modules   109.50      75.26
external muxes       0.00       0.00
others              36.00      24.74
-------------------------------------
total              145.50     100.00
```

Related Information

Generating Datapath Reports in *Datapath Synthesis in Encounter RTL Compiler* for detailed information on interpreting the report, reporting RTL sharing, and interpreting the `report datapath` command at different stages in the design flow.

Affected by this attribute: hdl_track_filename_row_col

report design_rules

`report design_rules [`*`design`*`]... [> `*`file`*`]`

Reports any design rule violations that are present in the specified design objects.

Options and Arguments

design	Specifies the design for which you want to generate a report. By default, a report is created for all designs currently loaded in memory.
file	Specifies the name of the file to which to write the report.

Examples

- The following example generates a report of the design rule violations for the specified design:

```
> report design_rules alu
============================================================
  Generated by:          RTL Compiler (RC) version
  Generated on:          Current date Current time
  Module:                alu
  Technology library:    tutorial 1.0
  Operating conditions:  typical_case (balanced_tree)
  Wireload mode:         enclosed
============================================================

Max_transition design rule: no violations.


Max_capacitance design rule: no constraints.


Max_fanout design rule: no constraints.
```

Related Information

Affected by this command:	synthesize on page 294
Affected by these attributes:	ignore_library_max_fanout
	max_capacitance
	max_fanout
	max_transition

report dft_chains

```
report dft_chains [design] [-chain scan_chain]...
     [-dft_configuration_mode dft_config_mode_name]
     [-summary] [> file]
```

Reports for every chain in the design the following elements:

- The scan data input port and its hookup pin
- The scan data output port and its hookup pin
- The shift enable port and its hookup pin
- The DFT clock domain and the DFT clock domain edge of the chain (for muxed scan only)
- The length of the chain
- The elements in the chain

  In case of the muxed scan style, the report also lists for each element the test clock and test clock edge determined by the `check_dft_rules` command for that element.
- Any user-specified segments with their elements

  In case of the muxed scan style, this includes data lockup elements, and the location of the data lockup element in the chain.
- The bit position and length of any user-specified segment in the chain
- The head and tail test clock and test clock edge for any abstract segment in the chain

Options and Arguments

`-chain` *`scan_chain...`*

Reports only the listed scan chains

`design` Specifies the design for which you want to generate a report.

Note: If you have multiple top designs, you must either specify the design name, or change to the directory in the design hierarchy that contains the design for which you want the report.

`-dft_configuration_mode` *`dft_configuration_mode_name`*

Reports the scan chains related to the specified scan mode name.

`file`	Specifies the name of the file to which to write the report.
`-summary`	Condenses the scan chain report to include only chain name, scan-data pins, shift enable, clock domain and length information.

Examples

- The following example shows one scan chain, with three user-defined segments.

```
rc:/designs/test> report dft_chains
Reporting 1 scan chain

Chain 1: DFT_chain_1
  scan_in:      in[0]
  scan_out:     out[1] (shared output)
  shift_enable: SE (active high)
clock_domain: clk (edge: mixed)
  length: 8
    START segment segHead (type: floating)
      # @ bit 1, length: 2
     bit 1     out_reg_4 <clk/fall>
     bit 2     out_reg_5 <clk/fall>
    END segment segHead
    bit 3       out_reg_6 <clk/fall>
    bit 4       out_reg_7 <clk/fall>
    START segment segBody (type: fixed)
      # @ bit 5, length: 2
      bit 5     out_reg_1 <clk/rise>
      bit 6     out_reg_3 <clk/rise>
    END segment segBody
    bit 7       out_reg_2
    START segment segTail (type: floating)
      # @ bit 8, length: 1
      bit 8     out_reg_0 <clk/rise>
    END segment segTail
------------------------
```

- The following example shows the summary report for the previous example.

```
rc:/designs/test> report dft_chains -summary
Reporting 1 scan chain (muxed_scan)

Chain 1: DFT_chain_1
  scan_in:      in[0]
  scan_out:     out[1] (shared output)
  shift_enable: SE (active high)
  clock_domain: clk (edge: mixed)
  length: 8
------------------------
```

- The following examples show the scan chains of a design before and after DFT compression:
  - **Before compression**

```
Reporting 2 scan chains (muxed_scan)

Chain 1: AutoChain_1
  scan_in:      DFT_sdi_1
  scan_out:     DFT_sdo_1
  shift_enable: se (active high)
  clock_domain: clk (edge: rise)
  length: 10
    bit 1       out_reg[1]  <clk (rise)>
    bit 2       out_reg[2]  <clk (rise)>
    bit 3       out_reg[3]  <clk (rise)>
    bit 4       out_reg[4]  <clk (rise)>
    bit 5       out_reg[5]  <clk (rise)>
    bit 6       out_reg[6]  <clk (rise)>
    bit 7       out_reg[7]  <clk (rise)>
    bit 8       out_reg[8]  <clk (rise)>
    bit 9       out_reg[9]  <clk (rise)>
    bit 10      out_reg[10]  <clk (rise)>
------------------------
Chain 2: AutoChain_2
  scan_in:      DFT_sdi_2
  scan_out:     DFT_sdo_2
  shift_enable: se (active high)
  clock_domain: clk (edge: rise)
  length: 10
    bit 1       out_reg[11]  <clk (rise)>
    bit 2       out_reg[12]  <clk (rise)>
    bit 3       out_reg[13]  <clk (rise)>
    bit 4       out_reg[14]  <clk (rise)>
    bit 5       out_reg[15]  <clk (rise)>
    bit 6       out_reg[16]  <clk (rise)>
    bit 7       out_reg[17]  <clk (rise)>
    bit 8       out_reg[18]  <clk (rise)>
    bit 9       out_reg[19]  <clk (rise)>
    bit 10      out_reg[20]  <clk (rise)>
------------------------
```

  - **After compression**

    The report shows in addition

    - How the original scan chains have been divided in several internal chains.
    - For each internal chain the `START` and `END` (separate scan data input and output) are given together with the length of the internal channel.
    - An additional scan chain, `mask_chain` (if the user opted to add making logic)

      The mask chain is comprised of abstract segments only.

      The shift-enable signal of the mask chain is reported as `NONE` because it is a *connected* shift enable.

```
Reporting 3 scan chains (muxed_scan)

Chain 1: AutoChain_1 (compressed)
  scan_in:      DFT_sdi_1
  scan_out:     DFT_sdo_1
  shift_enable: se (active high)
  clock_domain: clk (edge: rise)
  length: 10
      <START compressed internal chain AutoChain_1_0 (sdi: g121/SWBOX_SI[0])>
    bit 1       out_reg[1]  <clk (rise)>
    bit 2       out_reg[2]  <clk (rise)>
      <END   compressed internal chain AutoChain_1_0 (sdo: g121/SWBOX_SO[0])
(length: 2)>
      <START compressed internal chain AutoChain_1_1 (sdi: g121/SWBOX_SI[1])>
    bit 3       out_reg[3]  <clk (rise)>
    bit 4       out_reg[4]  <clk (rise)>
      <END   compressed internal chain AutoChain_1_1 (sdo: g121/
SWBOX_SO[1]) (length: 2)>
      <START compressed internal chain AutoChain_1_2 (sdi: g121/SWBOX_SI[2])>
    bit 5       out_reg[5]  <clk (rise)>
    bit 6       out_reg[6]  <clk (rise)>
      <END   compressed internal chain AutoChain_1_2 (sdo: g121/SWBOX_SO[2])
(length: 2)>
      <START compressed internal chain AutoChain_1_3 (sdi: g121/SWBOX_SI[3])>
    bit 7       out_reg[7]  <clk (rise)>
    bit 8       out_reg[8]  <clk (rise)>
      <END   compressed internal chain AutoChain_1_3 (sdo: g121/SWBOX_SO[3])
(length: 2)>
      <START compressed internal chain AutoChain_1_4 (sdi: g121/SWBOX_SI[4])>
    bit 9       out_reg[9]  <clk (rise)>
    bit 10      out_reg[10]  <clk (rise)>
      <END   compressed internal chain AutoChain_1_4 (sdo: g121/SWBOX_SO[4])
(length: 2)>
------------------------
Chain 2: AutoChain_2 (compressed)
  scan_in:      DFT_sdi_2
  scan_out:     DFT_sdo_2
  shift_enable: se (active high)
  clock_domain: clk (edge: rise)
  length: 10
      <START compressed internal chain AutoChain_2_5 (sdi: g121/SWBOX_SI[5])>
    bit 1       out_reg[11]  <clk (rise)>
    bit 2       out_reg[12]  <clk (rise)>
      <END   compressed internal chain AutoChain_2_5 (sdo: g121/SWBOX_SO[5])
(length: 2)>
...
...
      <START compressed internal chain AutoChain_2_9 (sdi: g121/SWBOX_SI[9])>
    bit 9       out_reg[19]  <clk (rise)>
    bit 10      out_reg[20]  <clk (rise)>
      <END   compressed internal chain AutoChain_2_9 (sdo: g121/SWBOX_SO[9])
(length: 2)>
------------------------
Warning - could not find shift_enable signal for chain /designs/top/dft/
report/actual_scan_chains/mask_chain
Chain 3: mask_chain
  scan_in:      msi
  scan_out:     mso
  shift_enable: NONE
  clock_domain: mck (edge: rise)
  length: 10
```

```
        START segment DFT_segment_1 (type: abstract)
          # @ bit 1, length: 4
          pin       g121/msi[0] <mck (rise)>
          pin       g121/mso[0] <mck (rise)>
        END segment DFT_segment_1
        START segment DFT_segment_3 (type: abstract)
          # @ bit 5, length: 3
          pin       g121/msi[2] <mck (rise)>
          pin       g121/mso[2] <mck (rise)>
        END segment DFT_segment_3
        START segment DFT_segment_2 (type: abstract)
          # @ bit 8, length: 3
          pin       g121/msi[1] <mck (rise)>
          pin       g121/mso[1] <mck (rise)>
        END segment DFT_segment_2
    ------------------------
```

Related Information

Reporting on All Scan Chains in *Design for Test in Encounter RTL Compiler*

Compressing Scan Chains in *Design for Test in Encounter RTL Compiler*

Affected by this command:	connect_scan_chains on page 523
	compress_scan_chains on page 508
Related attributes:	Actual Scan Chain
	Actual Scan Segment

report dft_registers

```
report dft_registers [-pass_tdrc] [-fail_tdrc]
     [-lockup] [-latch] [-dont_scan] [-misc]
     [-shift_reg] [design] [> file]
```

Reports the DFT status of all flip-flop instances in the design. Use this command after running `check_dft_rules`. More specifically, the command reports

- Registers which pass the DFT rule checker

  For each flip-flop, it reports the hierarchical instance name along with their DFT test clock domain and active edge.

- Registers which fail the DFT rule checker

  For each flip-flop, it reports the hierarchical instance name along with the violation type (clock, or asynchronous set/reset) and the violation Id number.

  For an abstract segment that fails the DFT rule checker, it reports the name of the abstract segment, and the number of flip-flops in the segment.

- Registers that are preserved or marked `dont-scan`

  Note: Mapped flip-flops can be listed in this category if

  - The flip-flop is marked preserved and it is mapped to a non-scan flop

    However, if a flip-flop is marked preserved and is already mapped to scan for DFT purposes, it will be listed as either passing or failing the DFT rule checker—based on the DFT rule checker analysis—and it will not be listed in this category.

  - The flip-flop is mapped to a scan flop for non-DFT purposes

- Registers that are marked Abstract Segment dont-scan.
- Registers that are part of shift register segments
- Registers that are identified as lockup elements
- Registers that are level-sensitive elements
- Registers not part of any of the other categories

Without any options specified, the command reports on all categories of registers.

Options and Arguments

`design`	Specifies the design for which you want to generate a report. By default a report is created for all designs currently loaded in memory.
`-dont_scan`	Reports all edge-triggered registers that are not to be mapped to scan flops, or connected into the top-level chains.
`-fail_tdrc`	Reports all edge-triggered registers that fail the DFT rule checks.
`file`	Specifies the name of the file to which to write the report.
`-latch`	Reports all registers of type latch in the design. **Note:** Latch instances which are instantiated as lockup elements in a preserved segment are reported with the `-lockup` option.
`-lockup`	Reports data lockup elements of type latch or flop that are either instantiated in a preserved segment, or added to the design during scan chain configuration.
`-misc`	Reports all registers that are not reported through any of the other options, such as clock-gating registers.
`-pass_tdrc`	Reports all edge-triggered registers that pass the DFT rule checks.
`-shift_reg`	Reports all registers that are part of shift register segments.

Example

- The following example shows that 1 flip-flop passed the DFT rule checks, while 4 flip-flops failed the tests.

```
rc:/> report dft_registers
```

```
Reporting registers that pass DFT rules
  Iset_reg      PASS; Test clock: clk/rise
Reporting registers that fail DFT rules
  out_reg_0     FAIL; violations: clock #(0 ) async set #(1 )
  out_reg_1     FAIL; violations: clock #(0 ) async set #(1 )
  out_reg_2     FAIL; violations: clock #(0 ) async set #(1 )
  out_reg_3     FAIL; violations: clock #(0 ) async set #(1 )
Reporting registers that are preserved or marked dont-scan
Reporting registers that are marked Abstract Segment Dont Scan
Reporting registers that are part of shift register segments
Reporting registers that are identified as lockup elements
Reporting registers that are level-sensitive elements
Reporting misc. non-scan registers
Summary:
Total registers that pass DFT rules: 1
Total registers that fail DFT rules: 4
Total registers that are marked preserved or dont-scan: 0
Total registers that are marked Abstract Segment dont-scan: 0
Total registers that are part of shift register segments: 0
Total registers that are lockup elements: 0
Total registers that are level-sensitive: 0
Total registers that are misc. non-scan: 0
```

- The following report was generated after the scan configuration engine was run. In this case, four lockup elements were inserted:

```
rc:/> report dft_registers
```

```
Reporting registers that pass DFT rules
  out1_reg_0    PASS; Test clock: test_clk1/rise
  ...
  out1_reg_4    PASS; Test clock: test_clk1/fall
   ...
  out2_reg_0    PASS; Test clock: test_clk2/rise
   ...
  out2_reg_4    PASS; Test clock: test_clk2/fall
  ...
  out3_reg_0    PASS; Test clock: test_clk3/rise
  ...
  out3_reg_4    PASS; Test clock: test_clk3/fall
  ...
Reporting registers that fail DFT rules
Reporting registers that are preserved or marked dont-scan
Reporting registers that are marked Abstract Segment Dont Scan
Reporting registers that are part of shift register segments
Reporting registers that are identified as lockup elements
  DFT_lockup_g1
  DFT_lockup_g348
  DFT_lockup_g349
  DFT_lockup_g350
Reporting registers that are level-sensitive elements
Reporting misc. non-scan registers
Summary:
Total registers that pass DFT rules: 27
Total registers that fail DFT rules: 0
Total registers that are marked preserved or dont-scan: 0
Total registers that are marked Abstract Segment dont-scan: 0
Total registers that are part of shift register segments: 0
Total registers that are lockup elements: 4
Total registers that are level-sensitive: 0
Total registers that are misc. non-scan: 0
```

Related Information

Reporting Status of Flip-Flops in *Design for Test in Encounter RTL Compiler.*

Affected by this command:	check_dft_rules on page 501
Related attributes:	dft_dont_scan
	dft_scan_style
	dft_status
	dft_violation
	preserve (instance)
	preserve (subdesign)
	Scan Segment Attributes

report dft_setup

```
report dft_setup [design] [> file]
```

Reports DFT setup information for the design including both user-defined and tool-inferred information.

You must load and elaborate a design before you can use this command. The contents of this report further depends on the stage of the design that you are at.

Use this command at the end of the design process to document the DFT setup and resulting configuration of the design.

Options and Arguments

design	Specifies the design for which you want to generate a report. By default a report is created for all designs currently loaded in memory.
file	Specifies the name of the file to which to write the report.

Examples

- The following example shows the report after you have elaborated the design. Because you have not specified any setup information yet, the information shown is based on the default settings for DFT.

```
rc:/> report dft_setup

Design Name
===========
   top

Scan Style
==========
    muxed_scan
DFT rule check status is not available. Need to (re)run check_dft_rules

Global Constraints
==================
    Minimum number of scan chains: no_value
    Maximum length of scan chains: no_value
    Lock-up element type: level_sensitive
    Mix clock edges in scan chain: false
    Prefix for unnamed scan objects: DFT_

Test signal objects
===================
Test clock objects
==================
Reporting all test clocks as TDRC status is not available

DFT controllable objects
========================

DFT don’t scan objects
======================

DFT abstract don’t scan objects
===============================

DFT scan segment constraints
============================

DFT scan chain constraints
==========================

DFT actual scan chains
======================
```

■ The following example shows the report after the scan configuration engine was run:

```
rc:/designs/test> report dft_setup

Design Name
===========
   top

Scan Style
==========
    muxed_scan
Design has a valid DFT rule check status

Global Constraints
==================
    Minimum no of Scan chains: no_value
    Maximum length of scan chains: no_value
    Lock-up element type: level_sensitive
    Mix clock edges in scan chain: true
    Prefix to name user defined scan chains: DFT_

Test signal objects
==================
    shift_enable:
          object name: SE
          pin name: SE
          hookup_pin: SE
          hookup_polarity: non_inverted
          active: high
          ideal: true
          user defined: true


Test clock objects
==================
    test_clock:
          object name: clk
          user defined: false
          source: clk
          root source: clk
          root source polarity: non_inverting
          hookup_pin: clk
          period: 50000.0


DFT controllable objects
========================

DFT don’t scan objects
======================

DFT abstract don’t scan objects
===============================

DFT scan segment constraints
============================
    Segment:
          object name: segHead
          scan-in:
          scan-out:
          shift-enable: internal
          connected_shift_enable: false
          length: 2
          type: floating

    Segment:
         object name: segTail
         ...

    Segment:
          object name: segBody
          scan-in:
          scan-out:
          shift-enable: internal
          connected_shift_enable: false
          length: 2
          type: fixed


DFT scan chain constraints
==========================
    User Chain:
          object name: DFT_chain_1
          scan-in: in[0]
          scan-in hookup_pin: in[0]
          scan-out: out[1]
          scan-out hookup_pin: out[1]
          shared out: true
          shift_enable object name:
          max length: no_value
          head segment: segHead
          tail segment: segTail
          complete: false


DFT actual scan chains
======================
    Actual Chain:
          object name: DFT_chain_1
          scan-in: in[0]
```

```
scan-in hookup_pin: in[0]
scan-out: out[1]
scan-out hookup_pin: out[1]
shared out: true
shift_enable: SE
length: 8
segment objects: segHead segBody segTail
analyzed: false
test_clock domain: clk
test_clock edge: any
```

Related Information

Affected by these constraints:	define_dft shift_enable on page 572
	define_dft test_clock on page 577
	define_dft test_mode on page 581
Affected by this command:	check_dft_rules on page 501
	connect_scan_chains on page 523
Related attributes:	dft_controllable
	dft_dont_scan
	dft_lockup_element_type
	dft_max_length_of_scan_chains
	dft_min_number_of_scan_chains
	dft_mix_clock_edges_in_scan_chains
	dft_prefix
	dft_scan_style
	(instance) preserve
	(subdesign) preserve

Related attributes:	Actual Scan Chain Attributes
	Actual Scan Segment Attributes
	Scan Segment Attributes
	Scan Chain Attributes
	Test Clock Attributes
	Test Signal Attributes

report dft_violations

```
report dft_violations [-async] [-clock]
     [-abstract] [-shiftreg] [-tristate]
     [-race] [-xsource] [-xsource_by_instance]
     [design] [> file]
```

Reports for each violation the object name, the type of violation, the description of the violation, how to find the root pin or port of the problem, the source file and the line number where to find the problem, the number of registers affected, and the instance names of the affected registers. The report is sorted by violation type. Run the DFT rule checker to get meaningful information.

Without any options specified, the command reports on all types of violations.

Options and Arguments

`-abstract`	Reports all abstract segment test mode violations.
`-async`	Reports all async set and async reset violations.
`-clock`	Reports all clock violations.
`design`	Specifies the design for which you want to generate a report. By default a report is created for all designs currently loaded in memory.
`file`	Specifies the name of the file to which to write the report.
`-race`	Specifies to report potential race condition violations.
`-shiftreg`	Reports all shift register segment violations.
`-tristate`	Reports tristate design rules checking violations.
`-xsource`	Reports X-Source violations.
`-xsource_by_instance`	Reports X-Source violations by instance.

Examples

- The following example shows that the design has four violations, but only detailed information on the clock violations is requested.

```
rc:/> report  dft_violations -clock
Total 4 violations (muxed_scan)

Clock Rule Violations
=====================
Reporting 2 clock violations

Violation #0:
  Object name: vid_0_clock
  Type: clock violation
  Description: [CLOCK-05] internal or gated clock signal
  Source: g1/z (test10.v:12)
  Number of registers affected: 4
  Affected registers:
    out1_reg[0]
    out1_reg[1]
    out1_reg[2]
    out1_reg[3]

Violation #1:
  Object name: vid_1_clock
  Type: clock violation
  Description: [CLOCK-06] clock signal driven by a sequential element
  Source: divClk_reg/q (test10.v:14)
  Number of registers affected: 4
  Affected registers:
    out2_reg[0]
    out2_reg[1]
    out2_reg[2]
    out2_reg[3]

Violation Rule Summary Report
```

```
=============================
[CLOCK-05] internal or gated clock signal : 1
[CLOCK-06] clock signal driven by a sequential element : 1
```

- The following example reports one tristate net contention violation.

```
rc:/> report dft_violations
Total 1 violation (muxed_scan)

Tristate Net Violations
=====================
Reporting 1 tristate net contention violations

Violation #0:
 Object name: vid_0_tristate_net
 Type: tristate net violation
 Description: [TRISTATE_NET-01] tristate net potentially driven by conflicting
values

  Tristate net affected: tbus
  Tristate net load pin affected: tbus
  Tristate drivers causing contention: b2/Y b1/Y b3/Y b4/Y b6/Y b8/Y b7/Y b5/Y
...
Violation Rule Summary Report
=============================
[TRISTATE_NET-01] tristate net potentially driven by conflicting values : 1
```

Related Information

Affected by this command:	check_dft_rules on page 501
	fix_dft_violations on page 586
Related attributes:	Violations Attributes

report disabled_transparent_latches

```
report disabled_transparent_latches [> file]
```

Reports the disabled transparent latches and the `enable` to `Q` arcs that are not yet disabled. Transparent latches are latches with `enable` signal held constant at the active state. Without enabling transparent latches, paths through them cannot be traced.

Options and Arguments

`file`	Specifies the name of the file to which to write the report.

Related Information

Affected by this command:	enable_transparent_latches on page 64

report gates

```
report gates [-library_domain library_domain_list]
     [-power] [-yield]
     [-instance_hier instance] [design] [> file]
```

Reports the technology library cells that were implemented (and identifies their originating libraries), the area of the cell instances, and the break up of the instances into timing models, sequential cells, inverters and logic gate cells. Optionally power information can be reported.

Note: Timing models can refer to memory cells, IPs, integrated clock-gating cells, and so on.

Options and Arguments

`design`	Specifies the block for which you want to generate a report. You can also `cd` into the particular design directory and generate the report.
`file`	Specifies the name of the file to which to write the report.
`-instance_hier` *`instance`*	Restricts the reported information to the specified hierarchical instance.
`-library_domain` *`library_domain_list`*	Restricts the reported information to the specified library domains. **Note:** This option only applies when using CPF.
`-power`	Adds leakage power and internal power information.
`-yield`	Reports the yield cost and yield percentage values for each library cell.

Examples

- The following example reports information such as the type of cells that were used, number of instances, area, and originating library of the current design.

```
rc:/> report gates
============================================================
...
  Module:                alu
  Technology library:    tutorial 1.0
  Operating conditions:  typical_case (balanced_tree)
  Wireload mode:         enclosed
============================================================

  Gate  Instances   Area     Library
---------------------------------------
flopdrs         9   72.000    tutorial
inv1           35  105.000    tutorial
nand2         112  112.000    tutorial
nor2           37   55.500    tutorial
xor2           17   34.000    tutorial
---------------------------------------
total         210  378.500

   Type    Instances   Area   Area %
-------------------------------------
sequential         9   72.000   19.0
inverter          35  105.000   27.7
logic            166  201.500   53.2
-------------------------------------
total            210  378.500  100.0
```

- The following example shows the additional power information in the report.
    - ❑ The first table lists the leakage and internal power of all instances per used library cell.
    - ❑ The second table shows the number and percentage of instances coming from each library, the leakage and internal power consumed by these instances, as well as the percentage of power consumed by these instances.
    - ❑ The third table shows the leakage and internal power of all sequential cells, inverters, and combinational cells, as well as the percentage of power that each type consumes.

```
rc:/> report gates -power
============================================================
...
============================================================

                                    Leakage     Internal
    Gate    Instances     Area     Power (nW)  Power (nW)   Library
----------------------------------------------------------------------
ACHCONX2TH          1      28.856      59.948     372.198    slow_hvt
ADDFHX1             1      39.040     158.364     276.327    slow
...
XOR2XL             10     118.820     339.664     471.216    slow
```

```
XOR2XLTH          449    5335.018    6259.161   22146.660    slow_hvt
XOR3XL              1      28.856      83.655      80.446    slow
----------------------------------------------------------------------
total           11980  109000.238  139743.990  505460.121

                               Leakage  Leakage   Internal  Internal
 Library Instances Instances % Power (nW) Power %  Power (nW)  Power %
-----------------------------------------------------------------------
slow         1919         16.0  74607.161    53.4  168629.031     33.4
slow_hvt     10061        84.0  65136.830    46.6  336831.090     66.6

                                      Leakage  Leakage  Internal  Internal
  Type    Instances    Area    Area % Power (nW) Power %  Power (nW)  Power %
-----------------------------------------------------------------------------
sequential       212   7864.054    7.2  17285.068   12.4  112374.696     22.2
inverter        1684   5961.269    5.5   5752.468    4.1   22492.255      4.4
buffer            37    364.941    0.3   1877.474    1.3    3920.823      0.8
logic          10047  94809.9748  87.0 114828.979   82.2  366672.348     72.5
-----------------------------------------------------------------------------
total          11980 109000.2382  100.0 139743.990 100.0  505460.121    100.0
```

- The following example reports the yield cost and yield percentage values for each library cell:

```
Gate  Instances   Area       Cost       Yield    Library

------------------------------------------------------------

flopdrs        33  264.000  3.39278e-06  99.9997    tutorial

inv1          103   51.500   1.5022e-06  99.9998    tutorial

nand2         315  315.000  1.08311e-05  99.9989    tutorial

nor2           19   28.500  6.79989e-07  99.9999    tutorial

------------------------------------------------------------

total         470  659.000  1.64061e-05  99.9984


   Type    Instances   Area  Area %

-----------------------------------

sequential        33 264.000   40.1

inverter         103  51.500    7.8

logic            334 343.500   52.1

-----------------------------------

total            470 659.000  100.0
```

Related Information

Analyze Design in *Low Power in Encounter RTL Compiler*

Affected by this command: synthesize on page 294

report hierarchy

```
report hierarchy [design] [> file]
```

Generates a report based on the design hierarchy starting from the top level module down to the leaf module. The report will take the following format:

```
level instance ( module ) <status>
status  : preserve_<value> -- indicating preserve hierachy or inherited_preserve
value
        : blackbox --  indicating  unresolved instance
```

Note: The `hdl_track_filename_row_col` attribute needs to be set to `true` before elaboration in order to successfully use this command.

Options and Arguments

design	Specifies a specific design to report when there are multiple designs.
file	Specifies the name of the file to which to write the report.

Example

- The following example reports the hierarchy of desing `m1`:

```
============================================================
Hierarchy Report Format :

level instance ( module ) <status>

status :    preserve_<value> -- indicating preserve hierachy or
inherited_preserve value
       :    blackbox --  indicating  unresolved instance
==================================================================

 0 m1
  1 m2 ( m2 )
     2 m3 ( m3 )
        3 m3_0 ( m3_0 )
           4 m3_0_0 ( m3_0_0 )
        3 m4 ( m4 )
           4 m5 ( m5 )
              5 m5_bbox ( m5_bbox ) blackbox
           4 m6 ( m6 )
     2 m2myclk ( m2myclk )
```

Related Information

Affected by this attribute:	`hdl_track_filename_row_col`

report instance

```
report instance [-timing] [-power] instance... [> file]
```

Generates a report on the instances of the current design. By default, the report gives timing related information on the instances. Get power related information using the `-power` option.

Options and Arguments

`file`	Specifies the name of the file to which to write the report.
`instance`	Specifies the name of a leaf instance for which to generated the report.
`-power`	Reports instance leakage, internal power, net power and the computed probability, toggle rate, and net power on the nets of the instance, and the activity and power for each of the arcs. **Note:** In case the switching activities are user-asserted, the values are appended with an asterisk (*).
`-timing`	Reports timing information, such as slew-in, load, slew-out, delay, and slack.

Example

■ The following example reports timing information for the `n0000D3666` instance:

```
rc:/> report instance -timing -power n0000D3666
============================================================
  ...
  Module:                dpldalgn
  Technology library:    tutorial 1.0
  Operating conditions:  typical_case (balanced_tree)
  Wireload mode:         enclosed
============================================================

         Instance Timing Info
         --------------------
 Instance n0000D3666 of libcell nor2

 Arc  Arc  Slew        Slew
From  To    In   Load  Out   Delay   Slack
--------------------------------------------

 A r  Y f  46.2  20.7  57.2  136.0  -1022.4
 A f  Y r  46.2  20.7  57.2  136.0  -1022.4
 B r  Y f  16.5  20.7  57.2  134.0   -900.2
 B f  Y r  16.5  20.7  57.2  134.0   -900.2
```

- The following example shows the timing and power information for instance `o_m2_clk2_1_reg_0`.

```
============================================================
  ...
  Module:                  m1
  Technology library:      slow 1.0
  Operating conditions:    slow (balanced_tree)
  Wireload mode:           enclosed
============================================================
 Instance m2/o_m2_clk2_1_reg_0 of libcell EDFFX1

 Arc   Arc  Slew        Slew
 From   To   In   Load   Out   Delay Slack
-------------------------------------------
 CK r   Q f  0.0  10.0  173.9  387.0   inf
 CK f   Q r  0.0  10.0  187.3  429.0   inf
  D r   Q f  0.0  10.0  173.9          inf
  D f   Q r  0.0  10.0  187.3          inf
  E r   Q f  0.0  10.0  173.9          inf
  E f   Q r  0.0  10.0  187.3          inf
 CK r  QN f  0.0   0.0   64.3          inf
 CK f  QN r  0.0   0.0   61.3          inf
  D r  QN f  0.0   0.0   64.3          inf
  D f  QN r  0.0   0.0   61.3          inf
  E r  QN f  0.0   0.0   64.3          inf
  E f  QN r  0.0   0.0   61.3          inf

        Instance Power Info
        -------------------

 Instance m2/o_m2_clk2_1_reg_0 of Libcell EDFFX1

  Leakage    Internal      Net
 Power(nW)   Power(nW)  Power(nW)
---------------------------------
      33.7     20470.8       58.3

                           Computed          Computed          Net
Pin         Net           Probability    Toggle Rate(/ns)    Power(nW)
--------------------------------------------------------------------
Q     o_m2_clk2_1[0]           0.4             0.0100          58.3
CK    n_0                      0.5             1.8725        4914.2
D     in_2[21]                 0.5             0.0200         133.0
E     en_2[7]                  0.5             0.0200         454.9
CK    n_0                      0.5             1.8725        4914.2
D     in_2[21]                 0.5             0.0200         133.0
E     en_2[7]                  0.5             0.0200         454.9


 Arc  Arc                 Arc        Arc
From   To      When    Activity   Power(nW)
-------------------------------------------
CK     CK   !D & !E      0.468      9395.6
CK     CK   D & !E       0.468      9563.0
CK     CK   !D & E       0.468     11860.1
CK     CK   D & E        0.468     11972.6
D      D                 0.020      7725.6
E      E                 0.020     10293.1
CK     Q                 0.010      7867.9
```

report isolation

```
report isolation
     { [instance_list]
     | [-detail] [-hierarchical]
       [-from_power_domain power_domain...]
       [-to_power_domain power_domain...] }
     [> file]
```

Reports isolation cell information for the design. The return value of the report corresponds to the number of leaf isolation cell instances found. The information returned depends on your current *position* in the design hierarchy.

Options and Arguments

`-detail`	Requests a detailed isolation cell report. A detailed report shows the names of the hierarchical isolation cell instances, the power domains they are inserted between, the power domain they are stored with, the type of isolation cell, and the number of leaf isolation cell instances they contain. The following types of cells can be distinguished: ■ L—Level shifter library cell with isolation logic ■ I—Isolation library cell ■ D—Discrete isolation cell (can contain an `AND gate`, or `OR` gate, or latch, and possibly an inverter)
`file`	Specifies the name of the file to which to write the report.
`-from_power_domain` *`power_domain`*	Reports all isolation cells whose drivers are output pins of instances in the specified power domains.
`-hierarchical`	Traverses the hierarchy starting from the current module and reports all the isolation cell instances found in the current module and its children modules.
`instance_list`	Reports detailed information for the specified hierarchical isolation cell instances. The detailed report lists ■ For the hierarchical instance: the from and to domain, the location, the type of the isolation cells, the enable driver (if it can be determined), and for discrete types, the function. ■ For each leaf cell: the driver and the load.

Note: The report by instance also shows the type of the isolation cell. In addition to the three types that are distinguished with the detailed report, the report by instance can also distinguish two other types:

- Generic—Refers to a user-specified (isolation) module that contains (an) unmapped cell(s).
- Complex—Refers to a user-specified (isolation) module that contains complex gates, or a hierarchy that contains mixed cell types, such as a pure isolation cell and an enabled level shifter.

  These two types cause a warning in the detailed isolation report.

`-to_power_domain` *`power_domain`*

Reports all isolation cells whose output pins are driving instances in the specified power domains.

Examples

- The following command shows the basic report for the top-level design. The design has four power domains, four hierarchical isolation cell instances and four leaf isolation cell instances were inserted.

```
rc:/> report isolation
============================================================
  Generated by:          Encounter(r) RTL Compiler version
  Generated on:          date
  Module:                top
  ...
============================================================

               Category
============================================
Unique power domains                    4
--------------------------------------------
Isolation Cell hierarchical instances    4
Isolation Cell instances                4
============================================

4
```

- The following command shows the detailed report for the design and reports all isolation instances found in the hierarchy starting from the top level.

```
rc:/> report isolation -detail -hierarchical
...
       Isolation Cell          From      To     Location    Type    Number of
         instance             domain    domain                        cells
============================================================================
mux_10_14RC_ISO_HIER_INST_22     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_23     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_24     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_26     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_27     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_28     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_29     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_30     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_31     p1        p2        p2        D           1
mux_10_14RC_ISO_HIER_INST_32     p1        p2        p2        D           1
RC_ISO_HIER_INST_19              p3        p4        p1        D           1
RC_ISO_HIER_INST_20              p3        p4        p1        D           1
RC_ISO_HIER_INST_25              p3        p4        p1        D           1
RC_ISO_HIER_INST_21              p3        p2        p3        D           1
                                           p4
---------------------------------------------------------------------------
 Summary
=========
               Category
============================================
Unique power domains                     4
--------------------------------------------
Isolation Cell hierarchical instances    14
Isolation Cell instances                 14
============================================

14
```

- The following command shows the report for instance `RC_ISO_HIER_INST_19`.

```
rc:/> report isolation RC_ISO_HIER_INST_19
...
Name:             RC_ISO_HIER_INST_19
From domain(s):      p3
To domain(s):        p4
Location:            p1
Type:                Discrete
Function:            Enable: active_high; Output: low
Enable:              s
Details:

  Isolation                  Driver                      Load
     cell                instance/pin(s)            instance/pin(s)
   instance
=======================================================================
RC_ISO_AND             myInsti/y_reg[2]/Q          y[2]
RC_ISO_NOT

-----------------------------------------------------------------------

1
```

report level_shifter

```
report level_shifter
     { [instance_list]
     | [-detail] [-hierarchical]
       [-from_power_domain power_domain_list]
       [-to_power_domain power_domain_list] }
     [-verbose] [> file]
```

Reports level-shifter information for the design. The return value of the report corresponds to the number of leaf level-shifter instances found.

Options and Arguments

`-detail`	Requests a detailed level shifter report.
`file`	Specifies the name of the file to which to write the report.
`-from_power_domain` *`power_domain_list`*	Reports all level shifters whose drivers are output pins of instances in the specified power domain.
`-hierarchical`	Reports level shifters hierarchically. If this option is omitted, reports all level shifters at the current level of the hierarchy.
`instance_list`	Reports detailed information for the specified hierarchical level-shifter instances at the current level.
`-to_power_domain` *`power_domain_list`*	Reports all level shifters whose output pins are driving instances in the specified power domain.
`-verbose`	Reports complete information if it has been not shown with `-detail` option.

Examples

- The following command shows the report for the top-level design. The design has three power domains, one hierarchical level-shifter instance stored in the power domain for the top design, and 2 leaf level-shifter instances.

```
rc:/> report level_shifter
============================================================
  Generated by:           Encounter(R) RTL Compiler version
  Generated on:           date
  Module:                 soc
  Library domain:         umc_08v
    Domain index:         0
    Technology libraries: scmetro_umcl130e_sp_tt_0p8v_25c 1.0
                          scmetro_umcl130e_ll_tt_0p8v_25c 1.0
                          scmetropmk_umc13sp_tt_0p8v_25c 1.0
                          scmetropmk_umc13sp_tt_0p8v_1p2v_25c 1.0
    Operating conditions: _nominal_ (balanced_tree)
  Library domain:         umc_120v
    Domain index:         1
    Technology libraries: scmetropmk_umc13sp_tt_0p8v_1p2v_25c 1.0
                          scmetro_umcl130e_ll_tt_1p2v_25c 1.0
                          scmetro_umcl130e_sp_tt_1p2v_25c 1.0
    Operating conditions: _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================


Summary
-------

Unique Power Domains : 3
======================================================================
Domain Interactions
                                              Level Shifter   Level Shifter
 From Power Domain    To Power Domain    Hierarchies      Instances
======================================================================
PD3 (0.8v-1.2v)      PD1 (0.8v)               1               2
======================================================================
                     Total:                   1               2
======================================================================
2
```

- The following report requests a hierarchical report. This report indicates that the design uses a total of three power domains, has two hierarchical level-shifter instances and 11 leaf level-shifter instances inserted.

```
rc:/> report level_shifter -hier
============================================================
...
Summary
-------

Unique Power Domains : 3
======================================================================
Domain Interactions
                                        Level Shifter   Level Shifter
 From Power Domain    To Power Domain    Hierarchies      Instances
======================================================================
PD1 (0.8v)            PD2 (0.8v-1.2v)        1              4
PD1 (0.8v)            PD3 (0.8v-1.2v)        1              4
PD2 (0.8v-1.2v)       PD3 (0.8v-1.2v)        1              1
PD3 (0.8v-1.2v)       PD1 (0.8v)             1              2
======================================================================
                      Total:                 4              11
======================================================================
11
```

- The following command shows the detailed report of the level shifters between power domain PD2 and PD3. The report shows the name of the hierarchical level-shifter instance and the hierarchical instance (module) it was inserted in.

```
rc:/> report level_shifter -from_power PD2 -to_power PD3 -detail -hier
...
==========================================================================================
     From               To            Module          Level          Instances   Location
    Power             Power                          Shifter
    Domain            Domain                        Hierarchy
   (Range)           (Range)
==========================================================================================
PD2 (0.8v-1.2v)   PD3 (0.8v-1.2v)   socsync_det   CPF_LS_HIER_INST_27   1          to
==========================================================================================


Summary
-------

Unique Power Domains : 3
======================================================================
Domain Interactions
                                        Level Shifter   Level Shifter
 From Power Domain    To Power Domain    Hierarchies      Instances
======================================================================
PD2 (0.8v-1.2v)       PD3 (0.8v-1.2v)        1              1
======================================================================
                      Total:                 1              1
======================================================================
1
```

- The following command shows the report for level-shifter instance `CPF_LS_HIER_INST_21`. The detailed report lists for each leaf level shifter instance all pins that it is connecting.

```
rc:/> report level_shifter [find / -inst CPF_LS_HIER_INST_21]
...
Name:       cg_pream/CPF_LS_HIER_INST_21
From Power Domain:    PD1 (0.8v-0.8v)
To Power Domain:      PD2 (0.8v-1.2v)
Location:             to
Instances:            4
Details:

       Level                      From                           To                  Is
      shifter                 instance/pin                  instance/pin          dedicated
     instance
===========================================================================================
        g1                      /soc/pd             cg_pream/count_reg[1]/RETN       true
                                                    cg_pream/count_reg[5]/RETN       true
                                                    cg_pream/count_reg[9]/RETN       true
                                                    cg_pream/count_reg[2]/RETN       true
                                                    cg_pream/count_reg[6]/RETN       true
                                                    cg_pream/count_reg[3]/RETN       true
                                                    cg_pream/count_reg[7]/RETN       true
                                                     cg_pream/prem_ok_reg/RETN       true
                                                    cg_pream/count_reg[0]/RETN       true
                                                    cg_pream/count_reg[4]/RETN       true
                                                    cg_pream/count_reg[8]/RETN       true
       g442                    /soc/ck1              cg_pream/count_reg[9]/CK        true
                                                     cg_pream/count_reg[5]/CK        true
                                                     cg_pream/count_reg[1]/CK        true
                                                     cg_pream/count_reg[6]/CK        true
                                                     cg_pream/count_reg[2]/CK        true
                                                     cg_pream/count_reg[7]/CK        true
                                                     cg_pream/count_reg[3]/CK        true
                                                     cg_pream/count_reg[8]/CK        true
                                                     cg_pream/count_reg[4]/CK        true
                                                     cg_pream/count_reg[0]/CK        true
                                                      cg_pream/prem_ok_reg/CK        true
       g443                   /soc/rec_en                cg_pream/g418/B             true
                                                        cg_pream/g415/A0             true
       g444                    /soc/rst                  cg_pream/g428/A             true
-------------------------------------------------------------------------------------------

4
```

Related Information

Affected by these commands:	commit_cpf on page 752
	read_cpf on page 760

report memory

```
report memory [> file]
```

Reports the memory resource used by the compiler in the computing platform.

Options and Arguments

file	Specifies the name of the file to which to write the report.

report memory_cells

```
report memory_cells [library] [> file]
```

Reports the memory cells in the library.

Options and Arguments

file	Specifies the name of the file to which to write the report.
library	Specifies the name of the library for which to report the memory cells. If no library is specified, the report applies to all libraries that are loaded.

Example

The following command lists the memory cells in the RF32X32_lib library.

```
rc:/> report memory_cells
```

```
Library : RF32X32_lib
Memory Cell: RF32X32
Memory Type: ram
Memory address width: 5
Memory word width: 32
Bus : Q, mode output, 31 to 0
Associated address bus is A
related_pin : CLK
timing_type : rising_edge
Bus : D, mode input, 31 to 0
Associated address bus is A
Misc Attributes clocked_on : CLK
related_pin : CLK
timing_type : hold_rising
related_pin : CLK
timing_type : setup_rising
related_pin : CLK
timing_type : rising_edge
timing_sense : non_unate
Pin : CLK, mode input
capacitance : 0.038
clock : true
max_transition : 1.000
min_pulse_width_high : 0.055
min_pulse_width_low : 0.130
min_period : 0.867
Pin : CEN, mode input
capacitance : 0.002
related_pin : CLK
timing_type : hold_rising
related_pin : CLK
```

```
timing_type : setup_rising
Pin : WEN, mode input
capacitance : 0.010
related_pin : CLK
timing_type : hold_rising
related_pin : CLK
timing_type : setup_rising
Bus : A, mode input, 4 to 0
bus_type : RF32X32_ADDRESS
capacitance : 0.008
related_pin : CLK
timing_type : hold_rising
related_pin : CLK
timing_type : setup_rising
related_pin : CLK
timing_type : hold_rising
related_pin : CLK
timing_type : setup_rising
```

report messages

```
report messages [-all] [-include_suppressed] [-error]
     [-warning] [-info] [> file]
```

Summarizes the information, warning and error messages that have been issued by RTL Compiler in the current run since the last report. The report contains the number of times the message has been issued, the severity of the message, the identification number, and the message text.

The `-all` option does not report those messages that were suppressed through the `suppress_messages` command. Specify the `-include_suppressed` option to report such messages.

By adding report messages to your Tcl prompt, RTL Compiler can summarize all messages issued during the last command right before prompting you for more input. This is useful after long commands (such as `elaborate`) that can generate many messages.

Options and Arguments

`-all`	Reports all messages since you started this RTL Compiler run.
`-error`	Reports the error messages.
`file`	Specifies the name of the file to which to write the report.
`-include_suppressed`	Reports those messages that were suppressed through the `suppress_messages` command.
`-info`	Reports the information messages.
`-warning`	Reports the warning messages.

Examples

Note: The following examples all apply to the same RTL Compiler session.

- The following example is the first request to report messages in a session.

```
rc:/> report messages

  =================
   Message Summary
  =================

Num  Sev      Id                        Message Text
------------------------------------------------------------------------
  1 Info ELAB-VLOG-9 Variable has no fanout.
                     This variable is not driving anything and will be
                     simplified
  3 Info LBR-30      Promoting a setup arc to recovery.
                     Setup arcs to asynchronous input pins are not
                     supported
  3 Info LBR-31      Promoting a hold arc to removal.
                     Hold arcs to asynchronous input pins are not
                     supported
  1 Info LBR-54      Library has missing unit.
                     Current library has missing unit.
```

- The following example executes the `report messages` command immediately after the previous `report messages` command and consequently does not return any new messages.

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

- The following example is the third `report messages` command in a row, but because the `-all` option is specified, the same output as with the first command is given:

```
rc:/> report messages -all

  =================
   Message Summary
  =================

Num  Sev      Id                         Message Text
------------------------------------------------------------------------
  1 Info ELAB-VLOG-9 Variable has no fanout.
                     This variable is not driving anything and will be
                     simplified
  3 Info LBR-30      Promoting a setup arc to recovery.
                     Setup arcs to asynchronous input pins are not
                     supported
  3 Info LBR-31      Promoting a hold arc to removal.
                     Hold arcs to asynchronous input pins are not
                     supported
  1 Info LBR-54      Library has missing unit.
                     Current library has missing unit.
```

- The following example requests to print all error messages since this run was started, but no error messages were found:

```
rc:/> report messages -all -error
```

report net_cap_calculation

`report net_cap_calculation` *`net`* `[>` *`file`*`]`

Reports the capacitance values for the different pins or ports that are connected to the specified net.

- For a pin, the capacitance value is the capacitance of the libcell pin (obtained from the `.lib`).
- For a port, the capacitance value is any capacitance annotated on the port (sum of the `external_pin_cap` and `external_wire_cap` attributes).
- The net capacitance is computed from the wire-load model available in the library. This is based on the `wireload_mode` attribute (`top`, `segmented`, or `enclosed`).
- The final capacitance is the sum of the net capacitance and the pin and port capacitance values to which the net is connected.

The columns "Terms from .lib" and "Cap from .lib" in the report will be populated if the wire-load model in the `.lib` appears as a table. Otherwise, it will be empty.

This command is not supported on those nets that have the `physical_cap` attribute set on them. Also, when you are in PLE mode, only the final capacitance is shown (no computation).

Options and Arguments

`file`	Redirects the report to the specified file.
`net`	Specifies the net name for which the report should be generated.

Related Information

Related command:	report net_res_calculation on page 395

report net_delay_calculation

```
report net_delay_calculation [-driver_pin {port|pin}...]
     [-load_pin {port|pin}...] [> file]
```

Reports the net delay, in picoseconds, between the specified driver and load pins. Both the driver and load pins should be on the same net. The delay computed would depend upon the tree type used (`tree_type` attribute): best case, worst case, or balanced tree.

Options and Arguments

`-driver_pin {`*`port`*`|`*`pin`*`}`	Specifies the starting port or pin on which to obtain the net delay.
`-load_pin {`*`port`*`|`*`pin`*`}`	Specifies the ending port or pin on which to obtain the net delay.
`file`	Redirects the report to the specified file.

Example

- The following command provides a report based on the in1[0] driver pin:

```
rc:/designs/areid/ports_in> report_net_delay_calculation -driver_pin in1[0]
============================================================
...
Operating conditions:   slow (balanced_tree)
Wireload mode:          segmented
============================================================

Formula: (Wres/f) * (Pcap + Wcap/f)

From      To        Wire res  Wire cap Fanout Pin cap of  Total pin cap of   Net
pin       pin        of net    of net of net   to pin       all to pins    delay

-------------------------------------------------------------------------------
in1[0]    inst1/g43/A  0.000      7.2    1      5.3            5.3        0.0
```

Related Information

Affected by this attribute: `tree_type`

report net_res_calculation

`report net_res_calculation` *`net`* `[>` *`file`*`]`

Reports the total wire resistance of the specified net. The total wire resistance is the sum of the individual net segment resistance (obtained from the wire-load model) and the external wire resistance (annotated on any port and obtained from the `external_wire_res` attribute) that is connected to the net.

- For a port, the resistance value is any resistance annotated on the port (the `external_wire_cap` attributes).
- The net resistance is computed from the wire-load model available in the library. This is based on the `wireload_mode` attribute (`top`, `segmented`, or `enclosed`).
- The final resistance is the sum of the net resistance and the pin and port resistance values to which the net is connected.

The columns "Terms from .lib" and "Cap from .lib" in the report will be populated if the wire-load model in the `.lib` appears as a table. Otherwise, it will be empty.

This command is not supported on those nets that have the `physical_res` attribute set on them. Also, when you are in PLE mode, only the final resistance is shown (no computation).

Options and Arguments

`file`	Redirects the report to the specified file.
`net`	Specifies the net name for which the report should be generated.

Related Information

Related command:	report net_cap_calculation on page 393

report nets

```
report nets [-hierarchical] [-pin pin...]
     [-minfanout integer] [-maxfanout integer]
     [net | instance]... [-sort string]
     [-cap_worst integer] [> file]
```

Generates a report on the nets of the current design. The report gives information for the top-level nets in the design. You can specify pin names, nets, instances, maximum and minimum fanout threshold values, nets, and instances. Control the data printed out using the `-minfanout` and `-maxfanout` options for nets that have fanout between these values.Nets that are followed by the "@" symbol in parenthesis indicate SPEF annotation.

Options and Arguments

`-cap_worst` *`integer`*	Specifies the number of worst capacitance nets that are to be reported.
`file`	Specifies the name of the file to which to write the report.
`-hierarchical`	Reports all the nets in the design hierarchy.
`-maxfanout`	Specifies an *`integer`* value and reports nets whose fanouts are below the given threshold value.
`-minfanout`	Specifies an *`integer`* value and reports nets whose fanouts are above the given threshold value.
`net``|`*`instance`*	Reports information on the specified nets or nets belonging to the instance.
`-pin` *`pin`*	Specifies a list of pin names and reports the nets connected to the pins.
`-sort`	Specifies the field name on which to sort. Valid field names are `load`, `resistance`, or `capacitance`.

Examples

- The following example reports the five worst capacitance nets in the top level:

```
rc:\> report nets -cap_worst 5
============================================================
  Generated by:           RTL Compiler (RC) Version
  Generated on:           Date
  Module:                 m1
  Technology library:     slow 1.5
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          segmented
============================================================
                          Wire      Wire   Wireload
  Net   Loads   Drivers Cap(fF) Res(k-ohm)    Model
-----------------------------------------------------
ai          2         3    14.5      0.000
n_135       2         2    10.4      0.000
n_0         2         1     6.7      0.000
ao[5]       1         2     6.7      0.000
bi          1         1     3.6      0.000
============================================================
```

- The following example sorts the nets in the top level by the number of loads:

  ```
  rc:\> report nets -sort load
  ============================================================
  ...
  ============================================================
                            Wire      Wire   Wireload
    Net   Loads   Drivers Cap(fF) Res(k-ohm)    Model
  -----------------------------------------------------
  n_1         8         1
  clk         3         1     0.0      0.000
  n_135       2         2    10.4      0.000
  n_0         2         1     6.7      0.000
  ai          2         3    14.5      0.000
  ......
  ============================================================
  ```

- The following example reports all the nets in the top level of the current design whose fanout is less than 2.

  ```
  rc:\> report net -maxfanout 2
  ==============================================================
  ...
  ==============================================================
                                    Wire      Wire     Wireload
        Net       Loads   Drivers Cap(fF) Res(k-ohm)     Model
  ---------------------------------------------------------------
  in1a[0]           1         1     0.4      0.000    AL_SMALL
  in1a[1]           1         1     0.4      0.000    AL_SMALL
  in1b[0]           1         1     0.4      0.000    AL_SMALL
  in1b[1]           1         1     0.4      0.000    AL_SMALL
  out2a[0]          1         1     0.4      0.000    AL_SMALL
  out2a[1]          1         1     0.4      0.000    AL_SMALL
  out2b[0]          1         1     0.4      0.000    AL_SMALL
  out2b[1]          1         1     0.4      0.000    AL_SMALL
  ```

```
out3a[0]           1         1     0.0      0.000    AL_SMALL
out3a[1]           1         1     0.0      0.000    AL_SMALL
out3b[0]           1         1     0.0      0.000    AL_SMALL
out3b[1]           1         1     0.0      0.000    AL_SMALL
out4a[0]           1         1     0.0      0.000    AL_SMALL
out4a[1]           1         1     0.0      0.000    AL_SMALL
out4b[0]           1         1     0.0      0.000    AL_SMALL
out4b[1]           1         1     0.0      0.000    AL_SMALL
================================================================
```

You can specify an instance name to the above example and get information on the nets associated with the instance(s) whose fanout is less than 2.

- The following example provides specific information on the `n_0 ai` net:

```
rc:\> report net n_0 ai
============================================================
...
============================================================
     Total   Slew Slew
Net Cap(fF)  Rise Fall Driver(s)   Load(s)
--------------------------------------------
n_0    20.3   0.0  0.0  g24/Y    g23/A
                                 g22/A
                                 bx_reg3/CK
ai     12.0   0.0  0.0  ai       g25/A
                                 bx_reg2/D
============================================================
```

- The following example reports the nets associated with the `g22/A bx_reg2/D` pin:

```
rc:/> report net -pin "g22/A bx_reg2/D"
============================================================
...
============================================================
     Total   Slew Slew
Net Cap(fF)  Rise Fall Driver(s)   Load(s)
------------------------------------------------------------
n_0    20.3   0.0  0.0  g24/Y    g23/A
                                 g22/A
                                 bx_reg3/CK
ai     12.0   0.0  0.0  ai       g25/A
                                 bx_reg2/D
============================================================
```

- The following example shows that the `address` nets are SPEF annotated while the `accum` nets are not:

```
============================================================
...
============================================================
address[0] (@)       1         1     1.0      0.000
address[1] (@)       1         1     0.8      0.000
accum[0]             1         1     2.1      0.000
accum[1]             1         1     8.5      0.000
```

report operand_isolation

```
report operand_isolation
     [-oi_instance instance...] [> file]
```

Reports operand-isolation information for the design. The information depends on whether the operand-isolation logic has been committed or not.

Note: This command only reports operand-isolation logic inserted by the RC-LP engine, provided that you do not remove the subdesigns corresponding to the operand-isolation logic. It does not report operand-isolation logic inserted by third-party tools.

Options and Arguments

`-oi_instance` *`instance`*

Reports detailed information for the specified operand-isolation instances. Information includes the name of the isolated instance, the list of control inputs and their associated switching activities, the list of data inputs that have been isolated, the list of output pins of the operand-isolation instance.

Note: If the isolated instance name is `(flattened)`, the datapath instance was flattened during optimization.

`file` Specifies the name of the file to which to write the report.

Examples

- The following command gives a summary after operand-isolation logic has been inserted.

```
rc:/> report operand_isolation
============================================================
  Generated by:          RTL Compiler-D (RC) version
  ....
============================================================

Operand Isolation Instances
---------------------------
Module  Operand Isolation Instance  Width  Isolated Instance  Control Inputs
-----------------------------------------------------------------------------
test    RC_OI_HIER_INST             8      add_13_21          en1, en2
        RC_OI_HIER_INST6            8      add_13_21          en1, en2
-----------------------------------------------------------------------------
Total   2
=============================================================================
```

where

- ❑ *Width* is the width of the data_bus input of the operand-isolation instance
- ❑ *Isolated instance* is the datapath instance whose input is isolated.
- ❑ *Control inputs* lists the control signals that are inputs to the operand-isolation instance

- The following command generates detailed operand-isolation instance information for the specified operand-isolation instance:

```
rc:/> report operand_isolation -oi_instance RC_OI_HIER_INST
============================================================
  Generated by:          RTL Compiler-D (RC) version
...
============================================================

Operand Isolation Instance : RC_OI_HIER_INST
--------------------------
  Module:                test (test)
  Isolated Instance:     add_13_21 (test/add_13_21)
  Control Inputs:
    RC_OI_CTRL_PORT      =       en1 (/designs/test/ports_in/en1)
                                 TCF = (0.50000, 0.020000/ns)
    RC_OI_CTRL_PORT_1    =       en2 (/designs/test/ports_in/en2)
                                 TCF = (0.50000, 0.020000/ns)
  Data Inputs:
    RC_OI_DATA_PORT      =       in1[7] ({/designs/test/ports_in/in1[7]})
                                 in1[6] ({/designs/test/ports_in/in1[6]})
                                 in1[5] ({/designs/test/ports_in/in1[5]})
                                 in1[4] ({/designs/test/ports_in/in1[4]})
                                 in1[3] ({/designs/test/ports_in/in1[3]})
                                 in1[2] ({/designs/test/ports_in/in1[2]})
                                 in1[1] ({/designs/test/ports_in/in1[1]})
                                 in1[0] ({/designs/test/ports_in/in1[0]})
  Outputs:
    RC_OI_OUT_PORT       =       n_79 (/designs/test/i..13_21/pins_in/A[7])
                                 n_78 (/designs/test/i..13_21/pins_in/A[6])
                                 n_77 (/designs/test/i..13_21/pins_in/A[5])
                                 n_76 (/designs/test/i..13_21/pins_in/A[4])
                                 n_75 (/designs/test/i..13_21/pins_in/A[3])
                                 n_74 (/designs/test/i..13_21/pins_in/A[2])
                                 n_73 (/designs/test/i..13_21/pins_in/A[1])
                                 n_72 (/designs/test/i..13_21/pins_in/A[0])
```

Related Information

Reporting Operand Isolation Information in *Low Power in Encounter RTL Compiler*
.

Affected by this command:	synthesize on page 294
Affected by these attributes:	lp_operand_isolation_prefix

report ple

```
report ple [design]... [> file]
```

Returns the physical layout estimation information for the specified design. The command reports information like aspect ratio, shrink factor, site size, layer names, direction of layers, capacitance, resistance, area, and the source used to extract the physical information.

Options and Arguments

design	Specifies the design name on which to report. If no design is specified, the current design is loaded.
file	Specifies the name of the file to which to write the report.

Example

- The following example reports the ple information for the current design:

```
rc:/> report ple
```

```
============================================================
  Generated by:           Encounter(r) RTL Compiler
  Interconnect mode:      ple
...
============================================================

Aspect ratio  : 1.06
Shrink factor : 0.90
Site size     : 2.52 um (from lef [tech+cell])

                   Capacitance
Layer               / Length        Data source:
Name    Direction  (pF/micron)      lef_library
--------------------------------
METAL1      H         0.000530
METAL2      V         0.000496
...
                    Resistance
Layer                / Length        Data source:
Name    Direction  (ohm/micron)      lef_library
---------------------------------
METAL1      H          1.203704
METAL2      V          0.639683
...
                        Area
Layer                  / Length      Data source:
Name    Direction      (micron)      lef_library
---------------------------------
METAL1      H          0.108000
METAL2      V          0.126000
...
```

report port

```
report port [-delay] [-driver] [-load] port... [> file]
```

Generates reports on the ports of the current design. By default, the report gives information on port direction, external delays, exception objects and their types, driver, slew, fanout load, pin capacitance and wire capacitance for the ports. You can also specify the port names on which the report is to be generated and control the data printed using the `-delay`, `-driver` and `-load` options.

Options and Arguments

`-delay`	Reports external delay information (rise and fall delay and the external delay object)
`-driver`	Reports the external driver name and the slew (rise and fall) values of the ports.
`file`	Specifies the name of the file to which to write the report.
`-load`	Reports the external fanout load and the pin and wire capacitance values of the ports.
`port`	Specifies the port for which to generate the report.

Example

The following example reports the external delay information on the `ck1`, `e_out[6]`, and `ena` ports:

```
rc:/> report port -delay -driver -load ck1 e_out[6] ena
External Delays & Exceptions
----------------------------
                        Rise   Fall  Ext Delay      Exception
   Port     Dir  Clock  Delay  Delay   Object      Object/Type
-------------------------------------------------------------------
 ck1       in     CLK1  700.0  700.0  in_del_1                N/A
                  CLK2  500.0  500.0  in_del_2

 e_out[6]  out    CLK1  200.0  200.0  ou_del_1  del_1 (path_delay)
                  CLK2  300.0  300.0  ou_del_2
                        300.0  300.0  outrxt1

 ena       inout  CLK1  700.0  700.0  in_del_1                N/A
                  CLK2  500.0  500.0  in_del_2

 ena       inout  CLK1  200.0  200.0  ou_del_1  del_1 (path_delay)
                  CLK2  300.0  300.0  ou_del_2
                        300.0  300.0  outrxt1
```

report power

```
report power
     { -rtl_cross_reference [-detail]
       [-flat [-nworst number]] [-sort mode]
       [design | instance]...
       [-mode mode] [-tcf_summary]
     | [-hier | -flat [-nworst number]] [-depth number]
       [-sort mode] [design | instance | net]...
       [-mode mode] [-tcf_summary]
     | -clock_tree [clock]...
       -width float -height float }
     [> file]
```

Reports the power consumed. The information returned depends on your current *position* in the design hierarchy and on the specified objects. If no objects are specified, the report is given for the design or instance at the current *position* in the design hierarchy .

With the `-rtl_cross_reference` option specified, the power consumed by the instances is cross-referenced to the corresponding line in the RTL files. Set the `hdl_track_filename_row_col` attribute to `true` before elaboration to enable filename, column, and line number tracking.

Note: Nets connected to primary inputs and outputs are only reported at the top-level of an instance-based power report.

Options and Arguments

`clock`	Specifies the name of a clock for which you want to estimate the clock tree power. If no clock is specified, the power is estimated for all clocks in the design.
`-clock_tree`	Estimates the power of the clock tree. You can use this option with generic and mapped netlists.
`-depth` *`number`*	Specifies the number of hierarchy levels to descend in the report. If an instance is specified, the depth starts from the position of that instance in the design hierarchy. Use a non-negative integer. **Note:** This option applies to instance-based power reports, but is not supported with the `-rtl_cross_reference` option. *Default*: `infinite` (all levels of the hierarchy)

`design`	Specifies the design for which you want the power to be reported. Specify the path name to the design. If your current *position* in the design hierarchy is at the design level and you have only one design loaded, the report is by default given for the design.
`-detail`	Adds an abbreviated version of the RTL line and a list of the instances that correspond to that RTL line. In this report, the dynamic power is replaced with the internal power and net power (the two components of dynamic power). The detailed report also returns the power information for the primary inputs. **Note:** This option only applies if you specified the `-rtl_cross_reference` option.
`file`	Specifies the name of the file to which to write the report.
`-flat`	Reports the power of leaf instances (combinational and sequential instances) starting from the *current* position in the hierarchy. ■ If you perform a gate-level power analysis, the number of hierarchical levels that are expanded depends on the setting of the `-depth` option. ■ If you perform RTL power analysis using the `-rtl_cross_reference` option, power information for all modules in the current hierarchy is shown. **Note:** This option only applies to instance-based power reports.
`-full_instance_names`	Reports the full path names of the instances.
`-height` *`float`*	Specifies the estimated chip height (in microns). **Note:** This option can only be specified with the `-clock_tree` option and is optional if a DEF file was read in.
`-hier`	Reports the power of hierarchical instances. The number of hierarchical levels shown depends on the setting of the `-depth` option. If you specify neither the `-flat` or `-hier` option, the RC-LP engine uses the -hier option by default. **Note:** This option applies to instance-based power reports, but is not supported with the `-rtl_cross_reference` option.

`instance`	Specifies the instance for which you want the power to be reported. Specify the path name to the instance.
`-mode` *`mode`*	Only prints the power report for the specified power mode. The specified mode must correspond to one of the power mode names defined with a `create_power_mode` command in the CPF file that was read in. If you have a multi-mode design and you omit this option, the report will apply to the current state of the design. **Note:** This option cannot be used with the `-clock_tree` option.
`net`	Specifies the net for which you want the power to be reported. Specify the path name to the net. **Note:** Net-based power reports are not supported with the `-rtl_cross_reference` option.
`-nworst` *`number`*	Prints only the top worst entries of a sorted report. This option applies only to instance-based reports, and can only be used with the `-flat` option.
`-rtl_cross_reference`	
	Cross-references the power consumed to the corresponding line in the RTL files. The report also returns the leakage power, dynamic power, and total power for the top-level design.
`-sort` *`mode`*	Indicates how to sort the report. The following modes are available for *instance*-based reports:

`dynamic`	Sorts by descending total dynamic power, which is the sum of the internal and net power. **Note:** This option is not supported with the `-rtl_cross_reference` option.
`file`	Sorts by RTL file and line number. **Note:** This option applies only to RTL power analysis and is the default for RTL power analysis.
`internal`	Sorts by descending internal power.

`leakage` (default)	Sorts by descending leakage power.
`net`	Sorts by descending net power.

The following modes are available for *net*-based reports:

`dynamic` (default)	Sorts by descending total dynamic power.
`load`	Sorts by descending capacitive load on the net.
`net`	Sorts by descending total switching power.
`prob`	Sorts by descending probability of the nets being high.
`rate`	Sorts by descending toggle rates of the nets.

Note: If a report is requested for nets and instances, but the specified sort mode applies only to one category, the other category will be sorted according to its default.

`-tcf_summary` Adds a summary to the power report listing the following:

- The number of primary inputs asserted in the design.
- The total number of primary inputs connected in the design.
- The number of sequential outputs asserted in the design.
- The total number of sequential outputs connected in the design.
- The total number of nets in the design.
- *Nets asserted* refer to nets with asserted switching activities (probability and toggle rate).
- *Asserted clock nets* refer to nets whose `lp_probability_type` or `lp_toggle_rate_type` attribute value is set to `clock`.
- *Constant nets* refer to nets whose driver is either a constant object `0` or `1`.

For net-based reports, an asterisk (*) is appended to each net that has user-asserted switching activities.

`-verbose`	Replaces the dynamic power column with the components of the dynamic power—the internal power and net power.
`-width` *`float`*	Specifies the estimated chip width (in microns). **Note:** This option can only be specified with the `-clock_tree` option and is optional if a DEF file was read in.

Examples

- The following command requests a basic RTL power analysis of design `mult_bit_muxed_add`.

```
rc:/> build_rtl_power_models -clean_up_netlist
rc:/> report power -rtl_cross_reference
============================================================
  ...
  Technology library:     xxx yy
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
============================================================

                          Leakage   Dynamic    Total
       Design            Power(nW) Power(nW) Power(nW)
-------------------------------------------------------
mult_bit_muxed_add          71.146  1578.273  1649.419

                          Leakage   Dynamic    Total
        File         Row Power(nW) Power(nW) Power(nW)
-------------------------------------------------------
mult_bit_muxed_add.v  8     35.573   665.277   700.850
mult_bit_muxed_add.v  9     5.573   675.858   711.431
```

- The following command shows RTL power analysis for all levels of the hierarchy:

```
rc:/> report power -rtl -flat
============================================================
  ...
  Technology library:     xxx yy
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
============================================================

                          Leakage   Dynamic    Total
       Design            Power(nW) Power(nW) Power(nW)
-------------------------------------------------------
mult_bit_muxed_add          71.146  1578.273  1649.419

                          Leakage   Dynamic    Total
        File         Row Power(nW) Power(nW) Power(nW)
-------------------------------------------------------
muxed_add.v           8     28.308   519.317   547.625
muxed_add.v           9     21.419   403.153   424.572
muxed_add.v           12    21.419   418.665   440.084
```

- The following command requests a detailed RTL power analysis report.

```
rc:/> report power -rtl -detail -flat
============================================================
...
  Module:                 mult_bit_muxed_add
...
============================================================

                                                    Leakage   Internal    Net
       Design                                      Power(nW) Power(nW) Power(nW)
--------------------------------------------------------------------------------
mult_bit_muxed_add                                    71.146  1069.140   509.134
--------------------------------------------------------------------------------

                                                    Leakage   Internal    Net
    Primary Input                                  Power(nW) Power(nW) Power(nW)
--------------------------------------------------------------------------------
a[1]                                                   0.000     0.000    19.428
a[0]                                                   0.000     0.000    19.428
b[1]                                                   0.000     0.000    19.428
b[0]                                                   0.000     0.000    19.428
c[1]                                                   0.000     0.000    19.428
c[0]                                                   0.000     0.000    19.428
d[1]                                                   0.000     0.000    19.428
d[0]                                                   0.000     0.000    19.428
s                                                      0.000     0.000    81.713
---------------------------------------------------------------------------------

                                                    Leakage   Internal    Net
        File         Row        RTL Line Instances Power(nW) Power(nW) Power(nW)
--------------------------------------------------------------------------------
muxed_add.v          8  {if (s) begin}          g1    28.308   397.891   121.426
                                                g1
                                                g1
                                                g1
--------------------------------------------------------------------------------
muxed_add.v          9   {y = a + c;}           g1    21.419   330.297    72.856
                                                g1
--------------------------------------------------------------------------------
muxed_add.v          12  {y = b + d;}           g1    21.419   340.952    77.713
                                                g1
--------------------------------------------------------------------------------
```

- The following command requests a detailed RTL power analysis report for instance `add_9_15`.

```
rc:/> report power -rtl -detail [find . -inst add_9_15]
============================================================
  ...
  Module:                 mult_bit_muxed_add
...
============================================================


                                                   Leakage   Internal    Net
       Design                                     Power(nW) Power(nW) Power(nW)
--------------------------------------------------------------------------------
mult_bit_muxed_add                                   71.146  1069.140   509.134
--------------------------------------------------------------------------------

                                                   Leakage   Internal    Net
    Primary Input                                 Power(nW) Power(nW) Power(nW)
--------------------------------------------------------------------------------
a[1]                                                  0.000     0.000    19.428
a[0]                                                  0.000     0.000    19.428
b[1]                                                  0.000     0.000    19.428
b[0]                                                  0.000     0.000    19.428
c[1]                                                  0.000     0.000    19.428
c[0]                                                  0.000     0.000    19.428
d[1]                                                  0.000     0.000    19.428
d[0]                                                  0.000     0.000    19.428
s                                                     0.000     0.000    81.713
--------------------------------------------------------------------------

                                            Leakage   Internal    Net
       File        Row  RTL Line  Instances Power(nW) Power(nW) Power(nW)
--------------------------------------------------------------------------
muxed_add.v        9   y = a + c; add_9_15     21.419   330.297    72.856
                                  add_9_15
--------------------------------------------------------------------------
```

Note: The report shows two `add_9_15` instances because RTL Compiler found two instances with that name.

- The following example first descends in the design hierarchy down to instance `add_9_15`, then requests a detailed RTL power analysis. The output is slightly different from the previous example.

```
rc:/> cd /designs/mult_bit*/instances_hier/ma0/instances_hier/add_9_15

rc:/designs/mult_bit_muxed_add/instances_hier/ma0/instances_hier/add_9_15>
report power -rtl -detail
============================================================
  ...
============================================================

                                            Leakage   Internal    Net
       File        Row  RTL Line  Instances Power(nW) Power(nW) Power(nW)
--------------------------------------------------------------------------
muxed_add.v        9   y = a + c; add_9_15     10.709   170.476    38.856
--------------------------------------------------------------------------
```

- The following command shows the clock tree power estimation for clock `clk`.

```
rc:/> report power -clock_tree iCLK1 -width 5 -height 5
============================================================
  Generated by:           Encounter(r) RTL Compiler version
  Generated on:           date
  Module:                 test
  Technology libraries:   typical 1.3
  Operating conditions:   typical (balanced_tree)
  Wireload mode:          segmented
  Area mode:              timing library
============================================================

Clock Power Estimation Summary for clock 'iCLK1'
================================================

-----------------------------------------------------
Estimate   Leakage (nW)   Dynamic (nW)   Total (nW)
-----------------------------------------------------
Max               0.007     147560.871     147560.878
Min               0.007      42160.249      42160.251
Typical           0.007      67941.241      67941.244

Leaf CGIC Cells                       4
Leaf Clock Buffers                    0
Total Clock Buffers                   2

Estimation Parameters
=====================

Clock Buffers Used: BUFX12 BUFX16 BUFX2
                    BUFX20 BUFX3 BUFX4
                    BUFX6 BUFX8 CLKBUFX12
                    CLKBUFX16 CLKBUFX2 CLKBUFX20
                    CLKBUFX3 CLKBUFX4 CLKBUFX6
                    CLKBUFX8 DLY1X1 DLY1X4
                    DLY2X1 DLY2X4 DLY3X1
                    DLY3X4 DLY4X1 DLY4X4

Max flops driven by one leaf buffer: 3
Die width: 5.0 um
Die height: 5.0 um
```

- The following command reports the power (after synthesis) at the current level in the hierarchy, which is the design.With the `-hier` option specified, the report lists the design and its hierarchical instances.

```
rc:/designs/mult_bit_muxed_add> report power -hier
============================================================
  ...
============================================================

                          Leakage   Dynamic    Total
     Instance      Cells Power(nW) Power(nW) Power(nW)
-------------------------------------------------------
mult_bit_muxed_add     6    71.146  1578.273  1649.419
  ma0                  3    35.573   636.281   671.854
  ma1                  3    35.573   638.728   674.301
```

- The following command reports the power (after synthesis) at the current level in the hierarchy, which is the design. With the `-verbose` option specified, the report shows in addition the components of the dynamic power.

```
rc:/designs/mult_bit_muxed_add> report power -verbose
============================================================
  ...
============================================================
                        Leakage   Internal    Net      Dynamic        Total
                                                              (Int+Net)(Leak+Dyn)
     Instance     Cells Power(nW) Power(nW) Power(nW) Power(nW)      Power(nW)
-------------------------------------------------------------------------------
mult_bit_muxed_add    6    71.146  1069.140   509.134  1578.273      1649.419
  ma0                 3    35.573   533.055   103.226   636.281       671.854
  ma1                 3    35.573   536.085   102.643   638.728       674.301
```

- The following command reports the power (after synthesis) at the current level in the hierarchy, which is the design.With the `-flat` option specified, the report lists leaf instances starting from the current position in the hierarchy.

```
rc:/designs/mult_bit_muxed_add> report power -flat
============================================================
  ...
============================================================

                Leakage   Dynamic    Total
Instance Cells Power(nW) Power(nW) Power(nW)
---------------------------------------------
ma0/g38           13.900   255.674   269.574
ma0/g39           13.900   253.342   267.242
ma1/g38           13.900   223.699   237.599
ma1/g39           13.900   253.350   267.250
ma0/g37            7.774   127.265   135.039
ma1/g37            7.774   161.679   169.453
```

- The following command reports the power (after synthesis) at the current level in the hierarchy, which is a subdesign.With the `-hier` option specified, the report lists the subdesign and its hierarchical instances. Because in this case there are no hierarchical instances, only the power for the subdesign is listed.

```
rc:/designs/mult_bit_muxed_add/instances_hier/ma0> report power -hier
============================================================
  ...
============================================================

                Leakage   Dynamic    Total
Instance Cells Power(nW) Power(nW) Power(nW)
---------------------------------------------
ma0          3    35.573   636.281   671.854
```

- The following command reports the power for an instance and sorts the report according to descending net power.

```
rc:/designs/mult_bit_muxed_add> report power -flat instances_hier/ma0 \
-sort net
============================================================
  ...
============================================================

               Leakage   Dynamic    Total
Instance Cells Power(nW) Power(nW) Power(nW)
---------------------------------------------
ma0/g39          13.900   255.674   269.574
ma0/g38          13.900   253.342   267.242
ma0/g37           7.774   127.265   135.039
```

- The following command reports the power for an instance and a net.

```
rc:/designs/mult_bit_muxed_add> report power nets/a[1] ma0 -flat
============================================================
  ...
============================================================

                Leakage   Dynamic    Total
Instance Cells Power(nW) Power(nW) Power(nW)
---------------------------------------------
ma0/g39          13.900   255.674   269.574
ma0/g38          13.900   253.342   267.242
ma0/g37           7.774   127.265   135.039

     Net         Net            Toggle
(asserted *) Power (nW) Prob. Rate (/ns) Cap. (nF)
---------------------------------------------------
a[1]             26.827 0.500      0.020     2.300
```

- The following design has five power domains and three power modes. The following commands show the power report for two of the modes after mapping. In this case the report has an additional column Domain (voltage) which shows for each instance to which power domain it belongs and what the voltage is of the domain in the reported mode. Note that when the second `report power` command is given, the tool adjusts the wireload models for the specified mode before reporting power.

```
rc:/designs/counter> report power -mode PMdefault
============================================================
 ...
 Module:         counter
 Library domain:     umc_0p8v
  Domain index:     0
  Technology libraries: ...
  Operating conditions: _nominal_ (balanced_tree)
 Library domain:     umc_1v
  Domain index:     1
  ...)
 Library domain:     umc_1p2v
  Domain index:     2
  ...
 Wireload mode:     enclosed
 Area mode:       timing library
 Power mode:       PMdefault
============================================================

                     Domain           Leakage   Dynamic    Total
   Instance        (Voltage)   Cells Power(nW) Power(nW) Power(nW)
-------------------------------------------------------------------
counter          PDcore(0.81v)   142     7.076  9192.022  9199.099
  monitor_power   PDmon(0.81v)    84     3.720  4050.236  4053.956
  adder_counter   PDadd(0.81v)    25     1.215  1573.961  1575.175
  bcd_counter     PDbcd(0.81v)    12     1.088  1697.572  1698.659
  binary_counter  PDbin(0.81v)    18     1.012  1565.877  1566.888
rc:/designs/counter> report power -mode PMmid
============================================================
 ...
 Power mode:       PMmid
============================================================


    Applying wireload models.
Info   : Changing wireload model of a design/subdesign. [TIM-92]
    : Changing wireload model of design ’counter’ from <none> to cmos065.
    : The change of wireload model will likely change the design’s timing
slightly.
...
    Computing net loads.

                 Library        Leakage   Dynamic    Total
   Instance       Domain Cells Power(nW) Power(nW) Power(nW)
-------------------------------------------------------------
counter           umc_1v   142    11.023 33056.713 33067.735
  monitor_power   umc_1v    84     5.805 14774.642 14780.447
  adder_counter   umc_1v    25     1.884  4185.202  4187.086
  bcd_counter     umc_1v    12     1.697  6138.779  6140.477
  binary_counter  umc_1v    18     1.569  6542.145  6543.714
```

Related Information

RTL Power Analysis in *Low Power in Encounter RTL Compiler*

Reporting Clock Tree Power in *Low Power in Encounter RTL Compiler*

Reporting on All Power Components in *Low Power in Encounter RTL Compiler*

Affected by these commands:	build_rtl_power_models on page 699
	synthesize on page 294
Affected by these attributes:	cell_leakage_power
	leakage_power_scale_in_nW
	lp_asserted_probability
	lp_asserted_toggle_rate
	lp_power_unit
Related attributes	lp_clock_tree_buffers
	lp_clock_tree_leaf_max_fanout
	lp_computed_probability
	lp_computed_toggle_rate
	lp_leakage_power

report power_domain

```
report power_domain
     [-detail] [-mode]
     [-power_nets] [-qor]
     [> file]
```

Reports power domain related information.

Note: An asterisk (*) identifies the default power domain.

Without any options specified, a summary report is given which shows for each power domain

- The name of the shutoff signal
- The polarity of the shutoff signal
- The operating voltage in the default power mode

Options and Arguments

`-detail`	Prints the summary information and the information you would get by specifying the `-mode`, `-power_nets` and `-qor` options.
`file`	Specifies the name of the file to which the report is to be written.
`-mode`	Prints the operating voltage of each power domain in each power mode.
`-power_nets`	Prints for each power domain the following information for the power and ground nets: ■ The external power or ground net ■ The internal power or ground net ■ The power supply to which the net must be connected ■ The name of the library that describes the power supply

`-qor`	Prints the following information for each power domain: ■ The cell area ■ The percentage of nets with user-asserted switching activities ■ The dynamic power consumption ■ The leakage power consumption **Note:** The results are given for the current power mode and are affected by the setting of the `lp_power_unit` attribute.

Example

- The following example shows the basic report (with no options specified).

```
rc:/designs/top> report power_domain
Summary
=======
============================================================
                Shut-off            signal
         -------------------------------------
 Name             Name          Active level   Voltage(V)
============================================================
LD1                -                 -             1.2
PD1(*)             -                 -             0.8
PD2      pm_inst/pse_enable[0]    active_high      0.8
PD3      pm_inst/pse_enable[1]    active_high      0.8
PD4      pm_inst/pse_enable[2]    active_high      0.8
------------------------------------------------------------
5
```

- The following example shows the power mode information. The default power mode is marked with an asterisk.

```
rc:/designs/top> report power_domain -mode
Power Modes
===========
============================================
                   Power Modes
--------------------------------------------
Power Domain   PM1        PM2    PM3   PM4
============================================
LD1            1.2        1.2    1.2   1.2
PD1(*)         0.8        0.8    0.8   0.8
PD2            0.8        OFF    OFF   OFF
PD3            0.8        0.8    OFF   OFF
PD4            0.8        0.8    0.8   OFF
--------------------------------------------
5
```

- The following example shows the power net information. The report indicates that no information was given for the ground nets.

```
rc:/designs/top> report power_domain -power_nets
```

```
Power Nets
==========
=========================================================================================
                 Power          Nets
               ------------------------
Power Domain   External     Internal   Rail Connection   Library
=========================================================================================
PD2             VDD_0.8      VDD2      VDDH           /libraries/library_domains/ld_1/lib2
PD3             VDD_0.8      VDD3      VDDH           /libraries/library_domains/ld_1/lib2
PD4             VDD_0.8      VDD4      VDDH           /libraries/library_domains/ld_1/lib2
-----------------------------------------------------------------------------------------

Ground Nets
==========
=========================================================================================
                Ground          Nets
               ------------------------
Power Domain   External      Internal  Rail Connection   Library
=========================================================================================
-----------------------------------------------------------------------------------------
```

Related Information

Affected by these commands: read_cpf on page 760

report qor

`report qor [-levels_of_logic] [`*`design`*`]... [> `*`file`*`]`

Reports the critical path slack, total negative slack (TNS), number of gates on the critical path, and number of violating paths for each cost group. It also gives the instance count, total area (net and cell area), cell area, leakage power, dynamic power, runtime, and host name information.

If you perform physical synthesis and start with a floorplan, the report also contains the floorplan utilization in %. If you executed either the `synthesize -to_placed` or the `predict_qos` command, the report will also contain a `Silicon Virtual Prototype` section that lists the total and average net length in micron, and the routing congestion in %. Routing congestion is a measure of track overflow . A value greater than 5% in either direction gives an indication that the design will be difficult to route. The information is static information from the most recent Encounter® batch job.

Options and Arguments

`design`	Specifies the design name on which to report. If no design is specified, the report is given for the current design.
`file`	Specifies the name of the file to which the report is to be written.
`-levels_of_logic`	Prints the number of gates on the critical path per cost group.

Example

- The following example reports the QoR data for the current design.

```
rc:\> report qor

============================================================
  Generated by:           RTL Compiler version
  Generated on:           date
  Module:                 cscan
  Technology libraries:   tutorial 1.0
                          slow_hvt 1.1
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
============================================================

Timing
------
Cost   Critical        Violating
Group  Path Slack TNS      Paths
----------------------------------
I2C       1182.4   0          0
C2O       1987.2   0          0
.....
```

```
Instance Count
--------------
Leaf Instance Count            41
Sequential Instance Count      16
Combinational Instance Count   25
Hierarchical Instance Count     0

Area & Power
------------
Total Area                         106.445
Cell Area                          106.445
Leakage Power                      0.583 nW
Dynamic Power                      8714.675 nW
Total Power                        8715.259 nW

Max Fanout                         2 (out1[0])
Min Fanout                         1 (in2[3])
Average Fanout                     1.2
Terms to net ratio                 0.5
Terms to instance ratio            3.0
Runtime                            4.19 seconds
Hostname                           rcae030.cadence.com
```

- The following command requests to report the number of gates on the critical path per cost group data. This adds a `No of gates on Critical Path` column the table under `Timing`. The rest of the report does not change.

```
rc:\> report qor -levels_of_logic
```

```
============================================================
...
============================================================

Timing
------
Cost   Critical      No of gates on  Violating
Group  Path Slack TNS  Critical Path    Paths
-------------------------------------------------
I2C        1182.4   0             2         0
C2O        1987.2   0             1         0
.....
```

- The following example reports the QoR data for a Dynamic Voltage Frequency Scaling (DVFS) design (design with multiple power modes). In this case, an additional `Mode` column is added to the table under `Timing`. The rest of the report does not change.

```
Timing
------

       Cost  Critical Path  Violating
Mode  Group      Slack       Paths
-------------------------------------
m1   default      No paths
     I2C            -202.6         1
     C2O            -116.5         1
     C2C          No paths
     I2O          No paths
m2   default        -202.6         2
     I2C          No paths
     C2O          No paths
     C2C          No paths
...
```

- The following example reports the QoR data after you executed the `synthesize -to_placed` command.

```
rc:/> report qor
============================================================
  ...
  Module:                 fifo
  Technology libraries:   slow 1.3
                          physical_cells
  Operating conditions:   slow
  Interconnect mode:      ple1
  Area mode:              physical library
============================================================

Timing
------

  Cost    Critical            Violating
 Group   Path Slack    TNS      Paths
----------------------------------------
default    No paths        0
CLK1         -802.8   -47540         90
----------------------------------------
Total                 -47540         90

Instance Count
--------------

Leaf Instance Count             209
Sequential Instance Count        82
Combinational Instance Count    127
Hierarchical Instance Count       0

Area & Power
------------
Total Area                         6744.198
Cell Area                          5195.741
Floorplan Utilization              59.05%
Leakage Power                      0.134 nW
Dynamic Power                      1112757.869 nW
Total Power                        1112758.004 nW


Max Fanout                         82 (n_60)
Min Fanout                         1 (d)
Average Fanout                     3.4
Terms to net ratio                 4.6
Terms to instance ratio            5.0
Runtime                            7.66 seconds
Hostname                           rcae003

Silicon Virtual Prototype
-------------------------
Total Net Length                   7280.40 um
Average Net Length                 32.65 um
Routing Congestion                 H: 0.00% V: 0.00%
```

report scan_power

```
report scan_power [-clock float]
     [ -flop float
     | -atpg [-atpg_options string] [-capture]
       [-start_vector integer] [-end_vector integer]
     | -scan_vectors file [-capture]
       [-start_vector integer] [-end_vector integer] ]
     [-report_only_switching] [-library string] [> file]
```

Reports the estimated average power consumption or average switching activities of the design during test.

Note: To use this command you need to have the Encounter Test software installed and your operating system PATH environment variable must include the path to the Encounter Test software. For more information on the exact product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Options and Arguments

`-atpg`	Invokes Encounter Test to compute switching activities from the test vectors.
`-atpg_options` *string*	Specifies a string containing extra options to run ATPG-based analysis. **Note:** For more information on these options, refer to the `create_tests` command in the *Command Line Guide* (of the Encounter Test documentation).
`-capture`	Reports the average power consumed in scan capture mode. **Note:** This option must be specified with either the `-atpg` or `-scan_vectors` option.
`-clock` *float*	Specifies the frequency in MHz of the scan clock. *Default*: frequency of the first test clock object found
`-end_vector` *integer*	Specifies at which test vector to stop when computing the switching activities.
file	Specifies the file to which to redirect the report.

`-flop` *`float`*

Specifies the frequency in MHz of the flops. The frequency should not be more than the clock frequency.

Default: 50% of the clock frequency.

`-library` *`string`*

Specifies the list of Verilog structural library files. Specify the list in a quoted string.

Note: This option is only required when you invoke this command on a mapped netlist.

`-report_only_switching`

Reports the average scan-in, scan-out, and average overall switching activity (toggle rate).

When you combine this option with the `-capture` option, the command reports the average capture switching activity.

Note: This information is only reported if you either requested to create test vectors (using the `-atpg` option) or specified to use existing test vectors (using the `-scan_vectors` option). Otherwise, the average switching activity is computed based on 50% of the clock frequency.

`-scan_vectors` *`file`*

Specifies a file containing test vectors. The file must be specified in Encounter Test TBDpatt format. The command can read files that have been compressed with `gzip` (`.gz` extension).

Note: The test vectors are written in TBDpatt format using Encounter Test `report_vectors` command. The test vectors may be written using `format` type `vectors|node`; the `vectors` option produces more compacted output. Use the following options when writing the test vector file:

```
report_vectors format=vector compact=fill\
outputfile=fileName ....
```

For more information on the TBDpatt format, refer to *Test Pattern Data Reference for Encounter Test*.

`-start_vector` *`integer`*

Specifies from which test vector to start when computing the switching activities.

Default: 1

Examples

- The following three sets of commands are equivalent. They all specify two files to be used as Verilog simulation libraries.

  ```
  set simLibs "sim/tsmc13.v sim/tpz013g3.v"
  report scan_power  -atpg -library $simLibs
  ```

  ```
  set rootDir sim
  set verilogLibs "$rootDir/tsmc13.v $rootDir/tpz013g3.v"
  report scan_power -atpg -library $verilogLibs
  ```

  ```
  set rootDir sim
  report scan_power -atpg -library "${rootDir}/tsmc13.v ${rootDir}/tpz013g3.v"
  ```

- The following command requests to estimate the power consumed during scan test when a scan clock frequency of 20 MHz is applied.

  ```
  rc:/> report scan_power -clock 20
  Computing the scan power with the following toggle frequencies:
  ... set clocks @ 20 MHz
  ... set flops  @ 10.0 MHz (computed at 50% of clock frequency)
  ============================================================
    Generated by:           version
    Generated on:           date
    Module:                 cpu
    Technology library:     typical 1.3
    Operating conditions:   typical (balanced_tree)
    Wireload mode:          segmented
  ============================================================

                  Leakage   Dynamic    Total
  Instance Cells Power(nW) Power(nW) Power(nW)
  ---------------------------------------------
  cpu       1588     0.285 93261.187 93261.472
  ```

- The following command controls the type of latchfill used to fill the non-care bits (non-targeted faults) while generating the ATPG vectors. This results in a reduction of the scan power.

  ```
  report scan_power -atpg -atpg_options "latchfill=repeat"
  ...
  Computing the scan power with the following toggle frequencies:
  ... set clocks @ 20.0 MHz
  ... set flops  @ 3.8 MHz (computed using scan power test vectors)
  ============================================================
    Generated by:           version
    Generated on:           date
    Module:                 cpu
    Technology library:     typical 1.3
    Operating conditions:   typical (balanced_tree)
    Wireload mode:          segmented
  ============================================================

                  Leakage   Dynamic    Total
  Instance Cells Power(nW) Power(nW) Power(nW)
  ---------------------------------------------
  cpu       1588     0.285 52384.169 52384.455
  ```

- The following command reports the average switching activity in scan-shift mode.

```
rc:/> report scan_power -atpg -report_only_switching
...
Switching activity report (computed using scan power test vectors)
    Average switching activity:          0.47
    Average scan-in switching activity:  0.46
    Average scan-out switching activity: 0.48
```

- The following command reports the average switching activity in capture mode.

```
rc:/> report scan_power -atpg -report_only_switching -capture
...
Switching activity report (computed using scan power test vectors)
    Average capture switching activity:  0.49
```

Related Information

Analyzing the Scan Power in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft test_mode on page 581
	define_dft test_clock on page 577

report sequential

```
report sequential [-instance_hier instance] [-hier]
     [subdesign |design] [> file]
```

Generates a report on the sequential elements of the current design. The report provides the sequential element name, row, column, filename information, and the sequential element type (flip-flop (async set/rest, sync set/reset, sync enable), latch, or timing model).

Note: Set the `hdl_track_filename_row_col` attribute to `true` before using the `elaborate` command to track the filename, row and column information.

Options and Arguments

`file`	Specifies the name of the file to which to write the report.
`-hier`	Reports all the flops in the design.
`-instance_hier` *`instance`*	Specifies the hierarchical instance name to report flops.
`subdesign`|*`design`*	Specifies the design or subdesign name to report flops.

Examples

- The following example reports all the sequential elements in the top level design:

```
rc:/> report sequential
============================================================
  Generated by:           Version
  Generated on:           Date
  Module:                 test
  Technology library:     slow 1.5
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          segmented
============================================================
                                 Instantiated/
 Register       File  Row Column  Inferred              Type
-----------------------------------------------------------------------------

sync_rst_reg    all.v  12   16   inferred      flip-flop synchronous reset
async_rst_reg   all.v  21   27   inferred      flip-flop asynchronous reset
sync_set_reg    all.v  30   37   inferred      flip-flop asynchronous set
no_rst_reg      all.v  39   45   inferred      flip-flop
sync_preset_reg all.v  44   58   inferred      flip-flop synchronous set
q_reg           all.v  6    12   inferred      latch
```

- The following example reports all the sequential elements in the design:

```
rc:/> report sequential -hier
report sequential: prints a sequential instance report.
============================================================
....
============================================================
                                                 Instantiated/
 Register                File    Row Column   Inferred          Type
-----------------------------------------------------------------------------
m2/m3/m4/m5/o_m5_0_reg_1 hier.v  25   22   instantiated  flip-flop synchronous
                                                                      enable
m2/m3/m4/m5/o_m5_0_reg_2 hier.v  27   22   instantiated  flip-flop synchronous
                                                                      enable
m2/m3/m4/m5/o_m5_1_reg_0 hier.v  30   22   instantiated  flip-flop synchronous
                                                                      enable
m2/m3/m4/m5/o_m5_0_reg_0 hier.v  23   22   instantiated  flip-flop synchronous
                                                                      enable
...................
...................
m2/o_m2_clk1_0_reg_2   hier.v 174   27   instantiated  flip-flop synchronous
                                                                      enable
```

- The following example reports a timing model:

```
rc:> report sequential
=============================================================================
Generated by:           RTL Compiler (RC) Version
Generated on:           Date
Module:                 m1
Technology library:     slow 1.0
Operating conditions:   slow (balanced_tree)
Wireload mode:          enclosed
=============================================================================
                                                           Instantiated/
      Register               File       Row Column    Inferred       Type
-----------------------------------------------------------------------------
clockgate/g2clatch_tlat timing_model.v  22   29   instantiated  timing_model
```

report slew_calculation

```
report slew_calculation  pin [-rise | -fall] [> file]
```

Reports how the slew of a cell driver pin is calculated from the look up table in the loaded technology library. The formula for calculating the delay is provided at the bottom of the report.

Options and Arguments

`file`	Redirects the report to the specified file.
`[-fall	-rise]`	Uses the falling or rising slew calculation on the driver pin.
	By default, all possible arcs are reported.
`pin`	Specifies the cell driver pin.

report spare_instances

```
report spare_instances
    [> file]
```

Important

This command is only available in RCQA mode.

Reports the number of spare instances. Before running this command, you must set the `spare_instance` attribute to denote an instance as a spare instance. The percentage value reported is:

(Number of spare instances/Total number of instances) * 100.

Options and Arguments

`file`	Specifies the name of the report file.

Examples

- The following command reports the number of spare instances to a file named `file.rpt`:

  ```
  rcqa:/> report spare_instances > file.rpt
  ```

report state_retention

```
report state_retention
     { [instance_list]
     | [-hierarchical] [-detail]
       [-power_gating_pin_driver {port|pin}...]
       [-power_domain power_domain-list]
     [-verbose] [> file]
```

Reports state_retention information for the design. The return value of the report corresponds to the number of state-retention registers found.

Options and Arguments

`-detail`	Requests a detailed state retention report.
`file`	Specifies the name of the file to which to write the report.
`-hierarchical`	Reports state retention registers hierarchically. If this option is omitted, reports all state retention registers at the current level of the hierarchy.
`instance_list`	Reports detailed information for the specified state retention registers.
`-power_domain` *`power_domain_list`*	Reports all state retention registers in the specified power domains.
`-power_gating_pin_driver {`*`pin`*`|`*`port`*`}`	Reports all state retention registers with the specified drivers.
`-verbose`	Lists the full paths of the state retention registers and power gating pins.

Examples

- The following command requests a hierarchical report. This report shows how many sequential instances are mapped to state retention registers and indicates why some sequential instances were not mapped.

```
rc:/> report state_retention -hier
============================================================
  Generated by:           Encounter(R) RTL Compiler version
  Generated on:           date
  Module:                 counter
  Library domain:         umc_08v
    Domain index:         0
    Technology libraries: scmetropmk_umc13sp_tt_0p8v_25c 1.0
                          scmetropmk_umc13sp_tt_0p8v_1p2v_25c 1.0
                          scmetro_umcl130e_ll_tt_0p8v_25c 1.0
                          GS60_W_125_1.08_CORE_RET_SNPM.db
    Operating conditions: _nominal_ (balanced_tree)
  Library domain:         umc_10v
    Domain index:         1
    Technology libraries: scmetropmk_umc13sp_tt_1v_25c 1.0
                          scmetropmk_umc13sp_tt_0p8v_1v_25c 1.0
                          scmetro_umcl130e_ll_tt_1p0v_25c 1.0
                          GS60_W_125_1.08_CORE_RET_SNPM.db
    Operating conditions: _nominal_ (balanced_tree)
  Library domain:         umc_120v
    Domain index:         2
    Technology libraries: scmetropmk_umc13sp_tt_1p2v_25c 1.0
                          scmetropmk_umc13sp_tt_1v_1p2v_25c 1.0
                          scmetro_umcl130e_ll_tt_1p2v_25c 1.0
                          GS60_W_125_1.08_CORE_RET_SNPM.db
    Operating conditions: _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

Summary
-------
=======================================================================
                       Category                        Number     %
=======================================================================
Number of Sequential instances mapped to SR cells        20     68.97
Number of Sequential instances not mapped to SR cells     9     31.03
-----------------------------------------------------------------------
Total sequential instances in the design                29.0     100
-----------------------------------------------------------------------

Reason why not mapped to SR cell
-----------------------------------------------------------------------
Excluded from mapping (no rule specified)                 9     31.03
Did not find appropriate SR cell                          0      0.00
-----------------------------------------------------------------------

Note: SR stands for State Retention

20
```

- The following command shows the detailed report of the state retention registers in hierarchical instance `monitor_power`. It shows for each sequential instance that was replaced, the library cell that was used, to which power domain the instance belongs, and the names of the power gating pins. The report also indicates that all sequential instances in this hierarchical instance were remapped to state retention registers.

```
rc:/> report state_retention [find / -inst monitor_power]
...
=====================================================================================
    Module           State          Libcell       Power        Power       % of total
                   Retention                      Domain       Gating         seqs
                    Instance                                    Pins
=====================================================================================
monitor_power/   cst_..._reg[0]   umc_..F2_SNPM   PDmon    1: (..NDRIVEN   27.59
                                                           2: (..~en[1]}
                 cst_..._reg[1]   umc_..F2_SNPM   PDmon    1: (..{en[0]}
                                                           2: (..~en[1]}
                 cst_..._reg[2]   umc_..F2_SNPM   PDmon    1: (..{en[0]}
                                                           2: (..~en[1]}
                 cst_..._reg[3]   umc_..F2_SNPM   PDmon    1: (..{en[0]}
                                                           2: (..~en[1]}
                 cst_reg[0]       umc_..F2_SNPM   PDmon    1: (..{en[0]}
                                                           2: (..~en[1]}
                 cst_reg[1]       umc_..F2_SNPM   PDmon    1: (..{en[0]}
                                                           2: (..~en[1]}
                 cst_reg[2]       umc_..F2_SNPM   PDmon    1: (..{en[0]}
                                                           2: (..~en[1]}
                 cst_reg[3]       umc_..F2_SNPM   PDmon    1: (..{en[0]}
                                                           2: (..~en[1]}
=====================================================================================


Summary
-------
=======================================================================
                       Category                          Number     %
=======================================================================
Number of Sequential instances mapped to SR cells          8     27.59
Number of Sequential instances not mapped to SR cells      0      0.00
-----------------------------------------------------------------------
Total sequential instances in the design                 29.0     100
-----------------------------------------------------------------------

Reason why not mapped to SR cell
-----------------------------------------------------------------------
Excluded from mapping (no rule specified)                  0      0.00
Did not find appropriate SR cell                           0      0.00
----------------------------------------------------------------------------
```

- The following command shows the detailed hierarchical report for all state retention registers in power domain `PDbin`. The instance names, library cell names, and power gating pins are all abbreviated. To see the full names, add the `-verbose` option to the command.

```
rc:/> report state_retention -power_domain PDbin -hier -detail
...
=====================================================================================
     Module          State          Libcell       Power       Power      % of total
                   Retention                      Domain      Gating        seqs
                    Instance                                   Pins
=====================================================================================
binary_counter/   bin_.._reg[0]   umc_..F2_SNPM   PDbin    1: (save) 0     13.79
                                                           2: (..g1660/Y
                  bin_.._reg[1]   umc_..F2_SNPM   PDbin    1: (save) 0
                                                           2: (..g1660/Y
                  bin_.._reg[2]   umc_..F2_SNPM   PDbin    1: (save) 0
                                                         2: (..g1660/Y
                  bin_.._reg[3]   umc_..F2_SNPM   PDbin    1: (save) 0
                                                           2: (..g1660/Y
=====================================================================================


Summary
-------
=======================================================================
                       Category                          Number     %
=======================================================================
Number of Sequential instances mapped to SR cells          4     13.79
Number of Sequential instances not mapped to SR cells      1      3.45
-----------------------------------------------------------------------
Total sequential instances in the design                 29.0     100
-----------------------------------------------------------------------

Reason why not mapped to SR cell
-----------------------------------------------------------------------
Excluded from mapping (no rule specified)                  1      3.45
Did not find appropriate SR cell                           0      0.00
-----------------------------------------------------------------------


Note: SR stands for State Retention
      ~ stands for active low signal

4
```

- The following command reports all state retention instances with the specified power gating pin.

```
report state_retention \
-power_gating_pin_driver [find / -pin monitor_power/g1660/Y ] -hier -detail
...
=====================================================================================
     Module           State           Libcell       Power       Power       % of total
                    Retention                       Domain      Gating         seqs
                     Instance                                    Pins
=====================================================================================
binary_counter/   bin_.._reg[0]   umc_..F2_SNPM   PDbin    1: (save) 0     13.79
                                                           2: (..g1660/Y
                  bin_.._reg[1]   umc_..F2_SNPM   PDbin    1: (save) 0
                                                           2: (..g1660/Y
                  bin_.._reg[2]   umc_..F2_SNPM   PDbin    1: (save) 0
                                                           2: (..g1660/Y
                  bin_.._reg[3]   umc_..F2_SNPM   PDbin    1: (save) 0
                                                           2: (..g1660/Y
=====================================================================================


Summary
-------
=======================================================================
                       Category                          Number     %
=======================================================================
Number of Sequential instances mapped to SR cells           4     13.79
Number of Sequential instances not mapped to SR cells       9     31.03
-----------------------------------------------------------------------
Total sequential instances in the design                  29.0     100
-----------------------------------------------------------------------

Reason why not mapped to SR cell
-----------------------------------------------------------------------
Excluded from mapping (no rule specified)                   9     31.03
Did not find appropriate SR cell                            0      0.00
-----------------------------------------------------------------------

Note: SR stands for State Retention
      ~ stands for active low signal

4
```

Related Information

Affected by these commands:	commit_cpf on page 752
	read_cpf on page 760

report summary

```
report summary
     [-mode mode] [-all]
     [design]... [> file]
```

Reports the area by mode used by the design, cells mapped for the blocks in the specified design, the wireload model, and the timing slack of the critical path. It also reports if any design rule is violated and the worst violator information.

Options and Arguments

`-all`	Reports all timing and DRC violations in the design.
`design`	Specifies the design for which you want to generate a report. By default, a report is created for all designs currently loaded in memory.
`file`	Specifies the name of the file to which to write the report.
`-mode` *`mode`*	Specifies the mode for which the report must be specified. **Note:** This option is only required for a design using a dynamic voltage frequency scaling methodology.

Examples

- The following example generates a report of the area used by the design, the worst timing endpoint in the design, and the design rule violations summary.

```
rc:/> report summary
============================================================
  Generated by:          RTL Compiler (RC) version
  Generated on:          date
  Module:                alu
  Technology library:    tutorial 1.0
  Operating conditions:  typical_case (balanced_tree)
  Wireload mode:         enclosed
============================================================

        Timing
        ------
 Slack       Endpoint
-------------------------------------

 -1082ps out1_tmp_reg[9]/D

        Area
        ----
Instance       Cells   Cell Area   Net Area     Wireload
```

```
-------------------------------------------------------------
gen_test         326        525         0    AL_MEDIUM (S)
 (S) = wireload was automatically selected
        Design Rule Check
        -----------------
Max_transition design rule: no violations.
Max_capacitance design rule (violation total = 19402.5)
Worst violator:
Pin                         Load (ff)           Max     Violation
-----------------------------------------------------------------
in0[5] (Primary Input)          96.9           5.0          91.9
Max_fanout design rule (violation total = 16.000)
Worst violator:
Pin                            Fanout           Max     Violation
-----------------------------------------------------------------
in0[5] (Primary Input)         8.000         4.000         4.000
```

Related Information

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affected by this command:	synthesize on page 294
	create_mode on page 237

report timing

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

Generates a timing report of the current design. By default, the report gives a detailed view of the critical path of the current design. If the current session has multiple designs, use the `cd` command to navigate to the desired design to generate the report. You can also generate a report on possible timing constraint problems (timing lint) or view slack at endpoints.

Note: All values are always expressed in picoseconds. This unit cannot be changed.

Important

When RTL Compiler detects a combinational feedback loop, it inserts a buffer from the technology library as a loop breaker instance before it performs timing analysis. To add the loop breaker, RTL Compiler might first need to uniquify a hierarchical instance which can result in a change in the netlist. As the module and instance name can change, this can affect your scripts and your database search.

Where applicable, special footnotes are used as shown in the table below to enhance the usability of the report.

Footnote	Meaning
`(*)`	Zero Slack Borrow Limit = This is the maximum amount that can be borrowed without violating timing on the lending side
`(@)`	Annotated capacitance
`(a)`	Net has asynchronous load pins which are being considered ideal
`(b)`	Timing paths are broken
`(C)`	Cell belongs to congested region (applies only to physical flow)
`(i)`	Net is ideal
`(m)`	Attribute `cell_delay_multiplier` is modified for this library cell
`(P)`	Instance is preserved
`(p)`	Instance is preserved but may be resized
`(u)`	Net has unmapped pin(s)

`(V)`	Net with virtual buffer(s)

Options and Arguments

`-cost_group` *`cost_group`*

Reports only paths for the specified cost groups.

`file`	Specifies the name of the file to which to write the report.
`-endpoints`	Reports the slack at all timing endpoints in the design instead of the detailed path report. The most critical endpoints are listed first.
`-exceptions`	Reports only paths to which one of the specified exceptions applies. **Note:** This option can be combined with `-from`, `-through`, `-to`, and `-endpoints` options to further restrict the path reporting.

`-from {`*`instance`*`|` *`external_delay`*`|` *`clock`* `|` *`port`* `|` *`pin`*`}`

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, or a combination of these, instances, or input ports to which the specified external delay timing applies.

`-full_pin_names`	Prints the full hierarchical path of each pin in the report. You can use these pin names from the report to paste into other commands.
`-gtd`	Generates the MRTRF (Machine readable format) output out of RTL Compiler. This is a file format used for representing timing information that can be understood by the Global Timing Debug (GTD) viewer in Encounter. The GTD viewer in Encounter parses this file and represents the timing information in graphical format. This helps illustrate the total slack, failing paths, components contributing the most delay in each path, the total number of violations, and more in the design.

`-gui`	Allows you to create a detailed timing report in the GUI without having to use the *menu* commands. By using this option from the command line you can fine-tune the report using all command-line options which are not all available in the dialogs. **Note:** This option has only effect when you are running the tool in GUI mode.
`-lint`	Reports, in an abbreviated output, possible timing problems in the design, such as ports that have no external delays, timing exceptions that cannot be satisfied, constraints that may have no impact on the design, and so on.
`-mode` *`mode_name`*	Reports the worst timing across all modes or analyzes timing in one particular mode.
`-num_paths` *`integer`*	
	Specifies the maximum number of paths to report. *Default:* the value of `-worst` **Note:** When combined with the `-endpoints` option, the number of endpoints is limited to the specified number.
`-paths` *`string`*	Reports only the specified timing restricted paths. Create the string argument using the `specify_paths` command.
`-physical`	Reports physical information, like the x, y location.
`-slack_limit` *`integer`*	
	Reports only paths with a slack smaller than the specified number.
`-summary`	Generates a short timing report that includes timing slack, start-point and end-point but does not include the full path.
`-through {`*`instance`*` | `*`port`*` | `*`pin`*`}`	
	Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/ mapped combinational instances. You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-to {`*`instance`*` | `*`external_delay`*` | `*`clock`*` | `*`port`*` | `*`pin`*`}`

Specifies a Tcl list of endpoints for the paths. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies.

Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects have the exception applied to them.

`-worst `*`integer`* Specifies the maximum number of paths to report to each endpoint.

Default: `1`

Examples

- The following example generates a report for the worst path to each of the four most-constrained endpoints. The extraction of the report shows four different endpoints:

```
rc:/designs/sample_design> report timing -num_paths 4
============================================================
  Generated by:          RTL Compiler (RC) version
  ...
============================================================

path   1:

    Pin                Type      Fanout  Load  Slew   Delay   Arrival
                                         (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
...
------------------------------------------------------------------------
Timing slack :     543ps
Start-point  : accum[1]
End-point    : aluout_reg_7/D

path   2:

    Pin                Type      Fanout  Load  Slew   Delay   Arrival
                                         (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
...
------------------------------------------------------------------------
Timing slack :     547ps
Start-point  : accum[1]
End-point    : aluout_reg_6/D

path   3:

    Pin                Type      Fanout  Load  Slew   Delay   Arrival
                                         (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
```

```
...
------------------------------------------------------------------------
Timing slack :    1030ps
Start-point  : accum[1]
End-point    : aluout_reg_5/D

path   4:

    Pin               Type      Fanout  Load  Slew   Delay   Arrival
                                        (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
...
------------------------------------------------------------------------
Timing slack :    1034ps
Start-point  : accum[1]
End-point    : aluout_reg_4/D
```

- The following example also generates a report for the four worst paths in the current design, but compared to the previous example, three of the worst paths now have the same endpoint.

```
rc:/designs/sample_design> report timing -num_paths 4 -worst 4
============================================================
  Generated by:          RTL Compiler (RC) version
  ...
============================================================

path   1:

    Pin               Type      Fanout  Load  Slew   Delay   Arrival
                                        (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
...
------------------------------------------------------------------------
Timing slack :     543ps
Start-point  : accum[1]
End-point    : aluout_reg_7/D

path   2:

    Pin               Type      Fanout  Load  Slew   Delay   Arrival
                                        (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
...
------------------------------------------------------------------------
Timing slack :     543ps
Start-point  : data[1]
End-point    : aluout_reg_7/D

path   3:

    Pin               Type      Fanout  Load  Slew   Delay   Arrival
                                        (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
...
------------------------------------------------------------------------
Timing slack :     543ps
Start-point  : accum[1]
End-point    : aluout_reg_7/D
```

```
path   4:

    Pin               Type      Fanout  Load  Slew   Delay   Arrival
                                        (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
...
------------------------------------------------------------------------
Timing slack :     547ps
Start-point  : accum[1]
End-point    : aluout_reg_6/D
```

- The following example reports the slack at the four most-constrained endpoints:

```
rc:/designs/sample_design> report timing -endpoints -num_paths 4
============================================================
  Generated by:          RTL Compiler (RC) version
  ...
============================================================

 Slack       Endpoint
-------------------------------------
  +543ps aluout_reg_7/D
  +547ps aluout_reg_6/D
 +1030ps aluout_reg_5/D
 +1034ps aluout_reg_4/D
```

- The following example reports the path from input port ‘a’ that has the least slack:

```
rc:/> report timing -from [find / -port accum[1]]
============================================================
  Generated by:          RTL Compiler (RC) version
  ...
============================================================

    Pin               Type      Fanout  Load  Slew   Delay   Arrival
                                        (fF)  (ps)    (ps)    (ps)
------------------------------------------------------------------------
(clock clock)            launch                                    0 R
(in_del_1)               ext delay                    +1000    1000 F
alu/accum[1]        <<<  in port       6   66.5     0      +0    1000 F

...
------------------------------------------------------------------------
Timing slack :     543ps
Start-point  : accum[1]
End-point    : aluout_reg_7/D
```

- The following example reports the physical information:

```
rc:/> report timing -physical

=================================================================
...
=================================================================
      Pin              Type     Fanout Load Slew Delay Arrival    Location
                                       (fF) (ps)  (ps)   (ps)      (x,      y)
---------------------------------------------------------------------------------
(clock clock1)        launch                                 0 R
wr_addr_reg[0]/CK                                 0          0 R
wr_addr_reg[0]/Q  (@)  SDFFRHQX1      2  6.0  118  +311     311 R  (107640, 51660)
g154/A                                      118    +0     311
g154/Y            (@)  CLKMX2X2       3 10.9  105  +206     517 R  (102580, 51660)
g146/B                                      105    +0     517
g146/CO           (@)  ADDHXL         1  4.6  167  +189     706 R  (101660, 59040)
```

```
g145/A                                      167    +0     706
...
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock1)        capture                           10000 R
------------------------------------------------------------------------------
Timing slack :    7997ps
Start-point  : wr_addr_reg[0]/CK
End-point    : wr_addr_reg[7]/D

(@) : Annotated capacitance.
```

- The following example reports only a condensed version of the timing report:

  ```
  rc:/> report timing -summary
  ```

  ```
  Timing slack : 9744ps
  Start-point  : in1[3]
  End-point    : out4
  ```

- The following example reports the path with the least slack that uses a `multi_cycle` exception named 'mc_2'

  ```
  rc:/> report timing -exceptions [find / -exception mc_2]
  ```

- The following example reports only the worst path being launched by clock `clk1`

  ```
  rc:/> report timing -paths [specify_paths -from clk1]
  ```

- The following example only prints the first three objects in each lint category. If there are more than three objects, a related message is issued:

  ```
  report timing -lint
  ```

  ```
  /designs/test/ports_out/SO1
  /designs/test/ports_out/SO2
  /designs/test/ports_out/SO3
  ......
  20 other pins in this category. Use the -verbose option for more details:
  ```

  ```
  report timing -lint -verbose
  ```

- The following example illustrate how to use the `-gtd` option:

  ```
  rc:/> report_timing -from ..... -to ..... -gtd > viol.mtarpt
  ```

  Open `viol.mtarpt` in the global timing debug viewer in Encounter.

Related Information

Checking the Constraints Using the report timing-lint Command in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Performing Multi-Mode Timing Analysis in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affected by this command:	create_mode on page 237
	specify_paths on page 274
	synthesize on page 294

report yield

```
report yield [-depth integer] [-min_count integer]
     [> file]
```

Reports the yield cost and yield percentage for each instance. This command is used in the design for manufacturing (DFM) flow.

Options and Arguments

`-depth` *`integer`*	Specifies the number of levels of recursion.
`-min_count` *`integer`*	Specifies the minimum instance count per block.
`file`	Redirects the report to the specified file.

Example

- The following example shows the defect-limited yield impact for library cell defects:

```
rc:/> report yield

Instance     Cells   Cell Area            Cost     Yield %
----------------------------------------------------------
cpu            470         659    1.600606e-05     99.9984
  alu1         248         283    7.606708e-06     99.9992
  pcount1       65          92    2.215669e-06     99.9998
  ireg1         33          88    1.629471e-06     99.9998
  accum1        33          88    1.629471e-06     99.9998
  decode1       50          67    1.568901e-06     99.9998
```

Related Information

Design For Manufacturing Flow in *Encounter RTL Compiler Synthesis Flows*

Affected by this command:	read_dfm
Related command:	report gates -yield
Affected by this attribute:	optimize_yield

timestat

```
timestat
     [string] [> file]
```

Reports the runtime and memory used up to this stage (time that the information was requested).

Options and Arguments

string	Specifies a user-defined string which allows you to identify at which stage in the design the information was requested. *Default:* `undefined`.
file	Redirects the command output to the specified file.

Examples

- The following script extract requests the information after RTL optimization.

  ```
  synthesize -to_generic -eff $SYN_EFF
  puts "Runtime & Memory after ’synthesize -to_generic’"
  timestat GENERIC
  ```

- The following example shows the output after mapping.

  ```
  rc:/> timestat map
  ===========================================
  The RUNTIME after map is 0.45 secs
  and the MEMORY_USAGE after map is 24.16 MB
  ===========================================
  ```

- The following example shows the output if no user-defined string was specified.

  ```
  rc:/> timestat
  ===========================================
  The RUNTIME after undefined is 0.45 secs
  and the MEMORY_USAGE after undefined is 24.16 MB
  ===========================================
  ```

validate_timing

```
validate_timing
     [-sdc sdc_files] [-netlist path] [-libs lib_list]
     [-include file | -rep_tim_str command]
     [-keep_temp_dir] [> file]
```

Generates an Encounter Timing System timing report.

To run this command you need to have access to the Encounter ® Timing System software.

Options and Arguments

file	Specifies the name of the file to which the report must be written. *Default*: `vtim_ets_timing_rpt`
`-include` *file*	Specifies the file containing the Encounter Timing System commands to be executed. These commands will be read in after the libraries, netlist and SDC constraints have been read into the Encounter Timing System tool.
`-keep_temp_dir`	Does not remove the temporary (`.vtim_ets`) directory in which the tool generates the SDC file or netlist.
`-libs` *lib_list*	Specifies the list of libraries to be read by Encounter Timing System. If no libraries are specified, the same libraries that were specified for synthesis are used.
`-netlist path`	Specifies the path to the netlist. If no netlist is specified, the netlist is generated with the `write_hdl` command in the `.vtim_ets` directory.
`-rep_tim_str` *command*	Specifies a string that contains a single Encounter Timing System report command.
`-sdc` *file_list*	Specifies the SDC file(s) for the design. If no SDC file(s) are specified, the SDC files are generated using the `write_sdc` command in the `.vtim_ets` directory.

Examples

- The following command reads the Encounter Timing System commands to be executed from the `ets_include` file.

  ```
  validate_timing -netlist test.v -libs $env(REGLIBS)/tutorial.lib \
  -include ets_include
  ```

  Because no SDC file was specified, the `vtim_run_ets.sdc` file is generated in the `.vtim_ets` directory.

- The following command directly specifies which Encounter Timing System command to be execute.

  ```
  validate_timing -rep_tim_str "report_timing"
  ```

9 Physical

def_move

```
def_move
     [-initialize]
     [-highlight]
     [-min_distance string]
```

Highlights cell movement in the physical tab of the GUI.

Use this command after you run the `synthesize -to placed` command.

Options and Arguments

`-highlight`	Highlights the cell movement with respect to their last stored location.
`-initialize`	Stores the location of the instances at the time the command is given with this option.
`-min_distance` *`string`*	Limits the highlighting to cells that have been moved more than the specified distance. Specify the distance in microns. If this option is omitted, highlighting is limited to those cells whose x and y locations both changed by a value greater than or equal to the row height.

Example

Following illustrates the usage:

```
def_move -initialize
synthesize -to_placed -incr
def_move -highlight
```

Related Information

Related command:	synthesize on page 294

predict_qos

```
predict_qos [-reference_config_file string]
     [-parasitic_output_file string]
     [-abandon_existing_placement]
     [-ignore_scan_chains]
     [design]
```

Analyzes and optimizes the design for silicon. This process enhances the correlation between results from place and route pre-clock tree synthesis and the results from RTL Compiler. The `predict_qos` command invokes Encounter. You will need an RC 400 license to be available prior to the command's execution.

Specifically, the `predict_qos` command generates a Silicon Virtual Prototype (SVP) to gauge the quality of silicon of the design. The steps in the SVP creation process include:

- Placement
- Trial route
- Parasitic extraction

The detailed placement information and the resistance and capacitance parasitics are then used for delay calculation and annotation of physical delays. The `predict_qos` command will operate in incremental mode if the standard cells are placed. Use `-abandon_existing_placement` option to suppress this behavior. The `predict_qos` command will perform virtual buffering by default.

You should use the `synthesize -incremental` immediately after `predict_qos` command for the best results.

The `predict_qos` command will not work with encrypted netlists.Therefore, de-encrypt your netlist before using the `predict_qos` command.

Options and Arguments

`-abandon_existing_placement`

Discards instance placement information.

`design`	Specifies the design for QoS prediction.

`-ignore_scan_chains`

Ignores the scan chain connections during placement estimation.

`-parasitic_output_file` *`string`*

Outputs the parasitics from QoS prediction in SPEF format to the specified filename.

`-reference_config_file` *`string`*

Specifies the configuration file to use as a template.

Examples

- The following example specifies the `penny.conf` as the configuration file template and `hillary.spef` as the outputted SPEF file:

```
rc:/> predict_qos -reference_config_file rc_enc_des/penny.conf \
    -parasitic_output_file hscott.spef
```

Related Information

Related command:	synthesize on page 294

read_def

```
read_def
     [-hierarchical] [-incremental] [-no_specialnets]
     def_file
```

Loads the specified DEF file.

RTL Compiler will perform a consistency check between the DEF and the Verilog netlist and issue relevant messages if necessary. The DEF file must define the die size. A warning message will be issued for any components that lie outside the die area. For better synthesis results, you should also have the pin and macro locations specified in the DEF, although it is not required.

RTL Compiler supports DEF 5.3 and above.

Important

The information extracted from the DEF file depends on the license you use to start the tool. In most cases, the `read_def` command will only extract the die size information (aspect ratio in particular) from the DEF file. Other floorplan information (such as, placed objects, blockages, constraints, and so on) will only be extracted if you start the tool with an `RTL_Compiler_Physical` license.

Options and Arguments

`def_file`	Specifies the DEF file.
`-design` *`design`*	Specifies the design to which to annotate the DEF information.
`-hierarchical`	Specifies that the DEF file is hierarchical.
`-incremental`	Specifies that the DEF file has incremental information. Therefore only updates of the pins and components are needed. ■ If a pin or component has physical data from the original DEF file, the physical data in the incremental DEF file will overwrite the original data. ■ If a pin or component did not have physical data in the original DEF file, physical data in the incremental DEF are used.
`-no_specialnets`	Specifies not to read the `SPECIALNETS` section in the DEF file.

Example

- The following example loads the `point.def` DEF file:

```
rc:/> read_def point.def
  Reading DEF file 'point.def'...
Warning : A DEF component does not exist in the netlist. [PHYS-171]
        : The component 'U1/foo1' does not exist.
Warning : A DEF component does not exist in the netlist. [PHYS-171]
        : The component 'U1/foo4' does not exist.
Info    : A COVER component has been read. [PHYS-182]
        : The instance 'U1/g20' is COVER.
Info    : A COVER component has been read. [PHYS-182]
        : The instance 'U1/g21' is COVER.
  Summary report for DEF file 'point.def'
    Components
    ----------
        Cover:   2
        Fixed:   8
     Physical:   2
       Placed: 594
     Unplaced:   2
        TOTAL: 608

       Macros:   0

          Pins
          ----
        Cover: 0
        Fixed: 0
       Placed: 0
     Unplaced: 0
        TOTAL: 0
```

Related Information

Related attribute:	phys_ignore_special_nets

read_encounter

`read_encounter config` *`configuration_file`*

Reads an Encounter configuration file into RTL Compiler. An Encounter configuration file is an ASCII file that contains Tcl variables that describe information such as the netlist or RTL, technology libraries, LEF information, constraints, and capacitance tables. Encounter configuration files have the `.config` extension.

After the file is loaded, the constraints and attributes specified in the configuration file will automatically be set. Hence, the design will be ready for synthesis or optimization or both.

Options and Arguments

`configuration_file`	Specifies the configuration file to load.

Examples

- Since the configuration file contains information such as technology libraries, HDL files, and constraints, the `read_encounter` command should be used at the beginning of a synthesis session. After the configuration file is loaded, you can immediately synthesize or optimize the design. The following example loads the `fast.config` configuration file, then synthesizes the design to gates.

```
rc:/> read_encounter config fast.config
rc:/> synthesize -to_mapped
...
```

Related Information

Related command:	write_encounter on page 202

read_spef

```
read_spef spef_file
     [-max_fanout integer]
     [-hierarchical] [-incremental]
```

Reads the SPEF file and loads the resistance and grounded capacitors from the file. Gzip compressed files (`.gz` extension) can also be loaded. In RTL Compiler, the SPEF file is generated by Encounter.

Options and Arguments

`-hierarchical`	Specifies that SPEF file is hierarchical.
`-incremental`	Specifies that the SPEF file contains incremental information. Allows to read in the data without resetting the original SPEF annotated values (if any).
`-max_fanout`	Any net with a fanout count greater than the specified value will not be annotated with the resistance and capacitance from the SPEF. Also, delay calculation will not be performed. The default value is1000.
`spef_file`	Specifies the SPEF file.

Related Information

Related command:	write_spef on page 463

report congestion

Refer to `report congestion` in Chapter 8, “Analysis and Report.”

reset_def

```
reset_def
```

Removes all physical information from the design.

Related Information

Related command:	read_def on page 453

specify_floorplan

```
specify_floorplan
     { -die_box {llx lly urx ury}
     | -height float -weight float }
     [-core_box {llx lly urx ury} ]
     design
```

Specifies the floorplan information for the specified design.

Options and Arguments

`-core_box {`*`llx lly urx ury`*`}`	Specifies the lower left and upper right coordinates of the core of the design.
`design`	Specifies the name of the design.
`-die_box {`*`llx lly urx ury`*`}`	Specifies the lower left and upper right coordinates of the die of the design.
`-height` *`float`*	Specifies the height of the die.
`-width` *`float`*	Specifies the width of the die.

Examples

- The following command specifies the coordinates of the die and the core of the design.

```
specify_floorplan MYCHIP -die_box {0 0 1550 1500} -core_box {100 100 1400 1400}
```

- The following commands first specify the width and height of the die, then the coordinates of the core. Parameter legality checking is performed as shown below.

```
rc:/> specify_floorplan DTMF_CHIP -height 1400 -width 1200
rc:/> specify_floorplan DTMF_CHIP -core_box {100 100 1400 1400}
Error   : The design core box must lie within the die box. [PHYS-102]
[specify_floorplan]
        : core box = {100 100 1400 1400}, die box = {0 0 1200 1400}
        : Wrong coordinates were specified for the core box.
```

Related Information

Affects this command:	synthesize on page 294

update_congestion_map

```
update_congestion_map design
```

Updates the congestion map for the specified design.

You can run this command after any command that updates the physical data.

write_def

```
write_def [-ignore_groups] [-ignore_placed_instances]
     [-scan_chains] [design] [> file]
```

Writes a floorplan, in DEF format, for the specified design. RTL Compiler does not store all the information from the original DEF (for example, `VIAS`, `SLOTS`, `ROWS`, `TRACKS`, etc.). However, the generated floorplan includes data from both the RTL Compiler session as well as the original imported DEF. The DEF does not contain the netlist information (net connectivity information) aside from the power/ground nets defined in the input DEF (`SPECIALNETS` section).

You can write out the DEF in gzip format by specifying the `.gz` extension when writing out the file.

Options and Arguments

`design`	Specifies a particular design for which to write out the floorplan. Only one design can be specified at a time.
`file`	Redirects the floorplan to the specified file.
`-ignore_groups`	Discards the instance groups that are defined in RC. These come from the input DEF (`GROUPS` section). The output DEF will have no `GROUPS` or `REGIONS` sections.
`-ignore_placed_instances`	Only writes preplaced instances (`+ FIXED` tag in the DEF) to the DEF. These are objects such as macros. Without this option, the output DEF will include all the instances that are preplaced or placed. Unplaced components will never be written to the DEF. This is all in the `COMPONENTS` section.
`-scan_chains`	Specifies that scan chain information should be included in the floorplan.

Related Information

Related attribute:	phys_ignore_special_nets

write_spef

```
write_spef [> file]
```

Writes out the parasitics (resistance and capacitance information) of the design in SPEF (Standard Parasitic Exchange Format) format.

Options and Arguments

file	Specifies the file to which to write the parasitics.

Related Information

Related command:	read_spef on page 456

10 Quality Analyzer

Important

These commands are only available when you run the tool in RCQA mode.

add_rule_group

```
add_rule_group rule_group_name
     -label label_name
     [-tool tool_name]
     [-disable]
```

Defines a rule group under the predefined existing rule set.

Options and Arguments

`-disable`	Disables all rule that belong to the rule group. The supported values of tool are ""(null) and RC.
`-label` *`label_name`*	Specifies a label for the rule group. Labels with more than one separated word, such as `custom group`, must be enclosed in quotes (see example).
`-tool` *`tool_name`*	Specifies the tool for running the rules in the rule group.
`rule_group_name`	Specifies the name of the rule group.

Example

The following command sequence creates a new rule group called `user_rule_grp` with a label name of `custom group` in the rule set `ALL_DFT`:

```
rcqa:/> cd /rules/ALL_DFT
rcqa:/rules/ALL_DFT\> add_rule_group user_rule_grp -label “custom group”
```

launch

```
launch
     { clock_domain_crossing_checker
     | constraint_checker
     | equivalence_checker
     | low_power_checker
     | timing_validator }
```

Launches the various engines in GUI mode. All of the required scripts files are generated in the respective directories. You can modify the script files and run them separately.

Note: This command launches the tools in GUI mode, but not in the batch mode, so you can only exit the tool by clicking *Exit*, not *Exit GUI*.

Options and Arguments

`clock_domain_crossing_checker`	Launches the clock domain crossing checker
`constraint_checker`	Launches the constraint checker
`equivalence_checker`	Launches the equivalence checker
`low_power_checker`	Launches the low power checker
`timing_validator`	Launches the timing validator

read_config_file

`read_config_file` *`file`*

Loads a configuration file.

Options and Arguments

`file`	Specifies the name of the configuration file to load.

Example

The following command loads a configuration file named `config.file`:

```
rcqa:/> read_config_file config.file
```

Related Information

Supported Variables for Configuration Files in *Using Encounter RTL Compiler Quality Analyzer*

report buskeepers

Refer to report buskeepers in Chapter 8, “Analysis and Report.”

report checks

Refer to report checks in Chapter 8, “Analysis and Report.”

report spare_instances

Refer to report spare_instances in Chapter 8, “Analysis and Report.”

reset_session

```
reset_session [design]
```

Resets the quality analyzer session with its design, library and additional information for re-running the checks.

Options and Arguments

design	Specifies the name of the design.

Examples

The following commands show an example of the command sequence when using `reset_session`.

The first set of commands load a configuration file named `config.file` and runs all signoff checks:

```
rcqa:/> read_config_file config.file
rcqa:/> signoff_checks all
```

The next set of commands reset the session to the point before running the checks, and reloads the same configuration file:

```
rcqa:/> reset_session
rcqa:/> read_config_file config.file
```

Related Information

Related commands:	restore_session on page 473
	save_session on page 474

restore_session

```
restore_session
     -file file
     [-db]
```

Restores the saved quality analyzer session that was saved with the `save_session` command. This command loads the configuration file, library information, design information, and saved check results.

Options and Arguments

`-db`	Restores the database file that was specified with the `save_session -db` command.
`-file` *`file`*	Specifies the name of the saved session file.

Examples

- The following command restores the results of the logical design signoff checks from a saved file named `sessionA.clss`:

  ```
  rcqa:/> restore_session -file sessionA.clss
  ```

- The following command restores the results of `1.clss` and loads for the database file `1.clss.db`:

  ```
  rcqa:/> restore_session -file 1.clss -db
  ```

Related Information

Related commands:	reset_session on page 472
	save_session on page 474

save_session

```
save_session
     -to_file file
     [-db]
```

Saves the results of the logical design signoff checks in the current session to an encrypted file.

You can restore the saved session file using the `restore_session` command.

Options and Arguments

`-db`	Saves the database file in addition to the session file with a `.db` extension.
`-to_file` *`file`*	Specifies the name of the session file.

Examples

- The following command saves the results of the logical design signoff checks to a file named `sessionA.clss`:

  ```
  rcqa:/> save_session -to_file sessionA.clss
  ```

- The following command generate two files: `1.clss` with all the RCQA related setup, and `1.clss.db`:

  ```
  rcqa:/> save_session -to_file 1.clss -db
  ```

Related Information

Related commands:	reset_session on page 472
	restore_session on page 473

signoff_checks

```
signoff_checks { all | clock_domain_crossing | constraints | dft
     | hdl_lint | library | physical | power }
```

Runs the specified rule checks. Use this command only after loading a valid configuration file.

After running the checks, the RCQA software creates a `rcqa_*_chk` directory that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Options and Arguments

`all`	Runs all checks in this order: HDL lint, library, clock domain crossing, constraints, power, and DFT.
`clock_domain_crossing`	Runs clock domain crossing checks.
`constraints`	Runs constraint validation (SDC) checks.
`dft`	Runs Design for Test (DFT) checks.
`hdl_lint`	Runs HDL analysis for lint, structural, and synthesizability issues.
`library`	Runs library checks.
`physical`	Runs physical netlist checks.
`power`	Runs low power checks.

Related Information

Related commands:	signoff_checks all on page 477
	signoff_checks clock_domain_crossing on page 478
	signoff_checks constraints on page 479
	signoff_checks dft on page 480
	signoff_checks hdl_lint on page 481
	signoff_checks library on page 482
	signoff_checks physical on page 483
	signoff_checks power on page 484

signoff_checks all

```
signoff_checks all
     [> file]
```

Runs all rule checks in the following order:

- HDL lint
- Library
- Clock comain crossing
- Constraints
- Power
- DFT

Use this command only after loading a valid configuration file.

After running the checks, the RCQA software creates `rcqa_*_chk` directories that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Options and Arguments

`file`	Specifies the name of the file to direct the output.

Example

The following command runs all rule checks and outputs the results to a file named `all_checks`:

```
rcqa:/> signoff_checks all > all_checks
```

Related Information

Debug Directories in *Using Encounter RTL Compiler Quality Analyzer*

signoff_checks clock_domain_crossing

```
signoff_checks clock_domain_crossing
     [-diagnose] [-license string]
     [> file]
```

Runs clock domain crossing checks. Use this command only after loading a valid configuration file.

After running the check, the RCQA software creates an `rcqa_cdc_chks` directory that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Options and Arguments

`-diagnose`	Opens up the Clock Domain Crossing engine for debugging. After quitting the tool, it reports the checks that you can analyze using the Message browser
`file`	Specifies the name of the file to direct the output.
`-license` *`string`*	Specifies the license of the child technology.

Example

The following command runs clock domain crossing checks and enables the GXL license:

```
rcqa:/> signoff_checks clock_domain_crossing -license GXL
```

Related Information

Debug Directories in *Using Encounter RTL Compiler Quality Analyzer*

signoff_checks constraints

```
signoff_checks constraints
     [-diagnose] [-license string]
     [> file]
```

Runs constraint validation (SDC) checks. Use this command only after loading a valid configuration file.

After running the checks, the RCQA software creates an `rcqa_con_chks` directory that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Options and Arguments

`-diagnose`	Opens up the Conformal Constraint Designer engine for debugging. After quitting the tool, it reports the checks that you can analyze using the Message browser.
`file`	Specifies the name of the file to direct the output.
`-license` *`string`*	Specifies the license of the child technology.

Example

The following command runs constraint (SDC) checks and enables the GXL license:

```
rcqa:/> signoff_checks constraints -license GXL
```

Related Information

Debug Directories in *Using Encounter RTL Compiler Quality Analyzer*

signoff_checks dft

```
signoff_checks dft
     [> file]
```

Runs Design for Test (DFT) checks. Use this command only after loading a valid configuration file.

After running the checks, the RCQA software creates an `rcqa_dft_chks` directory that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Note: If you specify a CPF file, the power checks must be run first before running the DFT checks.

Options and Arguments

file	Specifies the name of the file to direct the output.

Example

The following command runs DFT checks:

```
rcqa:/> signoff_checks dft
```

Related Information

Debug Directories in *Using Encounter RTL Compiler Quality Analyzer*

signoff_checks hdl_lint

```
signoff_checks hdl_lint
     [> file]
```

Runs HDL analysis for lint, structural, and synthesizability issues. Use this command only after loading a valid configuration file.

After running the checks, the RCQA software creates an `rcqa_hdl_chks` directory that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Options and Arguments

file	Specifies the name of the file to direct the output.

Example

The following command runs HDL lint checks:

```
rcqa:/> signoff_checks hdl_lint
```

Related Information

Debug Directories in *Using Encounter RTL Compiler Quality Analyzer*

signoff_checks library

```
signoff_checks library
     [> file]
```

Runs library checks. Use this command only after loading a valid configuration file.

After running the checks, the RCQA software creates an `rcqa_lib_chks` directory that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Options and Arguments

`file`	Specifies the name of the file to direct the output.

Example

The following command runs library checks:

```
rcqa:/> signoff_checks library
```

Related Information

Debug Directories in *Using Encounter RTL Compiler Quality Analyzer*

signoff_checks physical

```
signoff_checks physical
     [> file]
```

Runs physical netlist checks. Use this command only after loading a valid configuration file.

After running the checks, the RCQA software creates an `rcqa_phys_chks` directory that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Options and Arguments

`file`	Specifies the name of the file to direct the output.

Example

The following command runs physical netlist checks:

```
rcqa:/> signoff_checks physical
```

Related Information

Debug Directories in *Using Encounter RTL Compiler Quality Analyzer*

signoff_checks power

```
signoff_checks power
     [-diagnose] [-license string]
     [> file]
```

Runs low power checks. Use this command only after loading a valid configuration file.

After running the checks, the RCQA software creates an `rcqa_pow_chks` directory that generates internal files required to perform the checks that includes the log file generated by the corresponding engine. If you do not get enough information from the `rcqa.log` file to debug the issue, you can refer to the individual log file of the internal engine available in the debug directories.

Note: If you specify a CPF file, the power checks must be run first before running the DFT checks.

Options and Arguments

`-diagnose`	Opens up the Conformal Low Power engine for debugging. After quitting the tool, it reports the checks that you can analyze using the Message browser.
`file`	Specifies the name of the file to direct the output.
`-license` *`string`*	Specifies the license of the child technology.

Example

The following command runs low power checks and enables the GXL license:

```
rcqa:/> signoff_checks power -license GXL
```

Related Information

Debug Directories in *Using Encounter RTL Compiler Quality Analyzer*

write_config_template

```
write_config_template
     [> file]
```

Writes out a configuration file template that lists out the supported file variables.

Options and Arguments

file	Specifies the name of the file to direct the output.

Examples

The following command writes out a configuration file template to a file named `config.file`:

```
rcqa:/> write_config_template > config.file
```

Related Information

Supported Variables for Configuration Files in *Using Encounter RTL Compiler Quality Analyzer*

Sample Configuration Files in *Using Encounter RTL Compiler Quality Analyzer*

11

Design for Test

analyze_scan_compressibility

```
analyze_scan_compressibility
      -library atpg_libraries
     [-chains number_of_full_scan_chains...]
     [-compressor {xor | misr | mimic_bidi_misr}]
     [-decompressor {broadcast | xor}]
     [-mask {wide0|wide1|wide2}]
     [-effort atpg_effort]
     [-fault_sample_size fault_sample_size]
     [-ratio_list list_of_compression_ratios]
     [-directory directory]
     [-dont_run_atpg]
     [design]
```

Analyzes a design for scan-based compressibility and produces actual compression results for each compression setting.

Note: To use this command you need to have a license for the Encounter Test Architect tool.

Options and Arguments

`-chains` *`integer`*

Specifies the number of scan chains to be analyzed. This option must be specified if no scan chains are defined for the design.

Note: This option is ignored if the design already has scan chains defined.

Default: none

`-compressor {xor | misr | mimic_bidi_misr}`

Specifies the type of compression logic for analysis:

- `xor` analyzes an XOR-based compressor
- `misr` analyzes an on-product MISR compressor
- `mimic_bidi_misr` analyzes an on-product MISR compressor and mimics the behavior of the design as though the pads can be configured bidirectionally.

Default: xor

`-decompressor {broadcast | xor}`

Specifies the type of decompression logic for analysis:

- `broadcast` specifies broadcast-based decompression logic (simple scan fanout).
- `xor` specifies an XOR-based spreader network in addition to the broadcast-based decompression logic.

Default: `broadcast`

`design` Specifies the name of the top-level design on which to perform analysis.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-directory` *`atpg_directory`*

Specifies the directory to run and store ATPG results.

Note: ATPG results tend to consume high amounts of disk space for larger designs.

Default: `./.asc`

`-dont_run_atpg` Specifies to only insert the different compression options and produces a list of the ATPG jobs to be run.

Note: This option can be used in combination with the `report_scan_compressibility` command as the list of ATPG jobs is produced. The ATPG jobs may be run with either of the following methods:

- Run on a different workstation.
- Run in parallel mode to reduce runtime if a Load Sharing Facility (LSF) environment is available.

Important

Neither of the preceding methods are automatically invoked by the `analyze_scan_compressibility` command or the `report_scan_compressibility` command.

`-effort {low | medium | high}`

Specifies the effort to be used for ATPG.

Note: Higher effort levels on larger designs cause increased runtimes.

Default: `medium`

`-fault_sample_size` *`integer_sample_size`*

Specifies the number of faults to simulate to predict test coverage. This number affects the number of partitions or slices to be run depending on the total number of faults in the design. Specifying a higher number obtains more accurate results while specifying a lower number reduced runtime.

Defaults: `20000` or for full ATPG `-1`

`-library` *`atpg_library...`*

Specifies the ATPG library or libraries.

`-mask {wide0| wide1 | wide2}`

Specifies the scan channel masking logic type for analysis.

The masking types that can be used depend on the compressor type specified with the `-compressor` option.

Default: `wide1`

Note: The syntax indicates which types are available for each of the compressor types.

`-ratio_list` *`list_of_compression_levels`*

Specifies the list of compression ratios to be analyzed.

Default: `"20 50 100"`.

Examples

- The following command performs XOR-based compression analysis on `wide1` logic masking for a compression list of `"20 50 100"`. It only inserts the compression options with the `-dont_run_atpg` option. The command is followed by manually invoking the list of ATPG runs.

```
rc:/> analyze_scan_compressibility  -chains 8 -compressor xor -mask wide1 \
 -ratio_list "20 50 100" -dont_run_atpg -library $atpg_libraries
```

```
analyze_scan_compressibility ...
...
...
```

- The following command performs MISR-based compression analysis, with `wide2` masking, an `xor` decompressor, a ratio list of `"10 20"`, eight scan chains, and assumes the pads are bidirectional for MISR-based compression.

```
rc:>  analyze_scan_compressibility -chains 8 -decompressor xor \
-compressor  mimic_bidi_misr -ratio_list "10 20" -mask wide2
```

```
analyze_scan_compressibility ...
...
...
Design-DLX_CORE
Compressor-mimic_bidi_misr
Decompressor-xor
Mask-wide2
####################################
Analyze_dft_compressibility Results
####################################

Achieved compression table with fullscan topup vectors
##############################################################
IC    TATR   TDVR   Cov     CL      Cycles        Runtime
##############################################################
fs    1      1      99.86%  161     57318         1:20
10    7      17     99.88%  17      8159          1:50
20    10     19     99.85%  9       5581          2:00

Achieved compression table without fullscan topup vectors
##############################################################
IC    TATR   TDVR   Cov     CL      Cycles
##############################################################
fs    1      1      99.86%  161     57318
10    7      23     98.02%  17      7193
20    14     43     97.28%  9       3971
Total atpg runtime for exp. 5:10 hrs.

IC     - Inserted compression
TATR   - Test application time reduction
TDVR   - Test data volume reduction
Cov    - Atpg coverage
CL     - Channel Length
Cycles - Total no. of cycles for test
Runtime- Atpg runtime
fs     - fullscan run
```

- The following command performs a preview of an XOR-based compression analysis, with `wide1` masking, with an `xor` decompressor, a ratio list of `"5 10"`, and eight scan chains

```
rc:>  analyze_scan_compressibility -chains 8 -decompressor xor \
-library  rules/tsmc13.v -ratio_list "5" -directory $outdir/newdirx \
 -dont_run_atpg -fault_sample_size 5000 -preview
```

```
analyze_scan_compressibility -directory $outdir/newdirx....
```

```
Preview of design:
   Total number of DFT violations: 40
  Total number of Test Clock Domains: 3
  Number of user specified non-Scan registers:   5
      Number of registers that fail DFT rules:   60
      Number of registers that pass DFT rules: 1350
  Percentage of total registers that are scannable: 95%

 Max chain length:169
 Min chain length:168
###############################################
IC  TAC   No.of.SC No.of.Chann  MaxCL MaxML
###########################################
fs  -    8       -           -     -
5   5    8       40          34    5
10  9.9  8       80          17    10

IC-Inserted compression
TAC-Tool achieved compression
No.of SC- No. of full scan chains
No.of Chann-No. of internal scan channels
Max.CL- Maximum length of internal channel
Max.ML- Maximum length of mask register channel
```

Related Information

Analyzing and Reporting Scan Compressibility in *Design for Test in Encounter RTL Compiler.*

Compressing Scan Chains in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft test_clock on page 577
	define_dft shift_enable on page 572

analyze_testability

```
analyze_testability [-library string]
     [-effort {low|medium|high}]
     [-atpg_options string]
     [-build_model_options string]
     [-fault_sample_size integer]
     [-etlog file] -directory string [design]
```

Invokes Encounter Test to perform Automatic Test Pattern Generator (ATPG) based testability analysis in either assume or fullscan mode.

Note: To use this command you need to have the Encounter Test software installed and your operating system PATH environment variable must include the path to the Encounter Test software. For more information on the exact product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Options and Arguments

`-atpg_options` *string*

Specifies a string containing the extra options to run ATPG-based testability analysis.

Note: For more information on these options, refer to the `create_tests` command in the *Command Line Reference* (of the Encounter Test documentation).

`-build_model_options` *string*

Specifies a string containing the extra options to build a model.

Note: For more information on these options, refer to the `build_model` command in the *Command Line Reference* (of the Encounter Test documentation).

design Specifies the name of the top-level design on which you want to perform test analysis and test-point selection.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-directory` *string* Specifies the working directory for Encounter Test.

`-effort {low | medium | high}`

Specifies the effort to be used for the ATPG-based testability analysis.

Default: `low`

`-et_log` *`file`*

Specifies the name of the Encounter Test log file. This file will be generated in the specified directory.

Default: `eta_from_rc.log`

`-fault_sample_size` *`integer`*

Specifies the number of faults simulated to predict the test coverage. This number affects the number of partitions or slices to be run depending on the total number of faults in the design.

The default sample size gives a good estimation of fault coverage while limiting the run time. Use a higher number to get a better accuracy, or a smaller number to reduce your run time.

Default: `20(K)`

`-library` *`string`*

Specifies the list of Verilog structural library files, for example, *`file1 file2`*. Refer to the `write_et_atpg` `-library` option description for additional information.

Note: This option is only required when you invoke this command on a mapped netlist.

Examples

- The following example performs only ATPG-based testability analysis. The command generates a report on the fault coverage in the log file.

  ```
  analyze_testability
  ```

- The following command instructs to build a model using the IEEE standard Verilog parser.

  ```
  analyze_testability -build_mode_options "vlogparser=IEEEstandard"
  ```

Related Information

Using Encounter Test Software to Analyze Testability in *Design for Test in Encounter RTL Compiler.*

Affected by these constraints:	define_dft test_mode on page 581
	define_dft test_clock on page 577

check_atpg_rules

```
check_atpg_rules [-library string] [-compression]
     [-directory string] [design]
```

Generates a template script to run Encounter Test to verify if the design and its test structures are ATPG-ready. More specifically, the generated script allows you to check for

- Tristate drivers for contention

  These conditions might cause manufacturing test problems

- Feedback loops

  Failure to break combinational feedback loops might cause reduced test coverage.

- Clock signal races

  Checks for any flip-flop with a potential race condition between its data and clock signal.

- Test clock control

This command generates the following files:

- `et.exclude`—A file listing objects to be excluded from the ATPG analysis
- `et.modedef`—A file describing the test mode when running ATPG in `assumed scan` mode
- *`topmodulename`*`.ASSUMED.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock) and their test function used to build the testmode before actual scan chains exist in the design
- *`topmodulename`*`.FULLSCAN.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- If the ATPG-based testability analysis is run in compression mode, the following pin-assignment files are generated in addition to the *`topmodulename`*`.FULLSCAN.pinassign` file. In this case, All three files include the compression test signals with their appropriate test functions to validate their specific test mode:
  - *`topmodulename`*`.COMPRESSION_DECOMP.pinassign`—A file generated *only* when inserting XOR-based decompression logic
  - *`topmodulename.`*`COMPRESSION`*`.pinassign`*—A file generated to verify the broadcast-based decompression logic

- `runet.tsv`—A template script file for Encounter Test to verify the design and its test structures

Options and Arguments

`-compression`	Performs additional checks if the design has compression logic.
`design`	Specifies the name of the top-level design for which you want to check if it ATPG-ready. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-directory` *`string`*	Specifies the working directory for Encounter Test. **Note:** If the script files already exist, rerunning the command will overwrite them. *Default*: `./check_atpg_rules`
`-library` *`string`*	Specifies the list of Verilog structural library files, for example, *`file1 file2`*. Refer to the `write_et_atpg` `-library` option description for additional information. **Note:** This option is only required when you invoke this command on a mapped netlist.

Example

```
rc:/> check_atpg_rules


Encounter Test scripts to check whether a design is ATPG ready have been written
to directory ’./check_atpg_rules’.
Invoke the scripts as ’et -e ./check_atpg_rules/runet.tsv’
```

check_dft_pad_configuration

`check_dft_pad_configuration [`*`design`*`] [> `*`file`*`]`

Checks and reports the data direction control of the pad logic for the test I/O ports.

Options and Arguments

`design`	Specifies the name of the top-level design to be checked. You should specify this name in case you have multiple top designs loaded. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`file`	Specifies the file to which to redirect the detailed output. If no file is specified, the output is written to standard out (`stdout`) and to the log file.

Related Information

Checking the Pad Configuration in *Design for Test in Encounter RTL Compiler.*

check_dft_rules

```
check_dft_rules [design] [-advanced]
     [-max_print_violations integer | > file]
     [-max_print_registers integer]
     [-max_print_fanin integer]
     [-dft_configuration_mode dft_config_mode_name]
     [-mbist [-interface_file_dirs string]]
```

Evaluates the design for DFT-readiness. Flip-flops that pass the DFT rule checks are later mapped to scan flip-flops during synthesis and included in a scan chain during scan connection. Flip-flops that fail the DFT rule checks and flip-flops marked with either a `dft_dont_scan` attribute or a `preserve` attribute, or flip-flops instantiated in lower-level blocks marked with a `preserve` attribute, are not mapped to scan flip-flops and are excluded from the scan chains. The DFT rule checker also analyzes the libraries and reports on the valid scan cells.

Table 11-1 lists the DFT rule violations that this command checks.
.

Table 11-1 Checking For and Auto-Fixing of DFT Rule Violations

DFT Rule Violation	Checked?	Auto-Fixed?
Uncontrollable asynchronous set or reset signals	Checked	Yes
Gated clocks and derived clocks	Checked	Yes
Flip-flop’s clock port connected to tied lines	Checked	Yes
Conflicting clock and asynchronous set or reset signals	Checked	No
Tristate contention for internal and external nets	Checked(*)	Yes
Same asynchronous set or reset data race conditions	Checked(*)	Yes
Clock and data race conditions	Checked(*)	No
Floating nets violations	Checked(*)	No
X-source violations	Checked(*)	Yes
Floating conditions	Not checked	No

Note: (*) indicates that the check is performed using the `-advanced` option.

To maximize fault coverage, you should try to fix any DFT rule violations, so that all flip-flops can be included in a scan chain. You can either modify the RTL or use the DFT fix capabilities using the `fix_dft_violations` command.

Tip

To include the RTL file name and line number at which the DFT violation occurred in the messages produced by `check_dft_rules`, set the `hdl_track_filename_row_col` root attribute to `true` before elaboration.

Table 11-2 lists the MBIST rules violations checked by the command. The command does not auto-fix detected MBIST rules violations.

Table 11-2 Checked MBIST Rule Violations

MBIST Rule
`Test_Control` is properly controlled at the MBIST engine pin via chip port
MBIST engine clock pins are properly controlled from chip ports
Interface files are consistent with JTAG instructions and MBIST logic inserted in the design

You can find the objects created by the `check_dft_rules` command in:

`/designs/`*`design`*`/dft/test_clock_domains`

The detected violations are placed in:

`/designs/`*`design`*`/dft/report/violation`

Options and Arguments

`-advanced`	Specifies to perform checking for tristate contention, same asynchronous set or reset data race conditions, clock and data race conditions, x-source generators, and floating nets violations.
`design`	Specifies the name of the top-level design to be checked. You should specify this name in case you have multiple top designs loaded. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dft_configuration_mode` *`dft_configuration_mode_name`*

Specifies the object name of the scan mode to be checked. A scan mode is defined using the define_dft dft_configuration_mode command.

Scan modes can be used to build the top-level scan chains with specific elements in different modes of operation (multi-mode), or when concatenating default scan chains into a single longer scan chain in a different scan mode of operation.

`file` Specifies the file to which to redirect the detailed output.

`-interface_file_dirs` *`dir1 dir2 ...`*`.`

Checks the content of MBIST interface files against the MBIST logic inserted into the design.

`-max_print_fanin` *`integer`*

Limits the number of pins and ports reported in the fanin cone for any violation.

Note: This option affects DFT checks performed for same asyncronous set or reset data race conditions, clock and data race conditions, and x-source generators.

Default: `20`

`-max_print_registers` *`integer`*

Limits the number of registers reported for any violation.

Default: `20`

`-max_print_violations` *`integer`*

Controls the number of DFT violations for which the details are printed to the screen and log file. Specify `-1` to write the details of all violations to the log file.

Default: `20`

`-mbist` Determines whether the MBIST logic inserted in the design satisfies the conditions necessary to control it from the chip tester accessible pins.

Note: The `-interface_file_dirs` option must be specified with this option.

Examples

- The following example defines the shift-enable signal and its active polarity, then runs the DFT rule checker. The output of the `check_dft_rules` command is written to the `DFT.rules` file. The return value of the command (2) corresponds to the number of violations found.

```
rc:/> define_dft shift_enable scan_en -active high
rc:/> check_dft_rules > DFT.rules
  Checking DFT rules for ’top’ module under ’muxed_scan’ style

  Checking DFT rules for clock pins
  ...
  Checking DFT rules for async. pins
  ...
Detected 2 DFT rule violation(s)
    ... see the log file for more details
Number of user specified non-Scan registers:   0
      Number of registers that fail DFT rules:   4
      Number of registers that pass DFT rules:   1
  Percentage of total registers that are scannable: 20%
rc:/> sh more DFT.rules

Checking DFT rules for ’top’ module under ’muxed_scan’ style
Processing techlib techlib_25
  Identified a valid scan cell ’SDFFHQX1’
         active clock edge:  rising
  Identified a valid scan cell ’SDFFHQX2’
         active clock edge:  rising
...
Identified 60 valid usable scan cells
Detected 2 DFT rule violation(s)
        Summary of check_dft_rules
        *************************
        Number of usable scan cells: 60
Clock Rule Violations:
---------------------
        Internally driven clock net: 1
            Tied constant clock net: 0
                 Undriven clock net: 0
        Conflicting async/clock net: 0
                    Misc. clock net: 0

Async. set/reset Rule Violations:
--------------------------------
        Internally driven async net: 1
              Tied active async net: 0
                 Undriven async net: 0
                    Misc. async net: 0

   Total number of DFT violations: 2
Clock Violation
# 0: internal or gated clock signal in module ’top’, net ’Iclk’, inst/pin ’g4/z’
(file: test3.v, line 11) [CLOCK-05]
  Effective fanin cone:
    clk
    en
Async Violation
# 1: async signal driven by a sequential element in module ’top’, net ’Iset’,
inst/pin 'Iset_reg/q' (file: test3.v, line 13) [ASYNC-05]
  Effective fanin cone:
    Iset_reg/q
        Violation # 0 affects  4 registers
        Violation # 1 affects  4 registers

        Note - a register may be violating multiple DFT rules
Total number of Test Clock Domains: 1
  DFT Test Clock Domain: clk
        Test Clock 'clk' (Positive edge) has   1 registers
Number of user specified non-Scan registers:   0
    Number of registers that fail DFT rules:   4
    Number of registers that pass DFT rules:   1
Percentage of total registers that are scannable: 20%
```

■ The following example shows tristate net checking with the `check_dft_rules -advanced` option.

```
rc:/> check_dft_rules -advanced
Checking DFT rules for 'test' module under 'muxed_scan' style
    Processing techlib tsmc_25 for muxed_scan scan cells
    Identified 60 valid usable scan cells
  Checking DFT rules for clock pins
Info    : Added DFT object. [DFT-100]
        : Added test clock domain 'clk'.
Info    : Added DFT object. [DFT-100]
        : Added test clock 'clk'.
  Checking DFT rules for async. pins
  Checking DFT rules for shift registers.
  Checking DFT rules for tristate nets.
  Checking DFT rules for clock data race conditions.
  Checking DFT rules for set reset data race conditions.
  Checking DFT rules for x-sources.
Detected 1 DFT rule violation(s)
        Summary of check_dft_rules
        **************************
        Number of usable scan cells: 60
Clock Rule Violations:
---------------------
          Internally driven clock net: 0
              Tied constant clock net: 0
                   Undriven clock net: 0
        Conflicting async & clock net: 0
                      Misc. clock net: 0

Async. set/reset Rule Violations:
--------------------------------
        Internally driven async net: 0
              Tied active async net: 0
                 Undriven async net: 0
                    Misc. async net: 0

Advanced DFT Rule Violations:
----------------------------------
         Tristate net contention violation: 1
        Potential race condition violation: 0
                       X-source violation: 0

   Total number of DFT violations: 1
```

```
Warning : DFT Tristate net contention Violation. [DFT-315]
        : # 0 <vid_0_tristate_net>: tristate net 'tbus' connected to pin
 'out_reg/d' potentially driven by conflicting values   [TRISTATE_NET-01]
 : To remove the net contention violation in scan-shift mode, either modify the
RTL, or use the '-tristate_net' option of the 'fix_dft_violations' command.

    Tristate net drivers:
      i_block2/g1/z
      i_block1/g1/z
      i_block3/g1/z

    Violations sorted by type and number of affected registers
    Note - a register may be violating multiple DFT rules.

        There are '1' tristate net violations.

---------------------
```

Related Information

Running the DFT Rule Checker in *Design for Test in Encounter RTL Compiler.*

Defining Scan Configuration Modes in *Design for Test in Encounter RTL Compiler*

Concatenating Scan Chains in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft abstract_segment on page 530
	define_dft dft_configuration_mode on page 538
	define_dft shift_enable on page 572
	define_dft test_clock on page 577
	define_dft test_mode on page 581
Affects these commands:	connect_scan_chains on page 523
	fix_dft_violations on page 586
	report dft_registers on page 648
	synthesize on page 294

Affected by these attributes:	dft_controllable
	dft_dont_scan
	dft_identify_test_signals
	dft_identify_top_level_test_clocks
	dft_scan_style
	(instance) preserve
	(subdesign) preserve
	dft_identify_xsource_violations_from_timing_models
Sets these attributes:	dft_status
	dft_violation
	type
	Violations Attributes

compress_scan_chains

```
compress_scan_chains  -ratio integer
     [-chains actual_scan_chain]...
     [-allow_shared_clock] [-auto_create] [-power_aware]
     [-decompressor {broadcast
        | xor [-spread_enable test_signal]}]
     [-master_control test_signal]
     [-compression_enable test_signal] [-target_period integer]
     [-jtag_control_instruction jtag_instruction]
        [-allow_multiple_jtag_control]]
     [ -compressor xor
        [-mask {wide1|wide2} [-mask_clock {port|pin}]
         [-mask_load test_signal] [-mask_enable test_signal]
         [-create_mask_or_misr_chain
          -mask_or_misr_sdi {pin|port}
          -mask_or_misr_sdo {pin|port} [-shared_output]]]
        [-mask_sharing_ratio integer
        [-apply_timing_constraints [-timing_mode_names mode_list]]
        [-write_timing_constraints file]
     | -compressor misr
        [-serial_misr_read]
           [-misr_observe test_signal]
        [-misr_clock {port|pin}]
        [{-misr_reset_enable test_signal] | [-misr_reset_clock test_signal]}]
        [-misr_read test_signal | -dont_exploit_bidi_scanio]
        [-misr_shift_enable test_signal]
        [-mask_sharing_ratio integer
        [-mask {wide0 | wide1 | wide2}
         [-mask_clock {port|pin}] [-mask_load test_signal]
         [-mask_enable test_signal_list]
         [-create_mask_or_misr_chain
          -mask_or_misr_sdi {pin|port}
          -mask_or_misr_sdo {pin|port} [-shared_output]]
     | -compressor hybrid
        [-serial_misr_read]
           [-misr_observe test_signal]
        [-misr_bypass test_signal]
        [-misr_clock {port|pin}]
        [{-misr_reset_enable test_signal] | [-misr_reset_clock test_signal]}]
        [-misr_shift_enable test_signal]
        [-mask {wide0 | wide1 | wide2}
        [-mask_sharing_ratio integer
         [-mask_clock {port|pin}] [-mask_load test_signal]
         [-mask_enable test_signal_list]
         [-create_mask_or_misr_chain
          -mask_or_misr_sdi {pin|port}
          -mask_or_misr_sdo {pin|port} [-shared_output]]]
      [-preview] [-inside instance] [design]
```

Adds decompression and compression logic to reduce the effective length of the actual scan chains.

Note: To use this command you need to have a license for the Encounter Test Architect tool.

Options and Arguments

`-allow_multiple_jtag_control`

When controlling the compression testmode from a JTAG macro, (that is, the `-jtag_control_instruction` option is specified), `compress_scan_chains` checks for the presence of other compression macros that are also JTAG controlled. RC-DFT does not automatically support such a configuration and therefore insertion of a second JTAG controlled compression macro is disallowed by default. Specify this option to bypass this check and insert additional JTAG controlled compression macros. When this option is specified, the tool assumes you will manually perform any additional stitching needed and will appropriately modify any files generated for Encounter Test.

Note: You must also specify the `-jtag_control_instruction` option with this option.

`-allow_shared_clock`

Allows the mask or `MISR` clock to be shared with an existing full-scan test clock.

Note: The mask or `MISR` clock can only be shared with a test clock if you added gating logic which prevents the scan flops from pulsing during the channel mask load or `MISR` reset sequences. Since functional clocks are typically used for scanning, this requirement means that the functional clocks must be gated during test.

Important

When specified with the `-auto_create` option, a `-mask_load` pin is automatically created. The mask_load pin must additionally be used to gate off the clock being shared. If this gating logic is not added, the mask loading or MISR reset procedure will corrupt the test data in the design.

`-apply_timing_constraints`

Applies timing constraints to the appropriate compression control signals to prevent the mapper from considering these paths for timing optimization.

Timing constraints will be applied in all user-specified timing modes.

Note: If your design has multiple timing modes but you did not specify the `-timing_mode_names` option to list the timing modes for which to write the constraints, no additional constraints are applied.

`-auto_create` Automatically creates the necessary test pins as top-level ports.

If you omitted any of the following options `-compression_enable`, `-spread_enable`, `-mask_clock`, `-mask_load`, `-mask_enable`, `-mask_or_misr_sdi`, `-mask_or_misr_sdo`, `-misr_clock`, `-misr_observe`, `-misr_reset_enable`, `-misr_read`,`-misr_bypass`, the appropriate ports will be created and named using the following format:

`prefixOption`

For example, *`prefix`*`compression_enable`, where *`prefix`* is the value of the `dft_prefix` root attribute.

`-chains` *`actual_scan_{chain...}`*

Specifies the names of the actual scan chains to be compressed.

By default all actual scan chains are compressed.

Note: When inserting `MISR` logic, all scan chains must eventually be compressed.

`-compression_enable` *`test_signal`*

Specifies the name of the test signal that enables configuring the actual scan chains in compression mode.

Note: If you do not specify the `-auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. If you request to build the compression logic with a master control signal (`-master_control`), the input port driving the compression enable signal can be an existing functional pin (specified through the `-shared_in` option of `define_dft test_mode`). If you do not specify the `-master_control` option, you must define the compression enable signal without the `-shared_in` option.

`-compressor {xor | misr| hybrid}`

Specifies the type of compression logic to be built:

- `xor` specifies to build an XOR-based compressor
- `misr` specifies to build a `MISR-based` compressor
- `hybrid` specifies to build a `MISR` compression with `MISR` bypass capability. Bypassing the MISR allows you to perform compression using just the XOR compressor.

Default: `xor`

`-create_mask_or_misr_chain`

Specifies to build a separate full-scan chain for the mask and `MISR` registers.

Note: To place the mask or `MISR` registers in a separate chain, an additional scan data input pin and scan data output pin are needed. You can either specify these pins using the `-mask_or_misr_sdi` and `-mask_or_misr_sdo` options, or make sure that the `-auto_create` option is specified.

`-decompressor {broadcast | xor}`

Specifies the type of decompression logic to be built:

- `xor` specifies to build an XOR-based spreader network in addition to the broadcast-based decompression logic
- `broadcast` specifies to build a broadcast-based decompression logic (simple scan fanout).

Default: `broadcast`

`*design*` Specifies the name of the top-level design whose scan chains must be compressed. You should specify this name in case you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dont_exploit_bidi_scanio`

Disables use of bidirectional scan I/O for a `MISR-based` compressor.

Note: This option is mutually exclusive with the `-misr_read` option.

`-inside` *`instance`* Specifies the instance in which to instantiate the compression logic.

By default, the compression logic is inserted as a hierarchical instance in the top-level of the design.

`-jtag_control_instruction` *`jtag_instruction`*

Specifies which JTAG instruction will be used to target the compression macro’s test data register.

`-mask {wide0| wide1 | wide2}`

Inserts scan channel masking logic of the specified type.

The masking types that can be used depend on the compressor type specified with the `-compressor` option.

By default, no masking logic is inserted.

Note: The syntax indicates which types are available for each of the compressor types.

`-mask_clock {`*`pin|port`*`}`

Specifies the clock that controls the mask registers.

Note: The input port associated with this option can be an existing functional pin. This clock cannot be shared with an existing full-scan test clock pin unless you also specify the `-allow_shared_clocks` option.

`-mask_enable` *`test_signal`*

Specifies the name of the test signal that controls whether mask bits should be applied during the current scan cycle.

For `wide2` masking, two mask enable signals must be specified.

Note: If you do not specify the `-auto_create` option, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-mask_load` *`test_signal`*

Specifies the name of the test signal that enables loading of the mask data into the mask data registers.

Note: If the mask_clock is dedicated (that is, is only used for mask register loading), this signal is not needed. If this signal is shared (is used to clock the MISR or other logic in the circuit), this signal is needed to gate non mask load clock pulses from corrupting the mask registers. If the `mask_clock` is shared with other logic, you can use this signal to protect the shared logic from corruption during the mask load sequence.

`-mask_or_misr_sdi {`*`pin|port`*`}`

Specifies the scan data input pin or port of the mask or `MISR` chain.

Note: The input port associated with this option can be an existing functional pin.

`-mask_or_misr_sdo {`*`pin`*`|`*`port`*`}`

Specifies the scan data output pin or port of the mask or `MISR` chain.

Note: If the output port associated with this option is an existing functional pin, you must specify the `-shared_out` option.

`-mask_sharing_ratio` *`integer`*

Specifies the number of internal scan channels sharing a mask register. The specified integer may not exceed the value specified for the compression ratio.

Note: This option is only valid with `wide1` and `wide2` masking.

`-master_control` *`test_signal`*

Specifies the master control signal that gates the compression enable signal used for compression.

Note: This test signal must be dedicated for test and must have been defined using the `define_dft test_mode` command.

`-misr_bypass` *`test_signal`*

Specifies the test signal used to bypass the `MISR-based` logic. This test signal is required in hybrid compression mode.

Note: If you do not specify the `-auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-misr_clock {`*`pin`*`|`*`port`*`}`

Specifies the clock that controls the `MISR` registers.

Note: This input port cannot be shared with an existing full-scan test clock unless it is only used to accumulate the MISR signature or if the `-allow_shared_clocks` option is specified. The `-misr_clock` option is only used to accumulate the MISR signature if there are separate `-mask_clock` and `-misr_reset_clock` signals specified.

`-misr_reset_clock` *`test_signal`*

Specifies a separate dedicated test signal that is used to asynchronously reset the MISR.

Note: This option is mutually exclusive with the `-misr_reset_enable` option.

`-misr_observe` *`test_signal`*

Specifies the test signal used to select Serial MISR Read. This is required when the `-serial_misr_read` option is specified unless `-auto_create` or `-jtag_control_instruction` is also specified.

Note: You must also specify the `-serial_misr_read` option with this option.

`-misr_read` *`test_signal`*

Specifies the test signal to configure any bidirectional scan I/O pads for MISR compression.

This option is mutually exclusive with the `-dont_exploit_bidi_scanio` option. Using scan I/O bidirectionally during MISR compression is only available with the -compressor misr option.   When using the `-compressor hybrid` option, all scan I/O are used unidirectionally.

`-misr_reset_enable` *`test_signal`*

Specifies the test signal used to reset the `MISR` registers.

Notes:

- This option is mutually exclusive with the `-misr_reset_clock` option.
- If you do not specify the `-auto_create` option, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-misr_shift_enable` *`test_signal`*

Specifies the test signal used to enable MISR accumulation during scan shifting.When this signal is de-asserted, the contents of the MISR register will not change.

Note: If this option is omitted, the default shift-enable test signal for the design is used. If this option is specified, you must have defined this test signal using the `define_dft shift_enable` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft shift_enable` command).

`-power_aware`	Ensures that the mux logic added during compression obeys the power-domain boundaries defined using CPF.
`-preview`	Reports the requested ratio, the maximum original scan chain length, the maximum subchain length, and the number of internal scan channels that would be created without making modifications to the netlist. Use this option to verify your compression architecture prior to inserting the compression logic.
`-ratio` *`integer`*	Controls the maximum length of the internal scan channels. The length of the internal scan channels is determined by dividing the longest actual scan chain length by the ratio.
`-serial_misr_read`	Specifies to include support for reading MISR bits serially through the scan data pins.

`-spread_enable` *`test_signal`*

Specifies the name of the test signal that enables applying the input test data to an XOR-based spreader network.

Use this option when `-decompressor` is set to `xor`.

Note: If you do not specify the `auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-shared_output`	Specifies that the scan data output port of the dedicated mask or `MISR` chain must be shared with a functional port. The tool creates the additional logic required to share the port.

`-target_period` *`integer`*

Specifies a target clock period (in picoseconds) used to optimize the compression macro. If the value zero (`0`) is specified, synthesis is performed with a low effort compile, and without applying external (input/output delay) constraints.

Default: value for `test_clock` period of the scan chains being compressed

`-timing_mode_names` *`mode_list`*

Specifies the timing modes for which to generate the additional timing constraints that apply to the compression control signals.

The timing modes are taken into account when you specify the `-apply_timing_constraints` and `-write_timing_constraints` options.

Note: This applies only to multi-mode designs. Modes are created with the `create_mode` command.

`-write_timing_constraints` *`file`*

Specifies the file to which to write the timing constraints applied to the appropriate compression control signals to prevent the mapper from considering these paths for timing optimization. The timing constraints are not written if you specify the `-preview` option.

Timing constraints will be applied in all user-specified timing modes.

Note: If your design has multiple timing modes but you did not specify the `-timing_mode_names` option to list the timing modes for which to write the constraints, constraints for all modes are written to the file.

Examples

- The following command requests XOR-based compression logic without masking, and requests creation of the necessary ports for the required test signals. In this case only a compression enable signal is required and thus one test port is created.

```
rc:/> compress_scan_chains -ratio 5 -compressor xor -auto_create
Will create a test port for ’compression_enable’
Checking out license ’Encounter_Test_Architect’... (1 seconds elapsed)
...
```

- The following command requests XOR-based compression logic with masking logic of type `wide1`, and requests creation of the necessary ports for the required test signals. In this case three extra test signals are required for the masking logic and thus four test ports are created.

```
rc:/> compress_scan_chains -ratio 5 -compressor xor -mask wide1 -auto_create
Will create a test port for ’compression_enable’
Will create a test port for ’mask_load’
Will create a test port for ’mask_enable’
Will create a test port for ’mask_clock’
Checking out license ’Encounter_Test_Architect’... (1 seconds elapsed)
....
```

- The following command requests an XOR-based compression with masking logic of type `wide1`, with decompression logic of type `xor`, and requests creation of the necessary ports for the required test signals. Compared to the second example, one extra test signal— the `spread_enable` signal—is required for the decompression logic, and thus five test ports are created.

```
rc:/> compress -ratio 5 -compressor xor -decompressor xor -mask wide1 \
-auto_create
Will create a test port for ’compression_enable’
Will create a test port for ’spread_enable’
Will create a test port for ’mask_load’
Will create a test port for ’mask_enable’
Will create a test port for ’mask_clock’
Checking out license ’Encounter_Test_Architect’... (2 seconds elapsed)
...
```

- The following command requests a `MISR`-based compression with masking logic of type `wide1`, and requests creation of the necessary ports for the required test signals. Compared the two second example, one extra test signal—the `misr_reset_enable` signal—is required to reset the `MISR` registers, and thus five test ports are created.

```
rc:/> compress -ratio 5 -compressor misr -mask wide1 -auto_create
Will create a test port for ’compression_enable’
Will create a test port for ’misr_reset_enable’.
Will create a test port for ’misr_clock’
Info - Defaulting the misr shift enable signal to the default shift_enable
/designs/test/dft/test_signals/SE
Will create a test port for ’mask_load’
Will create a test port for ’mask_enable’
Info - will share the ’misr_clock’ and the ’mask_clock’
....
```

Related Information

Compressing Scan Chains in *Design for Test in Encounter RTL Compiler*

Manually Inserting a Scan Compression Macro in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft jtag_instruction on page 543
	define_dft shift_enable on page 572
	define_dft test_mode on page 581
Affected by these commands:	connect_scan_chains on page 523
Affects this command:	report dft_chains on page 647
Sets these attributes:	compressed
	dft_compression_signal
	dft_mask_clock
	dft_misr_clock
	type

concat_scan_chains

```
concat_scan_chains [-preview]
     -chains actual_scan_chain(s)
     [-dft_configuration_mode dft_config_mode_name]
     -name string [design]
```

Inserts muxing logic into the scan path of actual scan chains such that the scan chains are concatenated to become a single, longer scan chain in a specific test mode of operation.

Note: An actual scan chain may be specified only once per test mode, that is, multiple configurations of the same scan chain in the same test mode are disallowed.

`-chains` *`scan_chain(s)`*

Names of the actual scan chains to be concatenated; where the scan chains are concatenated in the specified order.

`design` Specifies the name of the top-level design for which to concatenate the scan chains. Specify this name if you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dft_configuration_mode` *`dft_configuration_mode_name`*

Specifies the object name of the scan mode used to concatenate the actual scan chains.

`-name` *`string`* Name of the concatenated chain.

`-preview` Reports how the actual scan chains would be concatenated, but does not perform the concatenation.

Related Information

Concatenating Scan Chains in *Design for Test in Encounter RTL Compiler*

Controlling Scan Configuration in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft dft_configuration_mode on page 538
	define_dft shift_enable on page 572
	define_dft test_mode on page 581
	set_compatible_test_clocks on page 653

Affected by this command:	check_dft_rules on page 501
Affects this command:	report dft_chains on page 647
	write_dft_abstract_model on page 666
	write_scandef on page 693
Sets these attributes:	Actual Scan Chain attributes
	Actual Scan Segment attributes
Affected by these attributes:	dft_lockup_element_type
	dft_mix_clock_edges_in_scan_chains

configure_pad_dft

```
configure_pad_dft -mode {input | output | tristate}
     -test_control test_signal {pin|port}
```

Inserts the required logic to configure the data direction control for a bidirectional or tristate pad during test mode.

Note: This command can configure a generic pad.

Options and Arguments

`-mode {input | output | tristate}`

Specifies in which mode the pad must be configured in test mode.

`input`	Specifies to configure the pad in input mode.
`output`	Specifies to configure the pad in output mode.
`tristate`	Specifies to disable the pad.

`{`*`pin`*`|`*`port`*`}` Specifies the pin or top-level port that is connected to the I/O pad that the RC-DFT engine needs to configure.

`-test_control` *`test_signal`*

Specifies the test signal to use to control the pad.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

Related Information

Affected by these constraints:	define_dft shift_enable on page 572
	define_dft test_mode on page 581
Affects these commands:	check_dft_rules on page 501
	connect_scan_chains on page 523

connect_scan_chains

```
connect_scan_chains [design]
     [-preview] [-auto_create_chains]
     [-dont_exceed_min_number_of_scan_chains]
     [-incremental] [-chains chain_list]
     [-dft_configuration_mode dft_config_mode_name]
     [-elements element_list]
     [-pack | -create_empty_chains]
     [-physical ]
     [-power_domain power_domain_list] [-update_placement]
```

Configures and connects scan flip-flops which pass the DFT rule checks into scan chains. This command works at the current level of the hierarchy and all lower hierarchies instantiated in this module. The design must be mapped to the target library before connecting scan chains in a design.

The command returns the number of scan chains that the scan configuration engine creates (or would create if you use the `-preview` option).

You can find the objects created by the `connect_scan_chains` command in:

```
/designs/design/dft/actual_scan_chains
/designs/design/dft/actual_scan_segments
```

Options and Arguments

`-auto_create_chains`	Allows the scan configuration engine to add new chains that are not defined through a `define_dft scan_chain` constraint. Without this option, the scan configuration engine reports an error if it needs more scan chains than have been defined with the `define_dft scan_chain` command.
`-chains` *`chain_list`*	Connects only the specified user-defined chain names. If the list is empty, none of the user-defined chains can be connected at this time. New chains are created if you specify the `-auto_create_chains` option. The specified user-defined chains must have been defined using a `define_dft scan_chain` constraint. If omitted, all user-defined chains can be connected.

`-create_empty_chains`

Allows the scan configuration engine to create empty scan chains by making a direct connection from their scan data input to their scan data output if the number of scan chains to be configured is less than the minimum number of scan chains required in the design.

Note: Do not use this option when configuring scan chains to be used as internal scan channels that are loaded and unloaded using on-chip compression logic.

If this option is not specified, the scan configuration engine will move scan flops between compatible chains to satisfy the minimum number of scan chains requirement. This can result in configured scan chains having a sequential depth of one element.

`-dft_configuration_mode` *`dft_configuration_mode_name`*

Specifies the object name of the scan mode in which to build the scan chains.

`-dont_exceed_min_number_of_scan_chains`

Specifies to use the exact same number of scan chains as specified by `dft_min_number_of_scan_chains` attribute when building the scan chain.

`design`

Specifies the name of the top-level design to be checked. You should specify this name in case you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-elements` *`element_list`*

Considers only the specified elements for scan chain connection. An element can be a flip-flop, segment, or a hierarchical instance.

If you specify a hierarchical instance, all flops in this hierarchical instance that pass the DFT rule checker and that are mapped to scan for DFT, will be added to the chains.

If some of the scan flops in a hierarchical instance belong to a segment that crosses the boundary of this instance, these scan flops will only be connected if the remaining elements of the segment are also specified with the `-elements` option—either directly or indirectly through another hierarchical instance.

Note: If you specify this option with the `-power_domain` option, the specified elements must belong to the specified power domains.

`-incremental`

Adds new chains in incremental mode, without changing already connected scan chains stored in `/designs/design/dft/report/actual_scan_chains`

Do not use user-defined chains with the same names as the `actual_scan_chains`.

`-pack`

Packs the scan chains to their maximum limit instead of balancing the chains (that is, attempting to create chains with similar lengths).

You can specify a chain-specific constraint using the `-max_length` option of the `define_dft scan_chain` command or a global constraint by setting the value of the `dft_max_length_of_scan_chains` attribute.

`-physical`

Specifies to use the placement locations of the scan flops to connect the scan chains.

The placement information is obtained from the DEF file read in with the `read_def` command.

`-power_domain` *`power_domain_list`*

Considers only the scan flops that belong to the specified power domain(s) for scan chain connection.

`-preview`	Reports how the scan chains will be connected, but makes no modifications to the netlist. Use this option to verify your scan-chain architecture prior to connecting the scan chains.
`-update_placement`	Specifies to update the placement of the DFT logic that is added during the scan chain connection process.

Related Information

Controlling Scan Configuration in *Design for Test in Encounter RTL Compiler*

Library-Domain Aware Scan Chain Configuration in *Design for Test in Encounter RTL Compiler*

Power-Domain Aware Scan Chain Configuration in *Design for Test in Encounter RTL Compiler*

Physical Scan Chain Synthesis in *Design for Test in Encounter RTL Compiler*

Related commands:	write_compression_macro on page 662
Affected by these constraints:	define_dft abstract_segment on page 530
	define_dft dft_configuration_mode on page 538
	define_dft fixed_segment on page 540
	define_dft floating_segment on page 542
	define_dft preserved_segment on page 557
	define_dft scan_chain on page 560
	define_dft scan_clock_a on page 566
	define_dft scan_clock_b on page 569
	define_dft shift_enable on page 572
	define_dft shift_register_segment on page 575
	set_compatible_test_clocks on page 653
Affected by these commands:	check_dft_rules on page 501
	fix_dft_violations on page 586
	synthesize on page 294

Affects this command:	concat_scan_chains on page 520
	report dft_chains on page 647
Sets these attributes:	Actual Scan Chain attributes
	Actual Scan Segment attributes
Affected by these attributes:	dft_lockup_element_type
	dft_max_length_of_scan_chains
	dft_min_number_of_scan_chains
	dft_mix_clock_edges_in_scan_chains
	dft_prefix
	dft_scan_map_mode

define_dft

```
define_dft {abstract_segment | fixed_segment
     | boundary_scan_segment | dft_configuration_mode | jtag_macro
     | floating_segment | mbist_clock | preserved_segment | scan_chain
     | scan_clock_a | scan_clock_b | shift_enable
     | shift_register_segment | test_clock | test_mode }
```

Defines a DFT object. A DFT object can be a test signal, scan segment, or scan chain.

Options and Arguments

`abstract_segment`	Defines an abstract scan-chain segment object.
`boundary_scan_segment`	Defines a boundary-scan segment object.
`dft_configuration_mode`	Defines a scan mode for DFT configuration purposes.
`fixed_segment`	Defines a fixed scan-chain segment object.
`floating_segment`	Defines a floating scan-chain segment object.
`jtag_macro`	Defines a pre-instantiated third-party JTAG Macro.
`jtag_instruction`	Defines a user-defined instruction that is serially loaded into a boundary scan device.
`jtag_instruction_register`	Customizes the instruction register to allow adding user-defined instructions.
`mbist_clock`	Defines an MBIST clock object.
`preserved_segment`	Defines a preserved scan-chain segment object.
`scan_chain`	Defines a scan chain.
`scan_clock_a`	Defines the scan clock of the master latch for the LSSD scan style.
`scan_clock_b`	Defines the scan clock of the slave latch for the LSSD scan style.
`shift_enable`	Defines a `test_signal` object of type `shift_enable`.
`shift_register_segment`	Defines a shift register scan-chain segment object.
`test_clock`	Defines a test clock object.
`test_mode`	Defines a `test_signal` object of type `test_mode`

Related Information

Related commands:	define_dft abstract_segment on page 530
	define_dft boundary_scan_segment on page 535
	define_dft dft_configuration_mode on page 538
	define_dft fixed_segment on page 540
	define_dft floating_segment on page 542
	define_dft jtag_instruction on page 543
	define_dft jtag_instruction_register on page 547
	define_dft jtag_macro on page 549
	define_dft mbist_clock on page 554
	define_dft preserved_segment on page 557
	define_dft scan_chain on page 560
	define_dft scan_clock_a on page 566
	define_dft scan_clock_b on page 569
	define_dft shift_enable on page 572
	define_dft shift_register_segment on page 575
	define_dft test_clock on page 577
	define_dft test_mode on page 581

define_dft abstract_segment

```
define_dft abstract_segment [-name segment_name]
     {-module subdesign|-instance instance|-libcell cell}
     -sdi subport -sdo subport
     -clock_port subport [-rise|-fall] [-off_state {high|low}]
     [ -tail_clock_port subport
       [-tail_edge_rise | -tail_edge_fall]
       [-tail_clock_off_state {high|low}] ]
     { { -shift_enable_port subport -active {high|low}
       | -connected_shift_enable }
     | { -scan_clock_a_port subport -scan_clock_b_port subport
       | -connected_scan_clock_a -connected_scan_clock_b } }
     [ -test_mode_port subport
       -test_mode_active {low|high} ]...
     -length integer [-skew_safe]
     [-dft_configuration_mode dft_config_mode_name]
```

Defines an abstract segment. An abstract segment can be defined for objects of type blackbox, logic abstract module, or libcell timing model.

An abstract segment is a user-specified scan segment used at the next level of integration to define the sets of scan chains previously created for the object.

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft abstract_segment` constraints in:

`/designs/top_design/dft/scan_segments`

Options and Arguments

`-active {low|high}` Specifies the active value for the shift-enable port.

`-clock_port subport`

Specifies the clock port—at the boundary of the blackbox or logic abstract module—driving the flip-flops at the head of the segment.

`-connected_scan_clock_a (-connected_scan_clock_b)`

Indicates that the `scan_clock_a` (`scan_clock_b`) port of the module boundary is driven by external logic (preconnected). The external logic connected to the `scan_clock_a` (`scan_clock_b`) pin of the module will not be modified by the scan configuration engine.

Note: This option applies only for the clocked LSSD scan style.

`-connected_shift_enable`	
	Indicates that the shift enable port of the module boundary is driven by external logic (preconnected) or that the shift enable signal is internally generated within the module boundary. In either case, the external logic connected to the shift enable pin of the module, or the internal logic driving the shift enable pins of the flip-flops in the module will not be modified by the scan configuration engine. This option cannot be specified together with the `-shift_enable` option.
`-dft_configuration_mode` *`dft_configuration_mode_name`*	
	Specifies in which scan mode the abstract segment will be connected in the scan chains.
`-instance` *`instance`*	Specifies the instance name of the module for which the abstract segment is defined.
`-length` *`integer`*	Specifies the length of the abstract segment.
`-libcell` *`cell`*	Specifies the library cell for which the abstract segment is defined. This option applies to library cells that are implemented as timing models and whose description includes the relevant test-related pins (such as scan data input and output, clock, shift-enable) to infer the scan chain architecture.
`-module` *`subdesign`*	Specifies the subdesign (module) to which the element belongs.
`-name` *`segment_name`*	Defines a name for the segment that you can use to reference in the `define_dft scan_chain` constraint.
`-off_state {high|low}`	
	Specifies the off state of the system clock specified through the `-clock_port` option. **Note:** This option applies only to the clocked LSSD scan style.
`-rise | -fall`	Specifies the active edge of the clock specified through the `-clock_port` option. *Default*: `-rise`

`-scan_clock_a_port (-scan_clock_b_port)` *`subport`*

Specifies the `scan_clock_a` (`scan_clock_b`) port at the boundary of the blackbox or logic abstract module to which the segment belongs. Specify this option to have the `connect_scan_chains` command make the connection from the top-level `scan_clock_a` (`scan_clock_b`) signals to the `scan_clock_a` (`scan_clock_b)` port of the module.

Note: This option applies only for the clocked LSSD scan style.

`-sdi (-sdo)`

Specifies the scan data input (scan data output) of the segment.

- For a segment in a blackbox, specify a subport (port of the blackbox or logic abstract module).
- For a segment defined for a libcell, specify a pin of the libcell.

`-shift_enable_port` *`subport`*

Specifies the shift enable port at the boundary of the blackbox or logic abstract module to which the segment belongs. Specify this option if you want the `connect_scan_chains` command to make the connection from the top-level shift-enable signals to the shift-enable ports of the modules.

This option cannot be specified together with the `-connected_shift_enable` option.

`-skew_safe`

Indicates whether the abstract segment has a data lockup element connected at the end of its scan chain.

`-tail_clock_off_state {high|low}`

Specifies the off state of the system clock specified through the `-tail_clock_port` option.

Note: This option applies only to the clocked LSSD scan style.

`-tail_clock_port` *`port`*

Specifies the clock port—at the boundary of the blackbox or logic abstract module—driving the flip-flops at the tail of the segment. This option is only required if the clock used at the tail of the abstract segment is different from the clock specified through the `-clock_port` option.

`-tail_edge_rise | -tail_edge_fall`

Specifies the active edge of the clock specified through the `-tail_clock_port` option.

Default: `-tail_edge_rise`

`-test_mode_active {low | high}`

Specifies the active value for the test-mode port.

This option should immediately follow the corresponding `-test_mode_port` option.

`-test_mode_port` *`subport`*

Specifies the test mode port at the boundary of the blackbox module to which the segment belongs.

Specify this option when the block-level design includes test-mode activated logic.

When the test-mode signals are specified, the propagated values of the top-level test-mode signals must match the expected block-level test-mode values and the segment must also pass the clock-controllability rule checks in order for the segment to be included into a top-level scan chain.

Note: The tool does not make connections to the test-mode ports of the block-level design. The connections should already exist in the netlist.

Examples

- The following example defines an abstract segment with length 3 in blackbox module b. The clock driving the flip-flops at the tail of the segment is the same as the clock driving the first elements in the segment.

```
rc:/> define_dft abstract_segment -name a1 -module b -length 3 \
-sdi {p4[0]} -sdo {p5[0]} -shift_enable {p6[0]} -active high -clock p3 -rise
```

- The following example defines an abstract segment in a timing model reference `COMBELEM` with length 20.

```
rc:/> define_dft abstract_segment -name combElem_seg -libcell COMBELEM \
-sdi A -sdo Z -shift_enable_port B -active hi -clock_port D2 -rise -length 20
```

- The following example defines an abstract segment ABS with length 10 in instance `o1`. The same clock `clk` with active rising edge is used for all flip-flops of the segment.

```
define_dft abstract_segment -instance o1  -sdi sdi -sdo sdo \
-shift_enable_port se -active hi -clock_port clk -rise -length 10 -name ABS
```

Related Information

Defining Abstract Segments in *Design for Test in Encounter RTL Compiler*

Using Abstract Segments in *Design for Test in Encounter RTL Compiler*

Affects this constraint:	define_dft dft_configuration_mode on page 538
	define_dft scan_chain on page 560
Affects these commands:	check_dft_rules on page 501
	connect_scan_chains on page 523
	report dft_chains on page 647
Sets these attributes:	Scan Segment Attributes

define_dft boundary_scan_segment

```
define_dft boundary_scan_segment [-name segment_name]
     {-module subdesign | -instance instance | -libcell cell}
     {-bsdl_string string | -bsdl_file file}
     [-differential_pair {positive_leg_pin_name negative_leg_pin_name}]
     [-mode_a mode_a_pin]... [-mode_b mode_b_pin]... [-mode_c mode_c_pin]...
     [-highz highz_pin] -tdi tdi_pin -tdo tdo_pin [-clockdr clockdr_pin]
     [-capture_dr capturedr_pin] [-updatedr updatedr_pin]
     [-shiftdr shiftdr_pin] [-index bsr_position]
```

Defines a boundary scan segment with its associated pins for connection along the TDI-TDO path in the boundary scan register, connection to the `JTAG_Macro`, and optionally defines its position in the boundary scan register.

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft boundary_scan_segment` command in:

`/designs/`*`top_design`*`/dft/boundary_scan_segments`

Options and Arguments

`-bsdl_file` *`file`* Specifies the file containing the BSDL abstract string for the boundary scan segment.

`-bsdl_string` *`string`*

Specifies the BSDL abstract string for the boundary scan segment.

`-capturedr` *`capturedr_pin`*

Specifies the name of the `CAPTURE_DR` pin on the boundary scan segment.

`-clockdr` *`clockdr_pin`*

Specifies the name of the `CLOCK_DR` pin on the boundary scan segment.

`-differential_pair {`*`positive_leg_pin_name negative_leg_pin_name`*`}`

Specifies the differential pair in the form of a positive leg pin name and negative leg pin name on the boundary scan segment.

`-highz` *`highz_pin`* Specifies the name of the `HIGHZ` pin on the boundary scan segment

`-index` *`bsr_position`*	
	Specifies the relative position of the boundary scan segment in the BSR. Specify an integer value of zero or greater.
`-instance` *`instance`*	Specifies the instance name of the module for which the boundary scan segment is defined.
`-libcell` *`cell`*	Specifies the library cell for which the boundary scan segment is defined. This option applies to library cells that are implemented as timing models and whose description includes the `JTAG_Macro` related pins specified on the command line when defining the boundary scan segment.
`-mode_a` *`mode_a_pin`*	
	Specifies the name of the `MODE_A` pin on the boundary scan segment
`-mode_b` *`mode_b_pin`*	
	Specifies the name of the `MODE_B` pin on the boundary scan segment
`-mode_c` *`mode_c_pin`*	
	Specifies the name of the `MODEC` pin on the boundary scan segment
`-module` *`subdesign`*	Specifies the subdesign (module) to which the element belongs.
`-name` *`segment_name`*	Defines a name for the boundary scan segment.
`-shiftdr` *`shiftdr_pin`*	
	Specifies the name of the `SHIFT_DR` pin on the boundary scan segment.
`-tdi` *`tdi_pin`*	Required. Specifies the name of the `TDI` pin on the boundary scan segment.
`-tdo` *`tdo_pin`*	Required. Specifies the name of the `TDO` pin on the boundary scan segment.
`-updatedr` *`updatedr_pin`*	
	Specifies the name of the `UPDATE_DR` pin on the boundary scan segment.

Example

- The following example defines an boundary scan segment using a set of differential port pairs.

  ```
  rc:/> define_dft boundary_scan_segment-instance i_pads \
  -bsdl_file pads_bcell.abstract -mode_a MODE_A -mode_b MODE_B -mode_c \
  MODE_C -highz HIGHZ -tdi TDI -tdo TDO -clockdr CLOCKDR -updatedr UPDATEDR\
  -shiftdr SHIFTDR -index 4 \
  -differential_pair {in1 in2} \
  -differential_pair {out1 out2}
  ```

- The following example defines a boundary scan segment with three `mode_a`, two `mode_b`, and one `mode_c` pins where:
  - Each `mode_a` pin on the boundary-scan segment will be connected to the `JTAG_MACRO` `JTAG_INSTRUCTION_DECODE_MODE_A` output pin.
  - Each `mode_b` pin on the boundary-scan segment will be connected to the `JTAG_MACRO` `JTAG_INSTRUCTION_DECODE_MODE_B` output pin.
  - Each `mode_c` pin on the boundary-scan segment will be connected to the `JTAG_MACRO` `JTAG_INSTRUCTION_DECODE_MODE_C` output pin.

  ```
  rc:/> define_dft boundary_scan_segment-instance i_pads \
  -bsdl_file pads_bcell.abstract -mode_a MODE_A1 -mode_a MODE_A2 \
  -mode_a MODE_A3 -mode_b MODE_B1 -mode_b MODE_B2 -mode_c MODE_C \
  -highz HIGHZ -tdi TDI -tdo TDO -clockdr CLOCKDR \
  -updatedr UPDATEDR -shiftdr SHIFTDR -index 4 -differential_pair {in1 in2}\
  -differential_pair {out1 out2}
  ```

Related Information

Defining Boundary Scan Segments in *Design for Test in Encounter RTL Compiler*

Inserting Boundary Scan Logic in *Design for Test in Encounter RTL Compiler*

Affects these commands:	insert_dft boundary_scan on page 607
	write_bsdl on page 659
Sets this attribute:	bcell_segmen
	differential

define_dft dft_configuration_mode

```
define_dft dft_configuration_mode
     [-name scan_mode_name]
     [-mode_enable_high test_signal]...
     [-mode_enable_low test_signal]...
     [-jtag_instruction jtag_instruction]
     [design]
```

Defines a scan mode. Scan modes can be used to build the top-level scan chains with specific elements in different modes of operation (multi-mode), or when concatenating default scan chains into a single longer scan chain in a different mode of operation.

You can find the objects created by the `define_dft dft_configuration_mode` command in:

`/designs/`*`top_design`*`/dft/dft_configuration_modes`

`design`	Specifies the name of the top-level design for which the scan mode is defined. Specify this name if you have multiple top designs loaded. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-jtag_instruction` *`jtag_instruction_name`*	Specifies the JTAG instruction which controls the scan chains in the current mode.
`-mode_enable_high {`*`test_signal...`*`}`	Name of the test signal(s) set to a `logic_1` value that controls scan chains in the current mode.
`-mode_enable_low {`*`test_signal...`*`}`	Name of the test signal(s) set to a `logic_0` value that controls the scan chains in the current mode.
`-name`	Name of the scan mode.

Example

The following example defines scan mode `scanModeA` whose `pin1` is specified to have an active high logic value and `pin2` is specified to have an active low logic value in this mode of operation:

```
define_dft dft_configuration_mode -name scanModeA design\
-mode_enable_high pin1
-mode_enable_low pin2
```

Related Information

Defining Scan Chain Configuration Modes in *Design for Test in Encounter RTL Compiler*

JTAG-Controlled Scan Modes in *Design for Test in Encounter RTL Compiler*

Affects these commands:	report dft_chains on page 356
	check_dft_rules on page 501
	concat_scan_chains on page 520
	connect_scan_chains on page 523
	define_dft abstract_segment on page 530
	write_atpg on page 656
	write_dft_abstract_model on page 666
	write_et_atpg on page 669
	write_et_bsv on page 674
	write_et_mbist on page 682
	write_et_rrfa on page 687
	write_scandef on page 693
Affects these attributes:	DFT Configuration Mode Attributes

define_dft fixed_segment

```
define_dft fixed_segment [-name segment_name]
     {pin|port|instance|segment_name} ...
```

Defines a fixed segment. In a fixed segment, the elements will be connected in the specified order during scan chain connection; they cannot be reordered by a physical scan reordering tool.

A fixed segment is a user-specified scan segment which can be associated with either

- A user-defined top-level chain—created using the `define_dft scan_chain` command
- A tool-created scan chain—created using the `connect_scan_chains` command

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft fixed_segment` constraints in:

`/designs/top_design/dft/scan_segments`

Options and Arguments

`{pin|port|instance|segment_name}`

Specifies an element in the scan segment being defined. List the elements in the order they should appear in the scan chain (in shift order, that is, left-most corresponds to first bit shifted-in). An element can be hierarchical pin, a port, a flip-flop instance, a combinational gate, or a scan segment.

Note: If the segment goes through a multi-input/output combinational gate, you must indicate the scan path through the gate by specifying its input and output pin as two separate consecutive elements.

`-name segment_name` Defines a name for the segment that you can use to reference in the `define_dft scan_chain` constraint.

Examples

- The following example defines the fixed segment used in the example for define_dft scan_chain on page 560.

  ```
  define_dft fixed_segment -name segBody *seq/out_reg_1 *seq/out_reg_3
  ```

- The following example defines a fixed segment that contains two combinational components, four sequential registers, a scan abstract segment, and a combinational component endpoint.

```
define_dft fixed_segment -name fixedSeg \
    i_core/i_anor1/B0 i_core/i_anor1/Y i_core/i_anor2/B0 i_core/i_anor2/Y \
    i_core/i_flop11 i_core/i_flop22 i_core/i_flop33 i_core/i_flop44 \
    absSeg \
    i_core/bufToAnchorSeg/A i_core/bufToAnchorSeg/Y
```

Related Information

Creating Head, Body, and Tail Segments in *Design for Test in Encounter RTL Compiler*

Affects this constraint:	define_dft scan_chain on page 560
Affects these commands:	connect_scan_chains on page 523
	report dft_chains on page 647
Sets these attributes:	Scan Segment Attributes

define_dft floating_segment

```
define_dft floating_segment [-name segment_name]
     instance|segment_name} ...
```

Defines a floating segment. In a floating segment, the order of the elements can be changed during scan configuration and by a physical scan reordering tool.

A floating segment is a user-specified scan segment which can be associated with either

- A user-defined top-level chain—created using the `define_dft scan_chain` command
- A tool-created scan chain—created using the `connect_scan_chains` command

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft floating_segment` constraints in:

`/designs/top_design/dft/scan_segments`

Options and Arguments

`-name` *`segment_name`*	Defines a name for the segment that you can use to reference in the `define_dft scan_chain` constraint.

Examples

- The following example defines the floating segments used in the example for define_dft scan_chain on page 560.

```
rc:/designs/test> define_dft floating_segment -name segHead \
*seq/out_reg_4 *seq/out_reg_5
rc:/designs/test> define_dft floating_segment -name segTail *seq/out_reg_0
```

Related Information

Creating Head, Body, and Tail Segments in *Design for Test in Encounter RTL Compiler*

Affects this constraint:	define_dft scan_chain on page 560
Affects these commands:	connect_scan_chains on page 523
	report dft_chains on page 647
Sets these attributes:	Scan Segment Attributes

define_dft jtag_instruction

```
define_dft jtag_instruction -name string -opcode string
     [-register string] [-length integer]
     [-register_tdi {pin|port}] [-register_tdo {pin|port}]
     [-register_shiftdr {pin|port}] [-register_shiftdr_inverted]
     [-register_reset_inverted] [-register_capturedr {pin|port}]
     [-register_clockdr {pin|port}] [-register_updatedr {pin|port}]
     [-register_tck {pin|port}] [-register_reset {pin|port}]
     [-register_runidle {pin|port}] [-register_decode {pin|port}]
     [-capture string]
      [-tap_tdo {pin|port}] [-tap_decode {pin|port}] [-private]
     [-design design]
```

Defines a user-defined instruction that is serially loaded into a boundary scan device.

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft jtag_instruction` in:

`/designs/`*`top_design`*`/dft/boundary_scan/jtag_instructions/`*`instruction`*

Options and Arguments

`-capture` *`string`*	Specifies the values that must be captured into a register during the `CaptureDR` state.
`-design` *`design`*	Specifies the name of the design for which the JTAG instruction is defined. If you omit the design name and multiple designs are loaded, the top-level design of the current directory of the design hierarchy is used.
`-length` *`integer`*	Specifies the length of the custom test data register (TDR).
`-name` *`string`*	Specifies the name of the user-defined instruction.
`-opcode` *`string`*	Specifies the binary code for this instruction.
`-private`	Specifies that the defined instruction is private.
`-register` *`string`*	Specifies the name of the custom test data register (TDR).
`-register_capturedr {`*`pin`*`|`*`port`*`}`	Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_CAPTUREDR` pin on the `JTAG_MACRO` subdesign.

`-register_clockdr {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_CLOCKDR` pin on the `JTAG_MACRO` subdesign.

`-register_decode {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_INSTRUCTION_DECODE_`*`instruction`* pin on the `JTAG_MACRO` subdesign, where *`instruction`* is the name that you specified through the `-name` option.

`-register_reset {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_RESET` pin on the `JTAG_MACRO` subdesign.

`-register_reset_inverted`

Specifies that the `JTAG_RESET` pin of the custom test register (TDR) has an active low polarity.

`-register_runidle {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_RUNIDLE` pin on the `JTAG_MACRO` subdesign.

`-register_shiftdr {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_SHIFTDR` pin on the `JTAG_MACRO` subdesign.

`-register_shiftdr_inverted`

Specifies that the `JTAG_SHIFTDR` pin of the custom test register (TDR) has an active low polarity.

`-register_tck {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_TCK` pin on the `JTAG_MACRO` subdesign.

`-register_tdi {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_TDI` pin on the `JTAG_MACRO` subdesign.

`-register_tdo {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_`*`register`*`_TDO` pin on the `JTAG_MACRO` subdesign, where *`register`* is the name that you specified through the `-register` option

`-register_updatedr {`*`pin`*`|`*`port`*`}`

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_UPDATEDR` pin on the `JTAG_MACRO` subdesign.

`-tap_decode {`*`pin`*`|`*`port`*`}`

Specifies the name of the instruction-specific decode pin that must be created on the `JTAG_MACRO` subdesign.

`-tap_tdo {`*`pin`*`|`*`port`*`}`

Specifies the name of the instruction-specific test data output (TDO) pin that must be created on the `JTAG_MACRO` subdesign.

Examples

- The following example defines a private instruction `PROGRAM_TCB` for custom test data register `TCB_REG` that has a length of 8 bits. The opcode for the instruction is `0101`.

  ```
  define_dft jtag_instruction -name PROGRAM_TCB -opcode 0101 -register TCB_REG
  -length 8 -private
  ```

Related Information

Inserting Boundary Scan Logic in *Design for Test in Encounter RTL Compiler*

Inserting JTAG Macro Logic in *Design for Test in Encounter RTL Compiler*

Inserting Memory Built-In-Self-Test Logic in *Design for Test in Encounter RTL Compiler*

Affects these commands:	insert_dft boundary_scan on page 607
	insert_dft jtag_macro on page 614
	insert_dft mbist on page 618
Related command:	define_dft jtag_instruction_register on page 547
Sets these attributes:	JTAG Instruction Attributes

define_dft jtag_instruction_register

```
define_dft jtag_instruction_register
     -name string
     [-length integer] [-capture string]
     [-design design]
```

Customizes the instruction register to allow adding user-defined instructions.

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft jtag_instruction_register` in:

`/designs/`*`top_design`*`/dft/boundary_scan/`*`register_name`*

Options and Arguments

`-capture` *`string`*	Specifies the capture value of the instruction register. According to the IEEE 1149.1 standard the last two bits of the capture value must be `01`. *Default*: 01
`-design` *`design`*	Specifies the name of the design for which the instruction register is customized. If you omit the design name and multiple designs are loaded, the top-level design of the current directory of the design hierarchy is used.
`-length` *`integer`*	Specifies the length of the instruction register. The length of the register is determined by the number of user-defined instructions that you want to add. If *n* is the number of bits in the instruction register, a total of 2^n instructions can be defined including the four mandatory instructions. *Default*: 2
`-name` *`string`*	Specifies the name of the user-defined instruction register.

Examples

- The following example defines instruction register `INSTR_REGISTER` with length 3. A register of length 3 allows you to create 2^3 instructions, or four user-defined instructions besides the four mandatory instructions.

  ```
  define_dft jtag_instruction_register -name INSTR_REGISTER -length 3
  ```

Related Information

Inserting Boundary Scan Logic in *Design for Test in Encounter RTL Compiler*

Inserting JTAG Macro Logic in *Design for Test in Encounter RTL Compiler*

Inserting Memory Built-In-Self-Test Logic in *Design for Test in Encounter RTL Compiler*

Related commands:	define_dft jtag_instruction on page 543
Affects these commands:	insert_dft boundary_scan on page 607
	insert_dft jtag_macro on page 614
	insert_dft mbist on page 618
Related attributes:	JTAG Instruction Register Attributes

define_dft jtag_macro

```
define_dft jtag_macro
     [-module subdesign] [-libcell libcell]
     [-instance instance]
     [-bsr_shiftdr pin|port|subport]
     [-bsr_clockdr pin|port|subport]
     [-bsr_updatedr pin|port|subport]
     [-reset pin|port|subport]
     [-runidle pin|port|subport]
     [-shiftdr pin|port|subport]
     [-clockdr pin|port|subport]
     [-updatedr pin|port|subport]
     [-capturedr pin|port|subport]
     [-mode_a pin|port|subport]
     [-mode_b pin|port|subport]
     [-mode_c pin|port|subport]
     -tdi pin|port|subport
     [-boundary_tdo pin|port|subport]
     -tdo pin|port|subport
     [-highz pin|port|subport]
     [-tdo_enable pin|port|subport]
     -tck pin|port|subport
     -tms pin|port|subport
     [-trst pin|port|subport]
     [-dot6_acdcsel pin|port|subport]
     [-dot6_preset_clock pin|port|subport]
     [-dot6_trcell_enable pin|port|subport]
     [-dot6_acpulse pin|port|subport]
     [-por pin|port|subport] [-name string]
```

Identifies a pre-instantiated JTAG Macro.

Options and Arguments

`-boundary_tdo {`*`pin|port|subport`*`}`

Specifies the boundary register `TDO input` pin on the `JTAG_MACRO`.

`-bsr_clockdr {`*`pin|port|subport`*`}`

Specifies the clock data register (`CLOCKDR`) output pin for the boundary-scan register.

`-bsr_shiftdr {`*`pin|port|subport`*`}`

Specifies the shift data register (`SHIFTDR`) output pin for the boundary-scan register.

`-bsr_updatedr {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the update data register (`UPDATEDR`) output pin for the boundary-scan register.

`-clockdr {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the clock data register (`CLOCKDR`) output pin custom test data register.

`-capturedr {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the capture data register (`CAPTUREDR`) output pin custom test data register.

`--dot6_acdcsel {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the logical OR of the decoded `EXTEST_PULSE` and `EXTEST_TRAIN` instructions.

`-dot6_acpulse {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the AC test signal output of the `JTAG_MACRO`.

`-dot6_preset_clock {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the `preset_clock` output pin that provides a positive-active edge-sensitive clock signal to test receivers that have edge-sensitive initialization.

`-dot6_trcell_enable {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the logical OR of `EXTEST`, `EXTEST_PULSE` and `EXTEST_TRAIN` used to enable the test receiver cells.

`-highz {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the `HIGHZ` output pin to place the I/O pads in their `HIGHZ` state.

`-instance` *`instance`* Specifies the instance name of the module where the `JTAG_MACRO` resides.

`-libcell` *`cell`* Specifies the library cell for where the `JTAG_MACRO` resides.

`-mode_a {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the `mode_a` output pin to configure boundary cells in the boundary-scan register.

`-mode_b {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the `mode_b` output pin to configure boundary cells in the boundary-scan register.

`-mode_c {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the `mode_c` output pin to configure boundary cells in the boundary-scan register.

`-module` *`subdesign`* Specifies the subdesign (module) to which the element belongs.

`-name` *`subdesign`* Specifies the name of the `JTAG_MACRO`. If not specified, an object will be added under the `boundary_scan/jtag_macros` vdir with a default name of `jtag_macro_`*`n`*.

`-por {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the power-on reset input pin on the `JTAG_MACRO`.

`-reset {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the reset output pin indicating that the `JTAG_MACRO` is in the Test-Logic-Reset state.

`-runidle {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the `JTAG_RUNIDLE` output pin indicating that the `JTAG_MACRO` is in the Run-Test-Idle state.

`-shiftdr {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the shift data register (`SHIFTDR`) output pin custom test data register.

`-tck {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the TAP controller `TCK` input pin on the `JTAG_MACRO`. This option is required.

`-tdi {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the TAP controller `TDI` output pin on the `JTAG_MACRO`. This option is required.

`-tdo {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the TAP controller `TDO` input pin on the `JTAG_MACRO`. This option is required.

`-tdo_enable {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the enable output pin which drives the JTAG `TDO` output enable pin.

`-tms {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the TAP controller `TMS` input pin on the `JTAG_MACRO`.This option is required.

`-trst {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the TAP controller `TRST` input pin on the `JTAG_MACRO`.

`-updatedr {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Specifies the update data register (`UPDATEDR`) output pin custom test data register.

Examples

- The following example defines a third party TAP controller with a specified instance location.

```
rc:/> define_dft jtag_macro -instance user_defined_jtag \
-highz MY_JTAG_INSTRUCTION_DECODE_CTRL_HIGHZ \
-bsr_clockdr MY_JTAG_BOUNDARY_CLOCKDR -bsr_shiftdr MY_JTAG_BOUNDARY_SHIFTDR \
-bsr_updatedr MY_JTAG_BOUNDARY_UPDATEDR \
-mode_a  MY_JTAG_INSTRUCTION_DECODE_MODE_A \
-mode_b MY_JTAG_INSTRUCTION_DECODE_MODE_B \
-mode_c MY_JTAG_INSTRUCTION_DECODE_MODE_C -tdi MY_JTAG_TDI -tdo MY_JTAG_TDO \
-tms MY_JTAG_TMS -tck MY_JTAG_TCK -trst MY_JTAG_TRST \
-tdo_enable MY_JTAG_ENABLE_TDO -boundary_tdo MY_JTAG_BOUNDARY_TDO \
-por MY_JTAG_POR -name JM1
```

- The following example defines a third party TAP controller without a `TRST` input pin.

```
rc:/> define_dft jtag_macro -module MY_JTAG_MACRO \
-highz MY_JTAG_INSTRUCTION_DECODE_CTRL_HIGHZ \
-bsr_clockdr MY_JTAG_BOUNDARY_CLOCKDR -bsr_shiftdr MY_JTAG_BOUNDARY_SHIFTDR \
-bsr_updatedr MY_JTAG_BOUNDARY_UPDATEDR \
-mode_a MY_JTAG_INSTRUCTION_DECODE_MODE_A \
-mode_b MY_JTAG_INSTRUCTION_DECODE_MODE_B \
-mode_c MY_JTAG_INSTRUCTION_DECODE_MODE_C -tdi MY_JTAG_TDI -tdo MY_JTAG_TDO \
-tms MY_JTAG_TMS -tck MY_JTAG_TCK -tdo_enable MY_JTAG_ENABLE_TDO \
-boundary_tdo MY_JTAG_BOUNDARY_TDO -por MY_JTAG_POR -name JM1
```

Related Information

Defining Pre-Existing JTAG Macro Logic in *Design for Test in Encounter RTL Compiler*

Sets these attributes:	JTAG Macro Attributes

define_dft mbist_clock

```
define_dft mbist_clock test_clock -name mbist_clock
     [-design design] [-domain mbist_clock_domain]
     [-period integer] [-divide_period integer]
     |-hookup_pin pin [-hookup_period string] [-hookup_polarity string]]
     {pin|port} [{pin|port}] ...
```

Defines an MBIST clock and associates a clock waveform with the clock. The clock waveform can be different from the system clocks.

If you do not define MBIST clocks, the DFT rule checker automatically analyzes the MBIST clocks and creates these objects with a default waveform. The waveform information is useful in determining how to order scan flip-flops in a chain, and where to insert data-lockup elements in the chain.

MBIST clock waveforms are used to order flip-flops that belong to the same MBIST clock domain to minimize the addition of lockup elements. Flip-flops that are triggered first are ordered and connected last in a chain.

The command returns the directory path to the `mbist` object that it creates.You can find the objects created by the `define_dft test_clock` constraints in:

```
/designs/design/dft/mbist_clocks
```

Options and Arguments

`-design` *`design`*

Specifies the name of the design for which the MBIST clock is defined.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-divide_period` *`integer`*

Together with the `-period` option, determines the MBIST clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: `1`

`-domain` *`mbist_clock_domain`*

Specifies the clock domain associated with the MBIST clock.

Clocks belonging to the same domain can be mixed in a chain.

If you omit this option, a new DFT clock domain is created and associated with the MBIST clock.

Flip-flops belonging to different test clocks in the same domain can be mixed in a chain. Lockup elements can be added between the flip-flops belonging to different test clocks.

`-hookup_period` *`integer`*

Specifies the period interval at the hookup pin. The hookup period is is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: value of `-period` option

`-hookup_pin` *`pin`*

Specifies the core-side hookup pin to be used for the top-level MBIST clock during DFT synthesis.

Note: When you specify this option, the RC-DFT engine does not validate the controlability of any logic between the top-level test clock and its designated hookup pin under test-mode setup.

`-hookup_polarity {inverted | non_inverted}`

Specifies the polarity of the MBIST clock signal at the core-side hookup pin.

`-name` *`test_clock`*

Specifies the name of the MBIST clock that is being defined.

Each clock object in your design must have a unique name. If you define a new MBIST clock with the same name as an existing clock, an error message will be issued.

Note: The clock name allows you to search for the clock later (through the `find` command) or to recognize it in reports.

`-period` *`integer`*

Together with the `-divide_period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: `50000` (20 MHz test clock)

`{`*`pin`*`|`*`port`*`}`	Specifies the MBIST clock input pin or port. If you specify multiple pins, these pins are assumed to have zero skew: they can be mixed without lockup latches.

Example

- The following example defines three MBIST clocks, four MBIST clock ports and two MBIST clock domains.

  ```
  define_dft mbist_clock -name CLK1X -domain domain_1 -period 20000 CLK1
  define_dft mbist_clock -name CLK2X -domain domain_1 -period 20000 CLK2 CLK2b
  define_dft mbist_clock -name CLK3X -domain domain_2 -period 20000 CLK3
  ```

  The four MBIST clock ports are `CLK1`, `CLK2`, `CLK2b`, and `CLK3`. MBIST clock `CLK2X` comprises equivalent test clocks `CLK2` and `CLK2B` (they can be mixed in the same scan chain without any lockup element). Test clocks `CLK1X`, `CLK2X` belong to the same MBIST clock domain and are compatible (requires lockup elements between compatible chain segments triggered by the different test clocks). MBUST clock `CLK3X` belongs to its own domain.

Related Information

Specifying an Internal Clock Branch as a Separate Test Clock in *Design for Test in Encounter RTL Compiler*

Defining an Equivalent Test Clock for Different Top-level Clock Pins in *Design for Test in Encounter RTL Compiler*

Root Attributes in *Attribute Reference for Encounter RTL Compiler*

Affects these commands:	check_dft_rules on page 501
	connect_scan_chains on page 523
	fix_dft_violations on page 586
	report dft_chains on page 647
Sets these attributes:	MBIST Clock Attributes

define_dft preserved_segment

```
define_dft preserved_segment [-name segment_name]
     { {instance|segment_name}... [-sdi pin] [-sdo pin]
     | -analyze -sdi {pin|port} -sdo {pin|port} }
     [-connected_shift_enable]
     [-connected_scan_clock_a]
     [-connected_scan_clock_b]
     [-allow_reordering]
```

Defines a preserved segment. In a preserved segment, the elements of the mapped segment are already connected in the specified order, and they cannot be reordered by a physical scan reordering tool.

A preserved segment is a user-specified scan segment which can be associated with either

- A user-defined top-level chain—created using the `define_dft scan_chain` command
- A tool-created scan chain—created using the `connect_scan_chains` command

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft preserved_segment` constraints in:

`/designs/`*`top_design`*`/dft/scan_segments`

Options and Arguments

`-allow_reordering`	Indicates whether the order of the elements can be changed during scan configuration and by a physical scan reordering tool.
`-analyze`	Analyzes the connectivity of the scan segment, and returns the elements of the segment given its endpoints.

`-connected_scan_clock_a (-connected_scan_clock_b)`

Indicates that the `scan_clock_a` (`scan_clock_b`) port of the module boundary is driven by external logic (preconnected). The external logic connected to the `scan_clock_a` (`scan_clock_b`) pin of the module will not be modified by the scan configuration engine.

This option applies only for the clocked LSSD scan style.

`-connected_shift_enable`

Indicates that the shift enable port of the module boundary is driven by external logic (preconnected) or that the shift enable signal is internally generated within the module boundary. In either case, the external logic connected to the shift enable pin of the module, or the internal logic driving the shift enable pins of the flip-flops in the module will not be modified by the scan configuration engine.

`{`*`instance`*`|`*`segment_name`*`}`

Specifies an element in the scan segment being defined. List the elements in the order they should appear in the scan chain (in shift order, that is, left-most corresponds to first bit shifted-in). An element can be a flip-flop instance, a combinational instance, or a scan segment.

Additionally, the hierarchical scan data input and output pins of the segment can be specified using the `-sdi` and `-sdo` options respectively. If the hierarchial `SDI` and `SDO` pins of the segment are both at the boundary of the same lower module, the RC-DFT engine also traces the shift-enable signal back from the scan registers in the segment to the same module boundary. It hooks up the shift-enable signal at the module boundary whenever applicable (only if you did not specify the `-connected_shift_enable` option).

Specifies an element in the scan segment being defined. List the elements in the order they should appear in the scan chain (in shift order, that is, left-most corresponds to first bit shifted-in). An element can be a flip-flop instance, a buffer, an inverter, a (hierarchal) pin, or a scan segment.

`-name` *`segment_name`* Defines a name for the segment that you can use to reference in the `define_dft scan_chain` constraint.

`-sdi (-sdo)` Specifies the scan data input (scan data output) of the segment. Specify a hierarchical pin name or port.

Examples

- The following example defines a pre-existing segment in instance u_a using its scan data input and output pins.

```
rc:/> define_dft preserved_segment -name segmenta -analyze \
-sdi */u_a/SIa -sdo */u_a/SOa
```

- The following example defines a pre-existing segment by specifying its hierarchical scan data input and output pins, and its elements consisting of two combinational components, four sequential registers, a scan abstract segment, and a combinational component endpoint.

```
define_dft preserved_segment -name preservedSeg \
    -sdi i_core/i_anor1/B0 -sdo i_core/bufToAnchorSeg/Y \
    i_core/i_anor1 i_core/i_anor2 \
    i_core/i_flop11 i_core/i_flop22 i_core/i_flop33 i_core/i_flop44 \
    absSeg \
i_core/bufToAnchorSeg
```

Related Information

Creating Head, Body, and Tail Segments in *Design for Test in Encounter RTL Compiler*

Affects this constraint:	define_dft scan_chain on page 560
Affects these commands:	connect_scan_chains on page 523
	report dft_chains on page 647
Sets these attributes:	Scan Segment Attributes

define_dft scan_chain

```
define_dft scan_chain [-name name]
   {[-sdi sdi -sdo sdo [-create_ports]
     {-shared_output [-shared_select test_signal] |
      -non_shared_output}
    [-shift_enable test_signal]
    [-head segment] [-tail segment] [-body segment]
    [-complete | -max_length integer]
    [-domain test_clock_domain [-edge {rise|fall}]]
    [-terminal_lockup {level_sensitive|edge_sensitive}]
    [-hookup_pin_sdi pin] [-hookup_pin_sdo pin]
    [-configure_pad {tm_signal | se_signal} ]
   |-analyze -sdo sdo [-sdi sdi] [-dont_overlay]
     {-shared_out  | -non_shared_out} }
```

Creates a scan chain or analyzes an existing chain with the specified input and output scan data ports.

If you created a scan chain, the command returns the directory path to the `scan_chain` object that it creates. For newly created scan chains you can find the objects created by the `define_dft scan_chain` constraints in:

`/designs/`*`top_design`*`/dft/scan_chains`

If you successfully analyzed an existing scan chain, the command returns the directory path to the `actual_scan_chain` object that it creates. For successfully analyzed scan chains, you can find the objects created by the `define_dft scan_chain` constraints in:

`/designs/`*`top_design`*`/dft/report/actual_scan_chains`

Options and Arguments

`-analyze`	Analyzes the connectivity of an existing top-level scan chain in a structural netlist compiled in a previous RC session. You must at least specify the scan data output pin and optionally the scan data input pin to identify the chain.
`-body` *`segment`*	Indicates that the specified segment (an ordered set of scan flip-flops) is part of the body of the scan chain. The segment must have been previously defined with a `define_dft xxx_segment` constraint, where xxx is either `abstract`, `fixed`, `floating`, `preserved`, or `shift_register`.
`-complete`	Specifies that the defined chain is complete and no other flip-flops should be added to the chain.

`-configure_pad {`*`tm_signal`*` | `*`se_signal`*`}`

Specifies the test signal (test mode or shift enable signal) that the RC-DFT engine must use if it needs to configure the pad connected to the scan data input or output signal to control the data direction during test mode.

Recommendation: If the scan data input and output pin are shared with functional pins, it is recommended to use the shift enable test signal to configure pads. This will allow the pads to shift-in and shift-out data when shift-enable signal is active (scan-shift mode), and will allow the pads to operate in functional mode when shift-enable signal is inactive (capture mode).

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-create_ports`

Specifies whether to create the scan data input and output ports if they do not exist.

If you do not specify the scan data input or output signals using the `-sdi` and `-sdo` options, the ports can be created and named as *`prefix`*`_sdi_`*`num`* and *`prefix`*`_sdo_`*`num`*, where *`prefix`* is the value of the `dft_prefix` attribute.

`-domain `*`test_clock_domain`*

Specifies the DFT clock domain to associate with the scan chain. This clock domain must have been previously identified by the `check_dft_rules` command or defined with the `define_dft test_clock` constraint.

If you omit this option, and segments have been defined for the chain, the scan chain is automatically associated with the appropriate DFT clock domain. In the absence of segments, the scan chain can be assigned to any DFT clock domain.

Note: This option only applies to the muxed scan style.

`-dont_overlay`	Prevents that RTL Compiler reassociates (or overlays) user-defined segments of type preserve or fixed to its analyzed scan chains. Consequently, the segments are not re-established as a fixed entity (and hence are reorderable) when determining how to partition the chains for physical-based reordering. If these segments include multi-input combinational logic gates in the scan data path, the `write_scandef` command uses these gates to partition the analyzed scan chains into n-reorderable segments (referred to as scanDEF chains). The register before a multi-input combinational logic gate becomes the `STOP` point for one scanDEF chain, while the register after a combinational gate becomes the `START` point of another scanDEF chain.
`-edge {rise|fall}`	Specifies whether to use the falling or rising edge of the test clocks in the specified DFT clock domain. You can specify this option only if you specified `-domain`. If you omit this option, the scan flip-flops triggered by the different active edges of the test clocks will be placed on their own scan chain. This option is ignored if you enabled the `dft_mix_clock_edges_in_scan_chains` attribute. **Note:** This option only applies to the muxed scan style.
`-head` *`segment`*	Indicates that the specified segment (an ordered set of scan flip-flops) must be placed at the head (closest to the scan data input) of the scan chain. The segment must have been previously defined with a `define_dft xxx_segment` constraint, where xxx is either `abstract`, `fixed`, `floating`, `preserved`, or `shift_register`.
`-hookup_pin_sdi` *`pin`*	
	Specifies the core-side hookup pin to be used for the scan data input signal during scan chain connection. **Note:** When you specify this option, the RC-DFT engine does not validate the control ability of any logic between the top-level scan data input signal and its designated hookup pin under test-mode setup.

`-hookup_pin_sdo` *`pin`*

Specifies the core-side hookup pin to be used for the scan data output signal during scan chain connection.

Note: When you specify this option, the RC-DFT engine does not validate the control ability of any logic between the top-level shift_enable signal and its designated hookup pin under test-mode setup.

`-max_length` *`integer`*

Specifies the maximum length that you allow for this scan chain.

If you omit this option, the maximum length defaults to the value of the `dft_max_length_of_scan_chains` design attribute.

Note: This option is ignored if the scan chain is defined with the `-complete` option, or if the number of flip-flop instances in a head, body, or tail segment exceeds the maximum value.

`-name` *`name`*

Specifies the name of the scan chain.

If you omit this option, a default name is used.

`-sdi` *`sdi`*

Specifies the scan data input signal.

- If you want to *create* a chain, specify a top-level port or a hierarchical pin name in case of an existing port or pin. If you want the tool to create the port, use the `create_ports` option and a primary input port with the specified name will be created.
- If you want to *analyze* an existing chain, specify a top-level port, a hiearchical pin, subport, or instance pin.

`-sdo` *`sdo`*

Specifies the scan data output signal.

- If you want to *create* a chain, specify a top-level port or a hierarchical pin name in case of an existing port or pin. If you want the tool to create the port, use the `create_ports` option and a primary output port with the specified name will be created.
- If you want to analyze an existing chain, specify a top-level port, a hiearchical pin, subport, or instance pin.

`-shared_select` *`test_signal`*

Specifies the select control signal to the mux inserted for a shared output port.

Default: The default shift-enable signal or chain-specific shift-enable signal is used as the select control signal to the mux.

`{-shared_output | -non_shared_output}`

Specifies whether an existing functional output port can be used as scan data output port. If the functional port can be used for scan data purposes, a mux is inserted in the scan data path by the `connect_scan_chains` command.

One of these options must be specified when the specified scan data output signal is already connected in the circuit.

`-shift_enable` *`test_signal`*

Designates a chain-specific shift-enable port or pin.

If you omit this option, the default shift-enable signal specified using a `define_dft shift_enable` constraint is used.

`-tail` *`segment`*

Indicates that the specified segment (an ordered set of scan flip-flops) must be placed at the tail (closest to the scan data output) of the scan chain. The segment must have been previously defined with a `define_dft xxx_segment` constraint, where xxx is either `abstract`, `fixed`, `floating`, `preserved`, or `shift_register`.

`-terminal_lockup {level_sensitive | edge_sensitive}`

Specifies the type of lockup element that configuration can insert at the tail end of the chain to connect to the specified scan data output signal.

If this option is not specified, no terminal lockup element will be inserted.

Note: This option only applies to the muxed scan style.

Example

- The following example creates a chain containing 3 segments previously defined:

```
rc:/des*/test> define_dft scan_chain -sdi in[0] -sdo out[0] -shared_out \
    -head segHead -tail segTail -body segBody -name chain1
...
Info    : Added scan chain. [DFT-151]
        : scan chain successfully defined.
/designs/test/dft/scan_chains/chain1
```

- The following example analyzes an existing scan chain:

```
rc:/> define_dft scan_chain -name topChain -sdi SI -sdo SO -analyze
...
Info    : Added scan chain. [DFT-151]
        : scan chain successfully defined.
/designs/test/dft/report/actual_scan_chains/topChain
```

Related Information

Controlling Scan Configuration in *Design for Test in Encounter RTL Compiler*

Affected by this constraint:	define_dft abstract_segment on page 530
	define_dft fixed_segment on page 540
	define_dft floating_segment on page 542
	define_dft preserved_segment on page 557
	define_dft shift_enable on page 572
	define_dft shift_register_segment on page 575
	define_dft test_clock on page 577
Affects these commands:	connect_scan_chains on page 523
	report dft_chains on page 647
Affected by this attribute:	dft_max_length_of_scan_chains
	dft_mix_clock_edges_in_scan_chains
	dft_prefix
Sets these attributes:	Scan Chain Attributes

define_dft scan_clock_a

```
define_dft scan_clock_a
     [-name name] [-no_ideal] driver
     [-period integer] [-divide_period integer]
     [-rise integer] [-divide_rise integer]
     [-fall integer] [-divide_fall integer]
     [ [-hookup_pin pin [-hookup_polarity string]]
       [-configure_pad {tm_signal|se_signal}]
     | [-create_port]]
```

Defines the scan clock of the master latch (`scan_clock_a`) of the clocked LSSD scan cell. The `scan_clock_a` signal controls the scan shifting of the master latch and is required for the clocked-LSSD scan style. The signal is created with active high polarity.

You can define only one signal for the design. If you specify more than one signal, the last definition overwrites the existing one.

The command returns the directory path to the `test_signal` object that it creates. You can find the object created by the `define_dft scan_clock_a` constraints in:

`/designs/`*`design`*`/dft/test_signals`

Options and Arguments

`-configure_pad {`*`tm_signal`*` | `*`se_signal`*`}`

Specifies the test signal that the RC-DFT engine must use if it needs to configure the pad connected to the `scan_clock_a` signal to control the data direction during test mode.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-create_port` Specifies whether to create the port if it does not exist.

`-divide_fall `*`integer`*

Together with the `-fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: `100`

`-divide_period` *`integer`*

Together with the `-period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: `1`

`-divide_rise` *`integer`*

Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: `100`

`driver`

Specifies the driving pin or port for the scan clock of the master latch (`scan_clock_a`) of the clocked LSSD scan cell.

`-fall` *`integer`*

Together with the `-divide_fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: `60`

`-hookup_pin` *`pin`*

Specifies the core-side hookup pin to be used for the `scan_clock_a` signal during scan chain connection.

Note: When you specify this option, the RC-DFT engine does not validate the controlability of any logic between the top-level `scan_clock_a` signal and its designated hookup pin under test-mode setup.

`-hookup_polarity {inverted|non_inverted}`

Specifies the polarity of the `scan_clock_a` signal at the core-side hookup pin.

`-name` *`name`*

Specifies the `test_signal` object name of the `scan_clock_a` signal.

If you omit this option, the RC-DFT engine assigns a name based on the hierarchical path of the driver, using underscores as delimiters in the path.

`-no_ideal`	Marks the `scan_clock_a` signal as non-ideal. This allows buffering of the `scan_clock_a` network during optimization. By default, the scan`_clock_a` signal is marked ideal. **Note:** If the test signal is marked as ideal, RTL Compiler sets the `ideal_network` attribute to `true` on the pin or port for the `scan_clock_a` signal
`-period` *`integer`*	Together with the `-divide_period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`. *Default*: `50000` (20 MHz test clock)
`-rise` *`integer`*	Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`. *Default*: `50`

Example

- The following example defines `sca` as the driver for the `scan_clock_a` signal and assigns SCA as the `test_signal` object name.

```
define_dft scan_clock_a -name SCA sca
```

Related Information

Defining Scan Clock Signals in *Design for Test in Encounter RTL Compiler*

Affects these commands:	connect_scan_chains on page 523
	report dft_chains on page 647
	write_atpg on page 656
Sets these attributes:	Test Signal Attributes
Affected by this attribute:	dft_scan_style

define_dft scan_clock_b

```
define_dft scan_clock_b
     [-name name] [-no_ideal] driver
     [-period integer] [-divide_period integer]
     [-rise integer] [-divide_rise integer]
     [-fall integer] [-divide_fall integer]
     [ [-hookup_pin pin [-hookup_polarity string]]
       [-configure_pad {tm_signal|se_signal}]
     | [-create_port]]
```

Defines the scan clock of the slave latch (`scan_clock_b`) of the clocked LSSD scan cell. The `scan_clock_b` signal controls the scan shifting of the slave latch and is required for the clocked-LSSD scan style. The signal is created with active high polarity.

You can define only one signal for the design. If you specify more than one signal, the last definition overwrites the existing one.

The command returns the directory path to the `test_signal` object that it creates. You can find the object created by the `define_dft scan_clock_b` constraints in:

`/designs/*design*/dft/test_signals`

Options and Arguments

`-configure_pad {tm_signal | se_signal}`

Specifies the test signal that the RC-DFT engine must use if it needs to configure the pad connected to the `scan_clock_b` signal to control the data direction during test mode.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-create_port` Specifies whether to create the port if it does not exist.

`-divide_fall integer`

Together with the `-fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: `100`

`-divide_period` *`integer`*

Together with the `-period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: `1`

`-divide_rise` *`integer`*

Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: `100`

`driver`

Specifies the driving pin or port for the scan clock of the slave latch (`scan_clock_b`) of the clocked LSSD scan cell.

`-fall` *`integer`*

Together with the `-divide_fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: `80`

`-hookup_pin` *`pin`*

Specifies the core-side hookup pin to be used for the `scan_clock_b` signal during scan chain connection.

Note: When you specify this option, the RC-DFT engine does not validate the controlability of any logic between the top-level `scan_clock_b` signal and its designated hookup pin under test-mode setup.

`-hookup_polarity {inverted|non_inverted}`

Specifies the polarity of the `scan_clock_b` signal at the core-side hookup pin.

`-name` *`name`*

Specifies the `test_signal` object name of the `scan_clock_b` signal.

If you omit this option, the RC-DFT engine assigns a name based on the hierarchical path of the driver, using underscores as delimiters in the path.

`-no_ideal`	Marks the `scan_clock_b` signal as non-ideal. This allows buffering of the `scan_clock_b` network during optimization. By default, the `scan_clock_b` signal is marked ideal. **Note:** If the test signal is marked as ideal, RTL Compiler sets the `ideal_network` attribute to `true` on the pin or port for the `scan_clock_b` signal
`-period` *`integer`*	Together with the `-divide_period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`. *Default*: `50000` (20 MHz test clock)
`-rise` *`integer`*	Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`. *Default*: `70`

Example

■ The following example defines `sca` as the driver for the `scan_clock_b` signal and assigns `SCB` as the `test_signal` object name.

```
define_dft scan_clock_b -name SCB scb
```

Related Information

Defining Scan Clock Signals in *Design for Test in Encounter RTL Compiler*

Affects these commands:	connect_scan_chains on page 523
	report dft_chains on page 647
	write_atpg on page 656
Sets these attributes:	Test Signal Attributes
Affected by this attribute:	dft_scan_style

define_dft shift_enable

```
define_dft shift_enable [-name name] -active {low|high}
     [-default] [-no_ideal]
     [ [-hookup_pin pin [-hookup_polarity string]]
       [-configure_pad {tm_signal|se_signal}]
     | -create_port ]
     {pin|port} [-design design]
```

Specifies the name and active value of the input signal that activates scan shifting. The input signal can be defined on a top-level port or an internal driving pin. This type of input signal is required by the `muxed_scan` style. The active value of the shift-enable signals is propagated through the design by the `check_dft_rules` command.

The command returns the directory path to the `test_signal` object that it creates. You can find the objects created by the `define_dft shift_enable` constraints in:

`/designs/design/dft/test_signals`

Options and Arguments

`-active {low | high}`	Specifies the active value for the shift-enable signal.
`-configure_pad {tm_signal | se_signal}`	Specifies the test signal that the RC-DFT engine must use if it needs to configure the pad connected to the shift-enable signal to control the data direction during test mode. **Note:** You must have specified the test signal using either the `define_dft test_mode` or `define_dft shift_enable` constraint.
`-create_port`	Specifies whether to create the port if it does not exist.
`-default`	Designates the specified signal as the default shift-enable signal for chains for which you omit the `-shift-enable` option. **Note:** You can designate only one signal as the default shift-enable signal.

`-design` *`design`*

Specifies the name of the design for which the shift enable is defined.

If you omit the design name and multiple designs are loaded, the top-level design of the current directory of the design hierarchy is used.

`-hookup_pin` *`pin`*

Specifies the core-side hookup pin to be used for the top-level shift-enable signal during DFT synthesis.

Note: When you specify this option, the RC-DFT engine does not validate the controlability of any logic between the top-level shift_enable signal and its designated hookup pin under test-mode setup.

`-hookup_polarity {inverted | non_inverted}`

Specifies the polarity of the shift-enable signal at the core-side hookup pin.

`-name` *`name`*

Specifies the `test_signal` object name of the shift-enable signal.

If you omit this option, the RC-DFT engine assigns a name based on the hierarchical path of the driver, using underscores as delimiters in the path.

`-no_ideal`

Marks the shift-enable signal as non-ideal. This allows buffering of the shift-enable network during optimization.

Default: The shift-enable signal is marked ideal.

Note: If the test signal is marked as ideal, RTL Compiler sets the `ideal_network` attribute to `true` on the pin or port for the shift-enable signal

`{`*`pin`*`|`*`port`*`}`

Specifies the driving pin or port for the shift-enable signal.

Note: If multiple designs are loaded and you did no specify the `-design` option, you can also specify the full path to the driver.

Example

- When the following constraint is given, the `check_dft_rules` command propagates a logic1 from the `p_top/SE` pin into the design.

```
define_dft shift_enable -active low -hookup_pin p_top/SE -hookup_polarity inverted
```

Related Information

Defining Shift-Enable Signals in *Design for Test in Encounter RTL Compiler*

Affects these commands:	check_dft_rules on page 501
	connect_scan_chains on page 523
	insert_dft shadow_logic on page 627
	report dft_chains on page 647
Sets these attributes:	Test Signal Attributes

define_dft shift_register_segment

```
define_dft shift_register_segment [-name segment_name]
     -start_flop instance
     -end_flop instance
```

Defines a shift register. Because a shift register is a shiftable scan chain segment, the RC-DFT engine can use the functional path of the shift register as the scan path by only scan-replacing the first flop in the shift register segment, while maintaining the existing connectivity of the flops.

Note: A shift register segment can only contain flops driven by the same clock and same clock edge.

A shift register is a user-specified scan segment which can be associated with either

- A user-defined top-level chain—created using the `define_dft scan_chain` command
- A tool-created scan chain—created using the `connect_scan_chains` command

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft shift_register_segment` constraint in:

`/designs/`*`top_design`*`/dft/scan_segments`

Note: Shift register segments are only supported for the muxed scan style.

Options and Arguments

`-end_flop` *`instance`*	Specifies the last flop in the shift register. Specify the hierarchical instance name of the flop.
`-name` *`segment_name`*	Defines a name for the segment that you can use to reference in the `define_dft scan_chain` constraint.
`-start_flop` *`instance`*	Specifies the first flop in the shift register. Specify the hierarchical instance name of the flop.

Examples

- The following example defines a shift register.

```
define_dft shift_register_segment -name myreg \
-start_flop *seq/out_reg_0 -end_flop *seq/out_reg_7
```

Related Information

Identifying Shift Registers in the Design in *Design for Test in Encounter RTL Compiler*

Affects this constraint:	define_dft scan_chain on page 560
Affects these commands:	connect_scan_chains on page 523
	report dft_chains on page 647
Related command:	identify_shift_register_scan_segments on page 593
Sets these attributes:	Scan Segment Attributes

define_dft test_clock

```
define_dft test_clock -name test_clock
     [-design design] [-domain test_clock_domain]
     [-period integer] [-divide_period integer]
     [-rise integer] [-divide_rise integer]
     [-fall integer] [-divide_fall integer]
     |-hookup_pin pin [-hookup_polarity string]]
     [-controllable] {pin|port} [{pin|port}] ...
```

Defines a test clock and associates a test clock waveform with the clock. The test clock waveform can be different from the system clocks.

If you do not define test clocks, the DFT rule checker automatically analyzes the test clocks and creates these objects with a default waveform. The waveform information is useful in determining how to order scan flip-flops in a chain, and where to insert data-lockup elements in the chain.

Test clock waveforms are used to order flip-flops that belong to the same DFT test clock domain to minimize the addition of lockup elements. Flip-flops that are triggered first are ordered and connected last in a chain.

The command returns the directory path to the `test_clock` object that it creates.You can find the objects created by the `define_dft test_clock` constraints in:

`/designs/`*`design`*`/dft/test_clock_domains`

Options and Arguments

`-controllable`	When specifying an internal pin for a test clock, this option indicates that the internal clock pin is controllable in test mode (for example, Built-in-Self-Test (BIST)). If you do not specify this option, the rule checker must be able to trace back from the internal pin to a controllable top-level clock pin. **Note:** If you specify an internal pin as being controllable, you need to ensure that this pin can be controlled for the duration of the test cycle. The tool will *not* validate your assumption.
`-design` *`design`*	Specifies the name of the design for which the test clock is defined. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-divide_fall` *`integer`*

Together with the `-fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: `100`

`-divide_period` *`integer`*

Together with the `-period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: `1`

`-divide_rise` *`integer`*

Together with the `-rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: `100`

`-domain` *`test_clock_domain`*

Specifies the DFT clock domain associated with the test clock.

Clocks belonging to the same domain can be mixed in a chain.

If you omit this option, a new DFT clock domain is created and associated with the test clock.

Flip-flops belonging to different test clocks in the same domain can be mixed in a chain. Lockup elements can be added between the flip-flops belonging to different test clocks.

`-fall` *`integer`*

Together with the `-divide_fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: `90`

`-hookup_pin` *`pin`*

Specifies the core-side hookup pin to be used for the top-level test clock during DFT synthesis.

Note: When you specify this option, the RC-DFT engine does not validate the controlability of any logic between the top-level test clock and its designated hookup pin under test-mode setup.

`-hookup_polarity {inverted | non_inverted}`

Specifies the polarity of the test clock signal at the core-side hookup pin.

`-name` *`test_clock`*

Specifies the name of the test clock that is being defined.

Each clock object in your design must have a unique name. If you define a new test clock with the same name as an existing clock, an error message will be issued.

Note: The clock name allows you to search for the clock later (through the `find` command) or to recognize it in reports.

`-period` *`integer`*

Together with the `-divide_period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: `50000` (20 MHz test clock)

`{`*`pin`*`|`*`port`*`}`

Specifies the test clock input pin or port.

If you specify multiple pins, these pins are assumed to have zero skew: they can be mixed without lockup latches.

`-rise` *`integer`*

Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: `50`

Example

- The following example defines three test clocks, four test clock ports and two DFT clock domains.

```
define_dft test_clock -name CLK1X -domain domain_1 -period 20000 CLK1
define_dft test_clock -name CLK2X -domain domain_1 -period 20000 CLK2 CLK2b
define_dft test_clock -name CLK3X -domain domain_2 -period 20000 CLK3
```

The four test clock ports are CLK1, CLK2, CLK2b, and CLK3. Test clock CLK2X comprises equivalent test clocks CLK2 and CLK2B (they can be mixed in the same scan chain without any lockup element). Test clocks CLK1X, CLK2X belong to the same DFT clock domain and are compatible (requires lockup elements between compatible chain segments triggered by the different test clocks). Test clock CLK3X belongs to its own domain.

Related Information

Specifying an Internal Clock Branch as a Separate Test Clock in *Design for Test in Encounter RTL Compiler*

Defining an Equivalent Test Clock for Different Top-level Clock Pins in *Design for Test in Encounter RTL Compiler*

Root Attributes in *Attribute Reference for Encounter RTL Compiler*

Affects these commands:	check_dft_rules on page 501
	connect_scan_chains on page 523
	fix_dft_violations on page 586
	report dft_chains on page 647
Sets these attributes:	Test Clock Attributes

define_dft test_mode

```
define_dft test_mode [-name name] -active {low | high}
     [-no_ideal] [-scan_shift]
     [ [-hookup_pin pin [-hookup_polarity string]]
       [-configure_pad {tm_signal|se_signal}]
     | [-create_port | -shared_in] ]
     {pin|port} [-design design]
```

Specifies the input signal and constant value that is assigned during a test session. The input signal can be defined on a top-level port or an internal driving pin.

The active value of the test mode signals is propagated through the design by the `check_dft_rules` command. Unless defined with the `-scan_shift` option, the test signal is expected to stay active throughout a test session.

The command returns the directory path to the `test_signal` object that it creates. You can find the objects created by the `define_dft test_mode` constraints in:

`/designs/`*`design`*`/dft/test_signals`

Options and Arguments

`-active {low | high}`

Specifies the active value for the test mode signal.

`-configure_pad {`*`tm_signal`*` | `*`se_signal`*`}`

Specifies the test signal that the RC-DFT engine must use if it needs to configure the pad connected to the test mode signal to control the data direction during test mode.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-create_port` Specifies whether to create the port if it does not exist.

Note: This option cannot be specified with the `-shared_in` option.

`-design `*`design`* Specifies the name of the design for which the test mode signal is defined.

Note: If you omit the design name and multiple designs are loaded, the top-level design of the current directory of the design hierarchy is used. You can also specify the full path to the driver.

`{`*`pin`*`|`*`port`*`}`	Specifies the driving pin or port for the test signal. **Note:** If multiple designs are loaded and you did no specify the `-design` option, you can also specify the full path to the driver.
`-hookup_pin` *`pin`*	Specifies the core-side hookup pin to be used for the top-level test-mode signal during DFT synthesis. **Note:** When you specify this option, the RC-DFT engine does not validate the controllability of any logic between the top-level test-mode signal and its designated hookup pin under test-mode setup.
`-hookup_polarity {inverted|non_inverted}`	Specifies the polarity of the test-mode signal at the core-side hookup pin.
`-name` *`name`*	Specifies the `test_signal` object name of the test signal. If you omit this option, the RC-DFT engine assigns a name based on the hierarchical path of the driver, using underscores as delimiters in the path.
`-no_ideal`	Marks the test-mode signal as non-ideal. This allows buffering of the test-mode network during optimization. *Default:* The test-mode signal is marked ideal. **Note:** If the test signal is marked as ideal, RTL Compiler sets the `ideal_network` attribute to `true` on the pin or port for the test signal.
`-scan_shift`	Indicates that this test signal should only be held to its test-mode active value during the scan shift operation of the tester cycle. This option is used to designate those test signals which must be held to their non-controlling functional values to prevent the state of the flip-flops from being asynchronous set or reset while ATPG data is being shifted into or out of the scan chains. As a consequence of specifying this option, the test signal will be treated as a clock signal by the ATPG tool. This allows ATPG to have control over the logic state of the signal during both the scan-shift and capture operations of tester cycle. If this option is not specified, the test signal will be held to its test-mode active value for the duration of the tester cycle.

`-shared_in`	Specifies whether the input port is also used as a functional port. By default, the signal applied to the specified driving pin or port is considered to be a dedicated test signal. **Note:** This option cannot be specified with the `-create_port` option.

Example

- When the following constraint is given, the `check_dft_rules` command propagates a logic1 from the `pad_top/TM` pin into the design.

  ```
  define_dft test_mode -active high -hookup_pin pad_top/TM
  -hookup_polarity non_inverted
  ```

Related Information

Defining Test Mode Signals in *Design for Test in Encounter RTL Compiler*

Affects these commands:	check_dft_rules on page 501
	fix_dft_violations on page 586
	insert_dft shadow_logic on page 627
	insert_dft test_point on page 631
Sets these attributes:	Test Signal Attributes

dft_trace_back

```
dft_trace_back
     [-mode integer] [-polarity] [-continue] {port|pin} [-print]
```

Returns the pin or port found by tracing back one level from the specified pin or port based on the requested mode. If a constant is encountered, the command returns 0 or 1.

Options and Arguments

`-mode` *`integer`*	Specifies the mode for tracing back. ■ 0 does not perform constant propagation ■ 1 performs tied-constant propagation ■ 2 performs tied-constant and test-mode propagation ■ 3 performs tied-constant, test-mode and shift-enable propagation *Default*: 3
`-continue`	Specifies to continue the trace back until a primary input, complex gate, or sequential gate is reached. Additionally, the trace will terminate if a logic constant is returned for the trace back pin.
`{`*`pin`*`|`*`port`*`}`	Specifies the pin or port from which to start the trace back.
`-polarity`	Specifies whether to report if the polarity changed through the trace. A returned value of 0 indicates no inversion. A returned value of 1 indicates an inversion.
`-print`	Prints the pin and polarity at every trace back.

Examples

- Following command traces back without performing constant propagation.

  ```
  rc:/> dft_trace_back -mode 0 /designs/top/instances_hier/g121/pins_in/CME
  /designs/top/ports_in/cme
  ```

- Following command traces back using the default mode. In this case a constant logic 1 value is reported.

  ```
  rc:/> dft_trace_back /designs/top/instances_hier/g121/pins_in/CME
  1
  ```

- Following command traces back using the default mode and requests to report whether there was a change in polarity. In this case, a constant logic 1 with no change in logic polarity is reported.

  ```
  rc:/> dft_trace_back /designs/top/instances_hier/g121/pins_in/CME -polarity
  1 0
  ```

fix_dft_violations

```
fix_dft_violations
     { -clock -test_control test_signal
       [-test_clock_pin {pin|port} [-rise | -fall]]
     | { -async_set | -async_reset
       | -async_set -async_reset }
       [ -async_control test_signal]
       -test_control test_signal
       [-insert_observe_scan
         -test_clock_pin {pin|port} [-rise | -fall]]}
     [-violations violation_object_id_list]
     [-tristate_net]
     [-xsource [-exclude_xsource instance...]]
     [-preview] [-dont_check_dft_rules] [-dont_map]
      [-design design]
```

Automatically fixes either

- *All* DFT violations of the specified types (clock, asynchronous set, asynchronous reset, tristate, or xsource).

  Only the specified violation types are fixed. If you allow to fix asynchronous set and reset violations using the same test mode signal, you can request both types to be fixed at the same time.

- The *identified* violations

  You can further limit the violations that RTL Compiler must fix to by specifying the violation ID (through the `-violations` option).

Note: Currently, clock violations are only fixed for the muxed scan style.

Options and Arguments

`-async_control` *`test_signal`*	
	Specifies the name of the test signal to use to control the asynchronous violations to be fixed.
`-async_reset`	Fixes the asynchronous reset violations on all instances.
`-async_set`	Fixes the asynchronous set violations on all instances.
`-clock`	Fixes the clock violations on all instances.

`-design` *`design`*
: Specifies the name of the design whose violations must be checked.

  If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dont_check_dft_rules`
: Prevents the DFT rules from being checked automatically after fixing violations.

`-dont_map`
: Prevents the inserted logic from being mapped even if the design is already mapped to the target library.

`-exclude_xsource` *`instance`*
: Specifies instances to exclude from automatic fixing of X-source violations.

`-insert_observe_scan`
: Inserts a flip-flop for observability. If you fix DFT violations in a generic netlist, the flip-flop is mapped to a scan flip-flop during synthesis. If you fix DFT violations in a mapped netlist, the flip-flop is mapped to a scan flip-flop. You need to rerun the `check_dft_rules` command to update the DFT status of the observability flop prior to connecting it in a scan chain.

  Note: This option can only be used when fixing an asynchronous set reset violation and requires the `-test_clock_pin` option.

`{`*`pin`*` | `*`port`*`}`
: Specifies the test signal pin or port to use to control the set or reset. By default, the set or reset are controlled by the `-test_control` option.

  To specify this option, the pin or port must first be declared as a `test_mode` signal using the `define_dft test_mode` constraint.

`-preview`
: Reports how the violations will be fixed, without making modifications to the netlist.

`[-rise | -fall]`
: Specifies to use the rising or falling edge of the test clock to fix the DFT violation during test-mode operation.

  Default: `-rise`

`-test_clock_pin {`*`pin`*` | `*`port`*`}`

Specifies the test clock signal to be used. Specify the pin or port from where the clock signal originates. In most applications, the clock pin or port is identified when checking the DFT rules.

This option is optional when fixing clock violations. By default, the RC-DFT engine performs a clock trace to identify a controllable test clock that appears in the fanin cone of the clock violation and uses this test clock to fix the actual clock violation.

This option is required when you want to insert observability flip-flops when fixing async violations. In this case, the test clock signal drives the clock pin of the observation flip-flops during test-mode operation.

`-test_control `*`test_signal`*

Specifies the particular test signal to use to fix the violation.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-tristate_net` Specifies to fix tristate net contention design rules violations.

`-violations `*`violation_object_id_list`*

Fixes the violations that are identified by their object name.

Note: The `check_dft_rules` command creates a `violations` directory in the design hierarchy under `/designs/`*`design`*`/dft/report`. The objects in this directory correspond to the violations found during the last execution of the `check_dft_rules` command. Use the `report dft_violations` command to list all remaining violations in the design.

`-xsource` Specifies to fix X-Source design rules violations.

Examples

- The following example instructs RTL Compiler to fix all clock, async set, and asynch reset violations using test mode signal `tm` and test clock `CK1` using the rising edge as the active edge.

```
fix_dft_violations -clock -async_set -async_reset
-test_control tm -insert_observe_scan -test_clock_pin CK1 -rise
```

If you do not want to use the same test clock to fix the clock violations and to drive the clock pin of the observation flip-flops, you need to enter two commands. For example,

```
fix_dft_violations -clock -test_control tm
```

```
fix_dft_violations -async_set -async_reset -test_control tm
-insert_observe_scan -test_clock_pin CK1 -rise
```

In this case, the RC-DFT engine automatically determines which test clock to use to fix the clock violations.

- The following example instructs RTL Compiler to fix all clock, async set, and asynch reset violations using test mode signal `tm` and test clock `CK1` using the rising edge as the active edge.

```
fix_dft_violations -clock -async_set -async_reset
-test_control tm -test_clock_pin CK1 -rise
```

- The following example instructs RTL Compiler to fix violations `vid_1` and `vid_3` if they are violations of type `async_set`.

```
fix_dft_violations -violations {vid_1 vid_3} -async_set -test_control tm
```

Related Information

Fixing DFT Rule Violations in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft shift_enable on page 572
	define_dft test_clock on page 577
	define_dft test_mode on page 581
Affects these commands:	check_dft_rules on page 501
	report dft_registers on page 648
	report dft_violations on page 650
	synthesize on page 294
Sets these attributes:	dft_status
	dft_violation
	Violations Attributes

fix_scan_path_inversions

`fix_scan_path_inversions` *`actual_scan_chain`*`...`

Fixes inversions for every scan element in the scan path. The command inserts inverters as required in a scan chain to reset the flops in the chain to `logic_0`.

Options and Arguments

`actual_scan_chain`	Specifies the scan chain(s) to undergo analysis and to insert inverters.

Related Information

Fixing Scan Path Inversions in *Design for Test in Encounter RTL Compiler*

identify_multibit_cell_abstract_scan_segments

```
identify_multibit_cell_abstract_scan_segments
     [-libcell libcell]... [-preview]
     [-dont_check_dft_rules] [-design design]
```

Identifies multi-bit scan cells in the design, and defines scan abstract segments for each instance of the multi-bit scan cell.

Multi-bit scan cells implemented using either a parallel or serial bit approach are supported by the tool.

Options and Arguments

`-design` *`design`*	Specifies the name of the top-level design on which to identify abstract segments for multi-bit scan cells. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-dont_check_dft_rules`	
	Prevents the DFT rules from being automatically checked after identifying abstract segments for multi-bit scan cells.
`-libcell {`*`cell...`*`}`	Specifies the multi-bit library cells on which to perform abstract segment identification.
`-preview`	Reports the identified abstract scan segments without defining them.

Examples

- The following example identifies a parallel multi-bit scan cell with two bits; where each bit would be defined as an abstract scan segment:

  ```
  rc:/> identify_multibit_cell_abstract_scan_segments -preview
  ```

  Would execute command:

  ```
  define_dft abstract_segment -length 1 -sdi SI1 -sdo Q1
  -shift_enable_port SE1 -active high -clock_port CP -rise
  -libcell/libraries/tcbn65ulp_c070701wc2/libcells/DUALSDFQD0 -name DUALSDFQD0
  ```

  Would execute command:

  ```
  define_dft abstract_segment -length 1 -sdi SI2 -sdo Q2
  -shift_enable_port SE2 -active high -clock_port CP -rise
  -libcell /libraries/tcbn65ulp_c070701wc2/libcells/DUALSDFQD0 -name DUALSDFQD0
  ```

- The following example identifies a serial multi-bit scan cell of length 4; where each instance of the cell would be defined as an abstract scan segment:

  ```
  rc:/> identify_multibit_cell_abstract_scan_segments -preview
  ```

  Would execute command:

  ```
  define_dft abstract_segment -length 4 -sdi SI -sdo Q4
  -shift_enable_port SM -active high -clock_port CK -rise
  -libcell /libraries/cs60ale_uc_scan/libcells/YSDM4ALU1 -name YSDM4ALU1
  ```

Related Information

Mapping to Multi-Bit Scan Cells in *Design for Test in Encounter RTL Compiler*

identify_shift_register_scan_segments

```
identify_shift_register_scan_segments
     [-min_length integer] [-max_length integer]
     [-preview] [-incremental]
```

Identifies all shift registers in the design whose minimum length either satisfies the default minimum length, or the specified minimum and maximum length values.

The RC-DFT engine uses the following naming convention for automatically identified shift-register segments:

`DFT_AutoSegment_n`

You can find the automatically identified shift-register segments in:

`/designs/top_design/dft/scan_segments`

Note: A shift register segment contains flops driven by the same clock and same clock edge.

A shift register is a scan segment which can be associated with either

- A user-defined top-level chain—created using the `define_dft scan_chain` command
- A tool-created scan chain—created using the `connect_scan_chains` command

Options and Arguments

`-incremental`

Adds new shift register segments in incremental mode, without changing already identified shift register segments stored in `/designs/design/dft/scan_segments`

Note: Without this option, the existing identified shift register segments will be removed and new segments will be identified based on the new values specified for the command options.

`-max_length integer`

Specifies the maximum length that an automatically identified shift register can have.

If the length of a functional shift register exceeds this length, it will be broken in multiple scan segments.

Note: There is no default for the maximum length.

`-min_length` *`integer`*	
	Specifies the minimum length a shift register must have to be automatically identified by the tool.
	Default: 2
`-preview`	Reports which shift register segments will be identified, without adding them to the list of scan segments.

Related Information

Identifying Shift Registers in the Design in *Design for Test in Encounter RTL Compiler*

Identifying Shift Registers in a Mapped Netlist before Creating the Scan Chains in *Design for Test in Encounter RTL Compiler*

Sets this attribute:	user_defined_segment

identify_test_mode_registers

```
identify_test_mode_registers
     -stil file
     [-macro string]
     [-library string]
     [-design design] [-preview] [-et_log string]
```

Identifies all internal registers whose output signals must remain constant during test mode and generates the corresponding test-mode signals required for the RC-DFT engine.

Note: To use this command you need to have the Encounter Test software installed and your operating system `PATH` environment variable must include the path to the Encounter Test software. For more information on the exact product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Options and Arguments

`-design` *design*	Specifies the name of the design for which to define the test-mode signals. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-et_log` *string*	Specifies log file for Encounter Test.
`-library` *string*	Specifies the list of Verilog structural library files. Specify the list in a quoted string. Refer to the `write_et_atpg` `-library` option description for additional information. **Note:** This option is only required when you invoke this command on a mapped netlist.
`-macro` *string*	Specifies the name of the macro in the STIL file that contains the initialization vectors to be simulated. *Default*: `test_setup`
`-preview`	Reports the fixed value registers but does not set the test mode values.
`-stil` *file*	Specifies the name of the STIL file that contains the mode initialization sequence.

Examples

- The following command requests a report of the list of the registers with fixed values.

```
rc:/> identify_test_mode_registers -stil newStil -prev

Identifying internal registers with fixed value outputs under test_mode setup.
... Creating intermediate files

WARNING : No user defined shift enable signal found.
ATPG interface file may contain incomplete information
Cadence Design Systems RC file: Cadence ATPG file created successfully.

-----------------------------------------
... Identifying the fixed value registers
-----------------------------------------

Test Function    Block Name
+TI              tc/ts_reg[0]
+TI              tc/ts_reg[1]
+TI              tc/ts_reg[2]
-----------------------------------------
Note:  The +/-TI Test Function flag denotes the active logic value that a signal
is to be held to during test mode.
       +TI denotes a logic value 1; -TI denotes a logic value 0
```

- The following command creates the test signals and automatically reruns the DFT rule checker.

```
rc:/> identify_test_mode_registers -stil newStil
Identifying internal registers with fixed value outputs under test_mode setup.
... Creating intermediate files

WARNING : No user defined shift enable signal found.
ATPG interface file may contain incomplete information
Cadence Design Systems RC file: Cadence ATPG file created successfully.

-----------------------------------------
... Identifying the fixed value registers
-----------------------------------------
INFO: Setting active high test mode signal on tc/ts_reg[0]/q
INFO: Setting active high test mode signal on tc/ts_reg[1]/q
INFO: Setting active high test mode signal on tc/ts_reg[2]/q
-----------------------------------------
  Checking DFT rules for 'top' module under 'muxed_scan' style

...
rc:/> ls dft/test_signals
/designs/top/dft/test_signals:
./                             rst                            tc__ts_reg[1]__q
incr                           tc__ts_reg[0]__q               tc__ts_reg[2]__q
```

Related Information

Identifying Fixed-Value Registers in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft test_mode on page 581
	define_dft test_clock on page 577
Sets this attribute:	user_defined_signal

insert_dft

```
insert_dft
     { analyzed_test_points |  boundary_scan | dfa_test_points
     | jtag_macro  | lockup_element | mbist | ptam
     | scan_power_gating | shadow_logic | test_point
     | user_test_point | wrapper_cell }
```

Inserts DFT test logic.

Options and Arguments

`analyzed_test_points`	Inserts test points selected by Encounter Test or adds shadow logic.
`boundary_scan`	Inserts boundary scan cells and the corresponding JTAG controller.
`dfa_test_points`	Inserts test points based on Deterministic Fault Analysis.
`jtag_macro`	Inserts a JTAG Macro controller into a netlist.
`lockup_element`	Inserts lockup elements in the specified analyzed scan chains.
`mbist`	Inserts Memory Built-In-Self-Test (MBIST) logic to test targeted memories in the design.
`ptam`	Inserts Power Test Access Mechanism (PTAM) control logic into the design.
`scan_power_gating`	Inserts gating logic at selected flop outputs to minimize switching power during scan shift
`shadow_logic`	Inserts DFT shadow logic to enable testing of shadow logic around a module.
`test_point`	Inserts a native test point.
`user_test_point`	Inserts a user-defined test point.
`wrapper_cell`	Inserts an IEEE-1500 style core-wrapper cell.

Related Information

Related commands:	insert_dft analyzed_test_points on page 600
	insert_dft boundary_scan on page 607
	insert_dft dfa_test_points on page 611
	insert_dft jtag_macro on page 614
	insert_dft lockup_element on page 617
	insert_dft mbist on page 618
	insert_dft ptam on page 622
	insert_dft scan_power_gating on page 625
	insert_dft shadow_logic on page 627
	insert_dft test_point on page 631
	insert_dft user_test_point on page 636
	insert_dft wrapper_cell on page 638

insert_dft analyzed_test_points

```
insert_dft analyzed_test_points
     { {-input_tp_file file
       | [-atpg [-atpg_effort {low|medium|high}]
         [-atpg_options string]
         [-build_model_options string]
         [-build_testmode_options string]]
         [-rrfa_effort {low|medium|high}]
         [-rrfa_options string]
         -directory string [-et_log file] [-verbose]
         [-output_tp_file file] }
       [-max_number_of_testpoints integer ]
       [-min_slack integer]
       [-share_observation_flop integer]
       [-library string]
     | -shadow_logic [-minimum_shadow_logic_pins integer]
       [-exclude_shadow_logic instance]...}
     [-test_control test_signal]
     [-test_clock_pin {port|pin}][design]
```

This command either

- Automatically inserts shadow logic around blackboxes and timing models (by adding observable flops in non-share mode)
- Invokes Encounter Test to
  - Perform Automatic Test Pattern Generator (ATPG) based testability analysis to prune out the ATPG detectable faults (if the `-atpg` option is selected)

    Note: Choosing the `-atpg` option does affect the runtime.
  - Perform Random Resistance Fault Analysis (RRFA) based testability analysis and test-point selection
  - Insert selected test points that have minimal impact on the slack

  Note: You need to have the Encounter Test software installed and your operating system PATH environment variable must include the path to the Encounter Test software. For more information on the exact product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Tip

The `insert_dft analyzed_test_points` command is recommended to be run with the design in default timing mode. Ensure that the design is in default timing mode before running this command by running the following commands:

```
set default_mode [filter default true [find / -mode *]]   // to retrieve the
default timing mode

report timing -mode $default_mode
```

Options and Arguments

`-atpg`	Runs ATPG-based testability analysis to prune the ATPG detectable faults before running random-resistant fault analysis.

`-atpg_effort {low | medium | high}`

Specifies the effort to be used for the ATPG-based testability analysis.

Default: `low`

`-atpg_options` *`string`*

Specifies extra options to run ATPG-based testability analysis in a string.

For more information on these options, refer to the `create_tests` command in the *Command Line Reference* (of the Encounter Test documentation).

Note: This option is mutually exclusive with the `-input_tp_file` option.

`-build_model_options {`*`option1=value option2=value`*`}`

Specifies extra options to apply when building the Encounter Test model.

Note: For more information on these options, refer to the `build_model` command in the *Command Line Reference* (of the Encounter Test documentation).

`-build_testmode_options {`*`option1=value option2=value`*`}`

Specifies extra options to apply when building the test mode for Encounter Test.

Note: For more information on these options, refer to the `build_testmode` command in the *Command Line Reference* (of the Encounter Test documentation).

`design`

Specifies the name of the top-level design on which you want to perform test analysis and test-point selection.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-directory` *`string`*

Specifies the working directory for Encounter Test.

Note: This option is only required when you run an RRFA-based analysis

`-et_log` *`file`*

Specifies the name of the Encounter Test log file. This file will be generated in the specified directory.

Default: `eta_from_rc.log`

`-exclude_shadow_logic {`*`instance...`*`}`

Excludes automatic shadow logic insertion for the specified instances. You can only specify instances of blackboxes or timing models.

`-input_tp_file` *`file`*

Specifies the name of the file containing the test point locations.

The file is specified in Encounter Test format.

If you do not specify this option, the test point locations are read from

- The file specified with the `-output_tp_file` option if you also specified the `-rrfa` option
- The following file in the working directory if you did not specify any file:
  `TB/testresults/TestPointInsertion.ASSUMESCAN.expt.`

Note: This option is mutually exclusive with `-atpg_options` and `-rrfa_options`.

`-library` *`string`*

Specifies the list of Verilog structural library files. Refer to the `write_et-atpg` `-library` option description for additional information.

Note: This option is only required when you invoke this command on a mapped netlist.

`-max_number_of_testpoints` *`integer`*

Specifies the number of test points to be inserted.

You must specify an integer value greater than `0` when attempting to insert test points from a file using the `-input_tp_file` option.

Default: `0`

`-min_slack` *`integer`*

Limits the insertion of a test point to those nodes that have the specified minimum slack (in ps).

Default: `2000`

`-minimum_shadow_logic_pins` *`integer`*

Limits shadow logic insertion to blackboxes or timing models that have more pins (inputs and outputs) than the specified value.

Default: `10`

`-output_tp_file` *`file`*

Specifies the output file generated by the RRFA-based analysis.

If you do not specify this option, the test point locations are written to the following file in the working directory: `TB/testresults/TestPointInsertion.ASSUMESCAN.expt`.

`-rrfa_effort {low | medium | high}`

Specifies the effort to be used for the RRFA-based analysis.

Default: `low`

`-rrfa_options` *`string`*

Specifies the extra options to run RRFA-based testability analysis in a string.

For more information on these options, refer to the `analyze_random_resistance` command in the *Command Line Reference* (of the Encounter Test documentation).

Note: This option is mutually exclusive with the `-input_tp_file` option.

`-shadow_logic` Automatically inserts registered (`no_share`) shadow logic around blackboxes and timing models. Test points are added for uni-directional pins only. Bidirectional pins and pins associated with test clock objects are skipped.

`-share_observation_flop` *`integer`*

Specifies the number of observation test nodes that can share an observation flop through an XOR tree.

Default: `1`

`-test_clock_pin {`*`port`*` | `*`pin`*`}`

Specifies the test clock that drives the clock pin of the inserted test points during test mode operation. Specify a port or pin that drives the test clock.

`-test_control` *`test_signal`*

Specifies the test signal to use to control the test points.

Note: You must have specified the test signal using the `define_dft test_mode` constraint.

`-verbose` Specifies to print test point details

Examples

- The following example performs only RRFA-based testability analysis and creates a file `myfile` with the suggested test point locations.

```
insert_dft analyzed_test_points -output_tp_file myfile
```

- The following example inserts10 test points from the specified test point file `myfile`.

```
insert_dft analyzed_test_points -max_number_of_testpoints 10 \
-test_control tm -test_clock_pin clk -input_tp_file myfile
```

- The following example performs ATPG-based testability analysis followed by RRFA-based testability analysis. The command generates a report on the fault coverage in the log file, and stores the suggested test point locations in the `TB/testresults/TestPointInsertion.ASSUMESCAN.expt` file.

  ```
  insert_dft analyzed_test_points -atpg
  ```

- The following example performs ATPG-based testability analysis, RRFA-based testability analysis, and inserts10 test points. During RRFA-based testability analysis, test point locations are written to the `TB/testresults/TestPointInsertion.ASSUMESCAN.expt` file, while during test point insertion, they are read from this file.

  ```
  insert_dft analyzed_test_points -atpg -max_number_of_testpoints 10 \
  -test_control tm -test_clock_pin clk
  ```

- The following example automatically inserts shadow logic around all blackboxes.

  ```
  rc:/> insert_dft analyzed_test_points -shadow_logic -test_control my_tm \
   -test_clock CK
  WARNING: pin 'A_CNTR/RAM/CK' is skipped from shadow DFT insertion since it is
  driven by a clock

  Total number of test points inserted: 76

  Mapping shadow DFT logic...
  ..
  Mapping DFT logic done.
  WARNING: pin 'S_CORE/ID/LOCAL_RAM/CK' is skipped from shadow DFT insertion
  since it is is driven by a clock




  WARNING: bidirectional pin 'S_CORE/ID/LOCAL_RAM/VI[31]' skipped from shadow
  DFT insertion.
  ...
  ...
  ...
  Total number of test points inserted: 80

  Mapping shadow DFT logic...
  ..
  Mapping DFT logic done.
  WARNING: pin 'S_CORE/IK/IDA/REAL_RAM/CK' is
  skipped from shadow DFT insertion since it is is driven by a clock
  ...
  ...
  ...
  Total number of test points inserted: 34

  Mapping shadow DFT logic...
  ..
  Mapping DFT logic done.
  ```

Related Information

Inserting DFT Shadow Logic in *Design for Test in Encounter RTL Compiler*

Using Encounter Test to Automatically Select and Insert Test Points in *Design for Test in Encounter RTL Compiler.*

Affected by these constraints:	define_dft test_mode on page 581
	define_dft test_clock on page 577

insert_dft boundary_scan

```
insert_dft boundary_scan [-design design]
     [-comp_enables_high port [port]...]
     [-comp_enables_low port [port]...]
     [-exclude_ports port [port]...]
     [-functional_clocks port [port]...]
     [-custom_cell_directory string]
     [-inside instance]
     [[-pinmap_file file] | [-physical]] [-power_on_reset pin|port]
     [-tck port] [-tdi port] [-tdo port]
     [-tms port] [-trst port]
     [-dont_map] [-preview] [-preserve_tdo_connection]
```

Inserts boundary scan cells and the corresponding JTAG Macro (if it is not yet instantiated in the design).

Note: You must have a license for the Encounter Test Architect tool to use this command.

Options and Arguments

`-comp_enables_high (-comp_enables_low)` *`port...`*

Specifies that the compliance value for the specified ports is active high (low) during functional mode. The compliance enable value is the value that a test port (test mode, shift enable) is tied to during the functional mode of the chip.

The ports with their compliance value are added to a BSDL `COMPLIANCE_PATTERNS` statement.

`-custom_cell_directory` *`string`*

Specifies the path to the directory that contains the files describing the custom boundary cells. Each cell must be described as a Verilog module in its own file. The basename of the file must match the name of the cell in the module description.

`-design` *`design`*

Specifies the name of the design in which you want to insert boundary scan logic. This option is required if you have loaded multiple designs.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dont_map`

Prevents the inserted boundary scan logic from being mapped to technology gates even if the design is already mapped.

`-exclude_ports` *`ports`*

Excludes the specified ports from being considered for boundary scan logic insertion.

`-functional_clocks` *`ports`*

Specifies the ports that are seen as clocks for ATPG. These ports include

- async set and reset ports
- clocks used in functional mode, but not in scan shift mode

`-inside` *`instance`*

Specifies a hierarchical instance into which to insert the `JTAG_MACRO`.

`-physical`

Specifies to build the boundary scan register using physical information to minimize its wire length.

The physical placement information is obtained from the DEF file read in with the `read_def` command

Note: This option is mutually exclusive to the `-pinmap` option.

`-pinmap_file` *`file`*

Specifies the name of the file containing the mapping between the design ports and the actual package pins. This mapping is also used to determine the boundary scan order.

Refer to Pinmap File Format for more information.

Note: This option is mutually exclusive to the `-physical` option.

`-power_on_reset`

Specifies the power-on-reset pin which will be connected to the `JTAG_POR` input pin on the `JTAG_Macro` subdesign. This connection is made when the boundary scan logic is inserted in the design.

`-preserve_tdo_connection`

Preserves the existing net connection to from-core and tristate enable pins of the TDO pad cell.

If you do not specify this option, the existing net connections will be broken and new net connections will be made during boundary scan insertion from the `JTAG_TDO` and `JTAG_ENABLE_TDO` pins to the from-core and tristate enable pins of the TDO pad cell, respectively.

Note: If you preserve the TDO connections, such that the net is driven by user logic other than a pre-instantiated JTAG macro, boundary scan insertion will insert a JTAG macro and leave its `JTAG_TDO` pin unconnected in the netlist.

`-preview` Shows the potential changes, without making any modifications to the netlist.

`-tck` *`port`* Specifies the port name of the driver for the test clock of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Default: `TCK`

`-tdi` *`port`* Specifies the port name of the driver for the test data (scan) input of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Default: `TDI`

`-tdo` *`port`* Specifies the port name of the test data (scan) output of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Note: An existing TDO port must have a tristate I/O pad.

Default: `TDO`

`-tms` *`port`* Specifies the port name of the test mode select input of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Default: `TMS`

`-trst` *`port`* Specifies the port name of the (asynchronous) test reset of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Default: `TRST`

Related Information

Inserting Boundary Scan Logic in *Design for Test in Encounter RTL Compiler*

Related constraints:	define_dft jtag_instruction_register on page 547
	define_dft jtag_instruction_register on page 547
	define_dft shift_enable on page 572
	define_dft test_clock on page 577
	define_dft test_mode on page 581
Affects these commands:	compress_scan_chains on page 508
	insert_dft mbist on page 618
	insert_dft ptam on page 622
Sets these attributes:	boundary_type
	Jdft_jtag_macro_exists
	JTAG Port Attributes

insert_dft dfa_test_points

```
insert_dft dfa_test_points -input_tp_file string
     [-max_number_of_testpoints integer]
     [-min_slack integer]
     [-fault_threshold integer]
     [-test_control test_signal]
     -test_clock_pin {pin|port}
     -share_observation_flop integer]
     [-verbose] [design]
```

This command will insert selected test points identified by Encounter Test Deterministic Fault Analysis. Test points can be inserted that have minimal impact on the slack and which target a minimum specified fault count.

Tip

The `insert_dft dfa_test_points` command is recommended to be run with the design in default timing mode. Ensure that the design is in default timing mode before running this command by running the following commands:

```
set default_mode [filter default true [find / -mode *]]   // to retrieve the
default timing mode

report timing -mode $default_mode
```

Options and Arguments

design	Specifies the name of the top-level design on which you want to perform test analysis and test-point selection. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-fault_threshold` *integer*	Specifies to insert only those test point locations whose fault count is greater than or equal to the number specified. *Default*: `0`
`-input_tp_file` *file*	Specifies the name of the file containing the test point locations. The file is specified in Encounter Test format.

`-max_number_of_testpoints` *`integer`*

Specifies the number of test points to be inserted.

If this option is not specified, then all the test points from the file specified with the `-input_tp_file` option will be processed for insertion.

Default: All

`-min_slack` *`integer`* Limits the insertion of a test point to those nodes that have the specified minimum slack (in ps).

Default: `2000`

`-share_observation_flop` *`integer`*

Specifies the number of observation test nodes that can share an observation flop through an XOR tree.

Default: `1`

`-test_clock_pin {`*`port`*` | `*`pin`*`}`

Specifies the test clock that drives the clock pin of the inserted test points during test mode operation. Specify a port or pin that drives the test clock.

`-test_control` *`test_signal`*

Specifies the test signal to use to control the test points.

Note: You must have specified the test signal using the `define_dft test_mode` constraint.

`-verbose` Specifies to print test point details

Examples

- The following example inserts the first 500 test points whose fault count is greater than or equal to 3 from the specified test point file `myfile`.

```
insert_dft dfa_test_points -max_number_of_testpoints 500 \
 -fault_threshold 3 -test_control tm -test_clock_pin clk  -input_tp_file myfile
```

- The following example inserts all test points for test nodes with a minimum slack of 3ps, and will share 3 observation test nodes through an XOR tree from the specified test point file `myfile`.

```
insert_dft dfa_test_points -min_slack 3000 -share_observation_flop 3 \
-test_control tm -test_clock_pin clk -input_tp_file myfile
```

Related Information

Using Encounter Test to Perform a Deterministic Fault Analysis on a Scan Connected Netlist in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft test_clock on page 577
	define_dft test_mode on page 581

insert_dft jtag_macro

```
insert_dft jtag_macro [-design design]
     [-dont_map] [-inside instance] [-insert_without_pad_logic]
     [-preserve_tdo_connection] [-create_ports]
     [-tck port] [-tdi port] [-tdo port]
     [-tms port] [-trst port] [-power_on_reset pin|port]
     [-boundary_type IEEE_11491|IEEE_11496]
```

Inserts a JTAG Macro (if one has not already been defined).

Note: You must have a license for the Encounter Test Architect tool to use this command.

Options and Arguments

`-boundary_type {IEEE_11491 | IEEE_11496}`	
	Specifies the boundary scan architecture for which to generate the inserted JTAG Macro. *Default:* `IEEE_11491`
`-create_ports`	Specifies whether to create the TAP ports if they do not exist. If you do not specify the TAP signals using the `-tdi`, `-tdo`, `-tms`, `-trst`, and `-tck` options, the ports can be created and named as *`prefix`*`_tdi`, *`prefix`*`_tdo`, *`prefix`*`_tms`, *`prefix`*`_trst`, and *`prefix`*`_tck`, where *`prefix`* is the value of the root-level `dft_prefix` attribute. An additional port, `prefix_tdo_enable` will be created. This port is the enable signal used to control the tristate pin of the top-level TDO pad. This option must be used in conjuction with the `-insert_without_pad_logic` option.
`-design` *`design`*	Specifies the name of the design in which you want to insert the JTAG Macro. This option is required if you have loaded multiple designs. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-dont_map`	Prevents the inserted JTAG Macro from being mapped to technology gates even if the design is already mapped.
`-inside` *`instance`*	Specifies a hierarchical instance into which to insert the JTAG Macro.

`-insert_without_pad_logic`

Specifies whether to insert the JTAG Macro into a design that does not have pad logic

`-power_on_reset {`*`pin`*` | `*`port`*`}`

Specifies the power-on-reset pin name.

`-preserve_tdo_connection`

Preserves the existing net connection to from-core and tristate enable pins of the TDO pad cell.

If you do not specify this option, the existing net connections will be broken and new net connections will be made from the JTAG Macro `JTAG_TDO` and `JTAG_ENABLE_TDO` pins to the from-core and tristate enable pins of the TDO pad cell, respectively.

Note: If you preserve the TDO connections, such that the net is driven by user logic other than the JTAG Macro, the `JTAG MacroJTAG_TDO` pin will be left unconnected in the netlist.

`-tck` *`port`*

Specifies the port name of the driver for the test clock of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.

Default: `TCK`

`-tdi` *`port`*

Specifies the port name of the driver for the test data (scan) input of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.

Default: `TDI`

`-tdo` *`port`*

Specifies the port name of the test data (scan) output of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.

Note: An existing TDO port must have a tristate I/O pad.

Default: `TDO`

`-tms` *`port`*

Specifies the port name of the test mode select input of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.

Default: `TMS`

`-trst` *`port`*	Specifies the port name of the (asynchronous) test reset of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name. *Default*: TRST

Examples

- The following example inserts the JTAG Macro into the top-level netlist, creates TAP ports on the top-level netlist, and preserves the existing net connection to from-core and tristate enable pins of the TDO pad cell.

```
insert_dft jtag_macro -create_tap_ports design=design_name \
 -preseve_tdo_commection -insert_without_pad_logic
```

- The following example inserts the JTAG Macro into a specified hierarchical instance and connects it to TAP ports without pad logic.

```
insert_dft jtag_macro -inside instance_level design=design_name \
 -insert_without_pad_logic
```

Related Information

Inserting JTAG Macro Logic in *Design for Test in Encounter RTL Compiler*

Related constraints:	define_dft jtag_instruction on page 543
	define_dft jtag_instruction_register on page 547
Affects these commands:	compress_scan_chains on page 508
	insert_dft mbist on page 618
	insert_dft ptam on page 622
Sets this attribute:	JTAG_PORT Attributes
	Jdft_jtag_macro_exists

insert_dft lockup_element

```
insert_dft lockup_element
     actual_scan_chain [actual_scan_chain]...
     [-terminal_lockup]
```

Inserts a lockup element as needed in the specified analyzed scan chains.

Analyzed scan chains are chains whose connectivity is traced by the RC-DFT engine when you define them using the `-analyze` option of the `define_dft scan_chain` command.

Use this command on mapped netlists whose existing (configured) scan chains were either created by hand or using a third-party DFT insertion tool.

Tip

The RC-DFT engine determines the type of lockup element to be inserted from the value of the `dft_lockup_element_type` design attribute.

Options and Arguments

`actual_scan_chain`	Specifies the name of an analyzed scan chain.
`-terminal_lockup`	Allows to add a terminal lockup element to the specified analyzed scan chains.

Example

The following example inserts lockup elements as needed in all analyzed scan chains.

```
insert_dft lockup_element [filter analyzed true \
[find /designs/design/dft -actual_scan_chain *]]
```

Related Information

Analyzing Chains in a Scan-Connected Netlist in *Design for Test in Encounter RTL Compiler*

Affected by this attribute: dft_lockup_element_type

insert_dft mbist

```
insert_dft mbist
     [-config_file config_file] [-connect_to_jtag] [-dont_create_jtag_ports]
     [-test_control test_signal] [-interface_file_dirs string]
     [-dont map] [-dont_check_dft_rules] [diagnose_mbist string]
     [-run_mbist string] [raa_mbist string] [repair_mbist string]
     [-dont_check_mbist_rules] [-read_mbist string]
     [-continue_mbist string] [-diagnose_rombist string]
     [-instance string] [-module_prefix string] [-directory dir_path]
     [-preview]
```

Inserts Memory Built-In-Self-Test (MBIST) logic to test targeted memories in the design.

Options and Arguments

`-config_file`	Specifies the file that contains the user-defined configuration parameters which selects the MBIST features and controls for the insertion of the test logic for targeted memories. This option can be used in conjunction with the `-interface_file_dirs` option as part of a bottom-up flow.
`-connect_to_jtag`	Connects tap pins to the inserted MBIST logic.
`-continue_mbist` *`string`*	Specifies the `CONTINUE_MBIST` instruction as configured in the IEEE 1149.x information hierarchy for the design. *Default:* `CONTINUE_MBIST`
`-diagnose_mbist` *`string`*	Specifies the `DIAGNOSE_MBIST` instruction as configured in the IEEE 1149.x information hierarchy for the design. *Default:* `DIAGNOSE_MBIST`
`-diagnose_rombist` *`string`*	Specifies the `DIAGNOSE_ROMBIST` instruction as configured in the IEEE 1149.x information hierarchy for the design. *Default:* `DIAGNOSE_ROMBIST`

`-directory` *`directory_path`*

Specifies the directory to which the interface files are written for subsequent input to other commands, including `insert_dft mbist` in support of bottom-up design flows.
Default: `current_working_directory/``mbist`

`-dont_check_dft_rules`

Prevents the DFT rules from being automatically checked after MBIST insertion.

`-dont_check_mbist_rules`

Prevents the MBUST rules from being automatically checked after MBIST insertion.

`-dont_create_jtag_ports`

Prevents creation of MBIST control ports for the TAP at the top level instance specified by the `-instance` option.

`-dont_map` Prevents the inserted logic from being mapped even if the design is already mapped to the target library.

`-instance` *`string`* Specifies the name of the top level instance.

`-interface_file_dirs`

Specifies a list of interface file directories where MBIST pattern control files are written from a block level execution of this command. In support of bottom-up design flows, these files represent the block being inserted into the larger design processed in this execution of the command. Separate the directory names with blank spaces.

`-module_prefix` *`string`*

Specifies an additional character string to append to the default prefix `tem`. The string is placed on all modules created and inserted by the `insert_dft mbist` command.

`-preview` Specifies to create a configuration file template without performing insertion. The template file can be used to review configuration file content or as a basis to further edit content.

Note: This option cannot be used with the `-config_file` option.

`-raa_mbist` *`string`*	Specifies the `RAA_MBIST` instruction as configured in the IEEE 1149.x information hierarchy for the design. *Default:* `RAA_MBIST`
`-read_mbist` *`string`*	Specifies the `READ_MBIST` instruction as configured in the IEEE 1149.x information hierarchy for the design. *Default:* `READ_MBIST`
`-repair_mbist` *`string`*	Specifies the `REPAIR_MBIST` instruction as configured in the IEEE 1149.x information hierarchy for the design. *Default:* `REPAIR_MBIST`
`-run_mbist` *`string`*	Specifies the `RUN_MBIST` instruction as configured in the IEEE 1149.x information hierarchy for the design. *Default:* `RUN_MBIST`
`-test_control` *`test_signal`*	Designates a test-control test signal, held at a constant value during a test session. This signal must be at a value of zero during memory BIST operations.

Examples

- The following command inserts the MBIST logic as described in the configuration file, by default, to the design module at the highest level of hierarchy of the design.

  ```
  insert_dft mbist -config_file ../et_inputs/my_configuration.txt
  ```

- The following command inserts the MBIST logic as described in the configuration file, into the design module `chip_top` of the design.

  ```
  insert_dft mbist -config_file  ./et_inputs/my_configuration.txt\
  -design chip_top
  ```

- The following command inserts the MBIST logic as described in the configuration file, into design module `chip_top` placing test mode initialization files and pin assign files into a specified directory.

  ```
  insert_dft mbist -config_file ../et_inputs/my_configuration.txt \
  -design chip_top \
  -directory ./my_et_files
  ```

Related Information

Inserting Memory Built-In-Self-Test Logic in *Design for Test in Encounter RTL Compiler*

Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	define_dft test_mode on page 581
Affects these commands:	insert_dft boundary_scan on page 607
	insert_dft jtag_macro on page 614
Related commands:	write_et_atpg on page 669
	write_et_bsv on page 674
	write_et_mbist on page 682
	write_et_rrfa on page 687

insert_dft ptam

```
insert_dft ptam
     -power_test_enable {pin|port}
     [-power_test_enable_active {low|high} ]
     [-shift_enable test_signal ]
     [-instruction string]
     [-connect_to_jtag]
     [-directory string ] [-preview] [-dont_map ] [-dont_check_dft_rules ]
```

Inserts Power Test Access Mechanism (PTAM) logic to facilitate chip power management during test.

Note: Some files may require customization according to the setup requirements.

Options and Arguments

`-connect_to_jtag`

Connects the PTAM logic to the JTAG macro instance. The JTAG macro instance must first be created using either the `insert_dft boundary_scan` command or the `insert_dft jtag_macro` command. If `-connect_to_jtag` is not specified and a JTAG macro instance is detected in the current session, a warning message will be issued to notify of the JTAG macro instance's existence, otherwise there will be no attempt to connect to a JTAG macro instance.

`-directory` *`string`*

Specifies the directory to which the mode initialization and pin assign files are written.

Default: *`current_working_directory`*/`ptam`

`-dont_check_dft_rules`

Prevents the DFT rules from being automatically checked after PTAM insertion.

`-dont_map`

Prevents the inserted logic from being mapped even if the design is already mapped to the target library.

`-instruction` *`string`*

Specifies the name of the instruction which must be loaded into the IEEE 1149.x TAP controller instruction register to access the PTAM test data register.

`-power_test_enable {`*`pin`*` | `*`port`*`}`

Identifies the power-test-enable pin or port. Asserting this pin to an active state will enable the PTAM logic to override the design's power manager control pins. This serves as a master mode control signal.

Note: This cannot be the same pin or port identified by `define_dft test_mode` that control pin sharing if the PTAM I/O are shared

`-power_test_enable_active {high |low}`

Specifies the active value for the power-test-enable pin or port.

Default: `high`

`-preview`

Shows the potential changes without making any modifications to the netlist.

`-shift_enable `*`test_signal`*

Designates a shift-enable test signal used to override the PTAM gating logic of the power manager output control signals during scan test.

If you omit this option, the default shift-enable signal specified using `define_dft shift_enable` is used as selected by its `default_shift_enable` attribute.

Examples

- The following command inserts the PTAM logic into design `chip_top.`

```
insert dft ptam -instruction PTAM -power_test_enable /chip_top/PwrTe \
-directory ${workdir}/testmode_data
```

Related Information

Inserting Power Test Access Mechanism (PTAM) Logic in *Design for Test in Encounter RTL Compiler*

Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler*

Affected by these commands:	read_cpf on page 760
	define_dft shift_enable on page 572
	create_isolation_rule in the *Common Power Format Language Reference*
	create_power_domain in the *Common Power Format Language Reference*
	create_state_retention_rule in the *Common Power Format Language Reference*

insert_dft scan_power_gating

```
insert_dft scan_power_gating
     -max_number_of_testpoints integer
     [-min_slack integer] -test_control test_signal | [-preview]
     [-report_virtual_scan_power integer... [-preview]]
     [-dont_check_dft_rules] [-input_tp_file file]
     [-output_tp_file file] [design]
```

Inserts gating logic at selected flop outputs to minimize switching power during scan shift. This command must be run prior to building the actual scan chains in the design.

`design`

Specifies the name of the top-level design on which to perform test point insertion.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dont_check_dft_rules`

Prevents the DFT rules from being automatically checked after inserting scan power gating.

`-input_tp_file` *`file`*

Specifies the name of the file containing the test point locations. The file is specified in Encounter Test format.

If you do not specify this option, the test point locations are read from the following:

- The file specified with the `-output_tp_file` option
- The following file in the working directory if you did not specify any file:
  `TB/testresults/TestPointInsertion.ASSUMESCAN.expt`.

`-max_number_of_testpoints` *`integer`*

Specifies the maximum number of test points to be inserted. Selection is based of the specified number of identified test points and the weight of the point resulting from logic cone analysis.The weight indicates the probability of a higher power level if toggling occurs.

`-min_slack` *`integer`*

Limits the insertion of a test point to those nodes that have the specified minimum slack (in ps).

Default: `2000`

`-output_tp_file` *`file`*

Specifies the name of the generated output file containing the recommended test points.

If you do not specify this option, the test point locations are written to the following file in the working directory: `TB/testresults/TestPointInsertion.ASSUMESCAN.expt`.

`-preview` Reports the test points to be added, without modifying the design.

`-report_virtual_scan_power` {*`integer...`*}

Performs a *virtual* insertion of the test points into the design and measures their impact on the reduction of scan power. Specify an integer value in MHZ for the scan clock frequency and/or flop-toggle frequency to use in the measurement. If a flop frequency is not specified, its value defaults to half the value of the scan clock frequency.

Note: This option is only valid with the `-preview` option.

`-test_control` *`test_signal`*

Specifies the test signal to enable the testpoint. This option is not required when the command is run with the `-preview` option.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode -scan_shift` constraint.

Related Information

Gating Functional Paths to Reduce Scan Shift Power in *Design for Test in Encounter RTL Compiler.*

Affected by these constraints:	define_dft shift_enable on page 572
	define_dft test_mode on page 581

insert_dft shadow_logic

```
insert_dft shadow_logic -around instances
     [-test_control test_signal]
     {-mode bypass
     |-mode no_share -test_clock_pin {port|pin}
          [-rise |-fall]
     |-mode share -test_clock_pin {port|pin}
          [-fall |-rise] }
     [-exclude pins | -only pins ] [-group pins]...
     [-balance] [-dont_map] [-preview]
```

Allows to insert two basic types of DFT shadow logic around a particular instance: bypass and scannable logic. Each shadow logic flip-flop can implement one control point and one observation point at the same time.

If you want to share observation and control points, either by setting `-mode` to `share` or `bypass`, the following sharing criteria are observed:

- If you specify `-group`, the specified inputs and outputs are grouped together as indicated
- For the remaining inputs and outputs that are not listed with `-group`, the first input will share the flip-flop with or be connected to the first output, the second input with the second output, and so on. The order is that specified in the HDL interface declaration.

Note: Test points are added for uni-directional pins only. Bidirectional pins and pins associated with test clock objects are skipped.

Options and Arguments

`-around` *`instances`*	Specifies the instances around which the DFT shadow logic must be inserted. Specify a hierarchical instance name.
`-balance`	Groups unmatched input and output pins to have a balanced number of groups.
`-dont_map`	Prevents the inserted logic from being mapped even if the design is already mapped to the target library
`-exclude` *`pins`*	Prevents the specified pins from being considered for shadow logic insertion.

`-group` *`pins`*	Specifies the pins to group when also using `-mode share` or `-mode bypass`. Each group can have multiple input pins and multiple output pins. Format the groups as follows: `{`*`input`*$_i$` ... `*`output`*$_j$`...}` Separate the pins by spaces. If you have more than one group, you must specify multiple `-group` options. **Note:** If `-mode` is set to `bypass`, each group must have at least one input and one output. Otherwise, the number of inputs or outputs can be equal or larger than zero.
`-mode`	Specifies the type of shadow logic to insert.

`bypass`	Implements bypass logic. If you specify this option, you must balance the number of inputs and outputs.
`no_share`	Inserts one scannable observation test point per input and one scannable control test point per output.
`share`	Pairs each input with an output and uses one scannable control and observation test point for each pair. If there are a different number of inputs and outputs, uses one scannable observe (or control) test point is used for each remaining input (or output).

`-only` *`pins`*	Restricts the pins to be considered for shadow logic insertion to the specified ones.
`-preview`	Shows the potential changes, without making any modifications to the netlist.
`[-rise|-fall]`	Specifies the edge of the test clock that is active during test mode operation. These options are only valid in conjunction with `-test_clock_pin`. *Default*: `-rise`

`-test_clock_pin {`*`port`*` | `*`pin`*`}`

Specifies the test clock that drives the clock pin of the shadow flip-flops. You can specify a port or pin that drives the test clock.

`-test_control` *`test_signal`*

Specifies the test signal to use to control DFT logic (the multiplexers after the controlling points).

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

Examples

In the following examples, the logic before the ATPG-untestable module is not observable and the logic after it is not controllable. Following is the Verilog input code for the ATPG-untestable module and its instantiation:

```
module blackbox (i1,i2, o1,o2,o3)
    input i1,i2;
    output o1,o2,o3;
...
blackbox U1 (.i1(n_1), .i2(n_2), .o1(n_3), .o2(n_4), .o3(n_5));
...
```

- Using the following examples, bypass logic is used to make the two inputs observable and the three outputs controllable. The first command pairs input `i1` to output `o1,` and input `i2` to output `o3` (skipping `o2`).The second command pairs input `i1` and output `o2`.

```
define_dft test_mode -name my_TM -active high TM
insert_dft shadow_logic -around U1 -test_control my_TM -mode bypass \
-exclude o2
insert_dft shadow_logic -around U1 -test_control my_TM -mode bypass \
-only {i1 o2}
```

- The following example uses scannable test points and shares these test points as control and observation points.

```
define_dft test_mode -name my_TM -active high TM
insert_dft shadow_logic -around U1 -test_control my_TM -test_clock_pin CK \
-mode share
```

- The following example uses scannable test points but does not share these test points for control and observation points.

```
define_dft test_mode -name my_TM -active high TM
insert_dft shadow_logic -around U1 -test_control my_TM -test_clock CK \
-mode no_share
```

- The following example uses scannable test points, shares these test points for control and observation points, and controls by grouping which pins share a common test point. More specifically, `i1` and `o2` share a test point, and `i2` and `o1`. In addition, no control point is inserted for the net driven by `U1/o3`.

```
define_dft test_mode -name my_TM -active high TM
insert_dft shadow_logic -around U1 -test_control my_TM -test_clock CK \
-exclude o3 -mode share -group {i1 o2} -group {i2 o1}
```

Related Information

Inserting DFT Shadow Logic in *Design for Test in Encounter RTL Compiler*

Affects these commands:	check_dft_rules on page 501
	report dft_registers on page 648
Related constraints:	define_dft shift_enable on page 572
	define_dft test_clock on page 577
	define_dft test_mode on page 581

insert_dft test_point

```
insert_dft test_point -location {pin|port}...
     [-test_control test_signal]
     -type { control_0 | control_1 } |
           {{async_0 | async_1 | async_any
            | control_node -node {pin|port}
            | control_observe_0 | control_observe_1
            | control_observe_node -node {pin|port}
            | control_scan
            | observe_scan [-max_observe_share integer]
            | scan | sync_0 | sync_1 | sync_any }
            -test_clock_pin {pin|port} [-rise|-fall] }
     [-dont_map]
```

Allows you to manually specify a control or observation test point to be added to the design. Control test points always require the specification of a test-mode signal. Test points that use scannable flip-flops to observe or control a node always require a test-clock signal.

For all of the scannable test points, you need to run `check_dft_rules` after the test point is inserted.

The command returns the path name of the inserted test point when it is a flip-flop.

Important

You can only specify multiple locations when you request to insert a test point of type `observe_scan`. In this case, the tool builds a balanced XOR-tree with all the specified location pins. Additionally, you can control the maximum number of pin locations to be observed by the same observation flop by specifying the `-max_observe_share` option. If a test-control signal is also specified, the tool builds the XOR-tree after each input is AND-ed or OR-ed with the test-control signal. This prevents switching along the XOR-tree when not it test mode. If the test control is active high, gating happens by AND-ing, otherwise by OR-ing. The output of the last XOR-gate is fed to the D input of the observation flip-flop.

Options and Arguments

`-dont_map` Prevents the inserted logic from being mapped even if the design is already mapped.

`-location {`*`port`*` | `*`pin`*`}`

Specifies the location of the control point or observation point. Specify an existing hierarchical pin name or a top-level port. For observation test points, the pin can be an input or output pin. For control test points, the result is different depending on the direction of the location. See the Examples on page 634.

Note: You can only specify multiple locations for a test point of type `observe_scan.`

Note: If you specify a bidirectional pin, no logic will be inserted unless you specify the direction of the pin.

`-max_observe_share `*`integer`*

Specify the maximum number of locations to be shared for an observe test point.

Note: This option applies only when you set `-type` to `observe_scan`.

`-node {`*`pin`*` | `*`port`*`}` Specifies the pin or port to insert when `-type` is set to `control_node` or `control_observe_node` and when the signal specified by `-test_control` is active.

`[-rise | -fall]` Specifies the edge of the specified test clock that is active during test mode operation. These options are only valid in conjunction with `-test_clock`.

Note: You must use the same clock edge when inserting a control flip-flop and an observation flip-flop.

Default: `-rise`

`-test_clock_pin {`*`port`*` | `*`pin`*`}`

Specifies the test clock that drives the clock pin of the inserted flip-flops during test mode operation. You can specify a port or pin that drives the test clock.

This option is required for all type point types set using the `-type` option except those of type `control_0` or `control_1`.

`-test_control` *`test_signal`*

Specifies the test signal to use to control or observe the specified location point.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

Note: Test points of type `observe_scan` do not require a test signal.

`-type`

Specifies the type of test point to insert at the specified location when the signal specified by `-test_control` is active. Possible values are:

`async_0`	Inserts an asynchronous control test point that forces the control point to the value 0.
`async_1`	Inserts an asynchronous control test point that forces the control point to the value 1.
`async_any`	Inserts an asynchronous control test point that forces the control point to take either the original value or the inverted value.
`control_0`	Inserts a constant value `0`.
`control_1`	Inserts a constant value `1`.
`control_node`	Inserts an arbitrary node.
`control_observe_0`	Inserts a control and an observation point. The control point is forced to the value 0.
`control_observe_1`	Inserts a control and an observation point. The control point is forced to the value 1.
`control_observe_node`	Inserts a control and an observation point. The control point is forced to the value of the node specified by `-node`.

`control_scan`	Inserts a flip-flop to force a particular value at the specified location during test mode operation. The flip-flop must be remapped to a scan flop before connecting it to a scan chain later on. **Note:** This option requires you to specify the `-test_clock_pin` option.
`observe_scan`	Inserts a flip-flop to observe the specified location. The flip-flop must be remapped to a scan flip-flop before connecting it to a scan chain later on. This option requires you to specify the `-test_clock_pin` option.
`scan`	Inserts a scannable control and observation test point. **Note:** This option requires you to specify the `-test_clock_pin` option.
`sync_0`	Inserts a synchronous control test point that forces the control point to the value 0.
`sync_1`	Inserts a synchronous control test point that forces the control point to the value 1.
`sync_any`	Inserts a synchronous control test point that forces the control point to take either the original value or the inverted value.

Examples

■ The following example inserts a scannable observation test point, using `CLK` to drive:

```
insert_dft test_point -location X/out -test_clock_pin CLK -type observe_scan
```

■ The following example inserts a control-1 and scannable observation point:

```
insert_dft test_point -location X/out -test_control TM \
-test_clock_pin CLK -type control_observe_1
```

- The following example inserts a scannable control point:

  ```
  insert_dft test_point -location X/out -test_control TM \
  -test_clock_pin CLK -type control_scan
  ```

- The following example inserts a scannable control and observation test point:

  ```
  insert_dft test_point -location X/out -test_control TM \
  -test_clock_pin CLK -type scan
  ```

- The following example inserts an async control-0 test point at hierarchical pin `X/out`:

  ```
  insert_dft test_point -location X/out -test_control TM -type async_0 \
  -test_clock_pin CK -fall.
  ```

- The following example inserts a synchronous control test point that forces the control point to the value 1 at hierarchical pin `X/out`:

  ```
  insert_dft test_point -location X/out -test_control TM -type sync_1 \
  -test_clock_pin CK -fall.
  ```

- The following example inserts two observation test points, one for `pin1`, `pin2` and `pin3`, and the other for `pin4`, `pin5` and `pin6`.

  ```
  insert_dft test_point -type observe_scan -max_observe_share 3 \
  -location pin1 pin2 pin3 pin4 pin5 pin6
  ```

Related Information

Inserting a Control and Observation Test Point in *Design for Test in Encounter RTL Compiler*.

Affects these commands:	check_dft_rules on page 501
	connect_scan_chains on page 523
	synthesize on page 294
Related constraints:	define_dft shift_enable on page 572
	define_dft test_clock on page 577
	define_dft test_mode on page 581
Related attributes:	Test Clock Attributes
	Test Signal Attributes

insert_dft user_test_point

```
insert_dft user_test_point -location {pin|port|subport}
     -cell {design|subdesign|libcell}
     {-cfi {pin|port}] | -no_cfi} [-cfo {pin|port}]
      -connect string [-connect string]...
     -name name
```

Inserts a user-defined test point at the specified location, and hooks it up to the specified pins.

Options and Arguments

`-cell {`*`design`*`|`*`subdesign`*`|`*`libcell`*`}`	Specifies the module or library cell to instantiate. The module can be loaded as a parallel design, or as a subdesign.
`-cfi {`*`pin`*`	`*`port`*`}`	Specifies the cell functional input (CFI) pin or port name.
`-cfo {`*`pin`*`	`*`port`*`}`	Specifies the cell functional output (CFO) pin or port name.
`-connect` *`string`*	Specifies a string consisting of a cell pin and the corresponding source-signal pin to which the cell pin must be connected. This string has the following format: `{`*`cell_pin source_pin`*`}` Use this option to specify most connections to the cell, except for the connections to the CFI and CFO pins.
`-location {`*`pin`*`|`*`port`*`|`*`subport`*`}`	Specifies a pin, port or subport that identifies where the test point must be inserted.
`-name` *`name`*	Specifies the instance name to be given to the user-defined test point.
`-no_cfi`	Specifies that the user test point cell has no CFI pin.

Examples

- The following example inserts design `MyUserTI` in design `top` at the `in1[2]` input port. Port `MyCFI` will be connected to input port `in1[2]`.

  ```
  insert_dft user_test_point -location top/in1[2] -cell /designs/MyUserTI \
  -cfi MyCFI -cfo MyCFO -connect {MyShiftEn se} -connect {MyWRCK wck}
  ```

Related Information

Inserting a User-Defined Control and Observation Test Point in *Design for Test in Encounter RTL Compiler*.

Affects these commands:	check_dft_rules on page 501
	connect_scan_chains on page 523
	synthesize on page 294

insert_dft wrapper_cell

```
insert_dft wrapper_cell -location pin_list
     [-floating_location_ok]
     [-skipped_locations_variable Tcl_variable]
     [-shared_through {buffer|combinational}]
     [-wck pin] -wsen pin
     {-decoded_select_cfi pin
     |-wint pin -wext pin [-wcap pin]}
     [-guard {0|1} -wig pin -wog pin] [-name segment_prefix]
```

Selects a built-in IEEE 1500 standard wrapper cell based on the given specifications, inserts it at the specified location, and hooks it up to the specified control signals.

The cell logic is automatically identified as a wrapper-cell segment. You can use the segment into another segment or chain.

The command returns the directory path to the `scan_segment` objects that it creates. If multiple locations are specified, the command returns multiple segments. You can find the objects created by the `insert_dft wrapper_cell` in:

`/designs/`*`top_design`*`/dft/scan_segments`

Important

Any segment inserted by this command cannot be removed.

Options and Arguments

`-decoded_select_cfi` *`pin`*

Specifies the source pin name of the decoded control logic for the select-cfi signal.

Use this option to connect an already decoded signal. Otherwise, control decoder logic may have to be inserted for each cell, and extra control wires will have to be hooked up to each wrapper cell.

`-floating_location_ok`

Specifies to insert a wrapper cell even if the specified pin (location) is floating.

`-guard {0|1}` Specifies the guard (safe_ value) out of a cell. The safe value prevents testing of one block from interfering with another block.

If this option is not specified, no guard value will be included.

`-location` *`pin_list`*

Specifies one or more pins that identify where the wrapper cell must be inserted.

Use the RC pin name to identify the input or output of a blackbox instance.

`-name` *`segment_prefix`*

Specifies the segment name prefix.

If you specified a single location pin, and there is no name conflict, the segment name will correspond to the specified prefix, otherwise a unique name will be generated for each segment that is derived from the specified prefix.

`-shared_through {buffer | combinational}`

Specifies whether the functional flop in the wrapper cell must be shared. If a shared cell is inserted, the command traces through the logic to identify a shareable functional flop (or flops).

A functional flop in a wrapper cell can be shared if

1. It is directly connected to the core pin through
   - ❑ a series of buffers (`buffer`)
   - ❑ complex combination logic (`combinational`)
2. It is mapped to a scan flip-flop for DFT purposes.
3. The clock pin to the flop is controllable; that is, the flip-flop must pass the DFT rules.
4. The flop has no connected set, reset, or enable pin.
5. The functional flop is not already shared with another wrapper cell.

Note: If you use this option and the cell cannot be shared with the functional flop, the RC-DFT engine gives a message and inserts a dedicated cell if you specified the `-wck` option, otherwise an error message is given.

`-skipped_locations_variable` *`Tcl_variable`*

Writes the locations where no wrapper cells can be inserted to the specified Tcl variable. If you omit this option, the command fails if you have any such locations specified.

`-wcap` *`driver`*	Specifies the capture control source pin or port name. **Note:** This option is ignored if you specified the `-decoded_select_cfi` option.
`-wck` *`driver`*	Specifies the test clock source pin or port name. This pin is required if you want to insert a dedicated wrapper cell.
`-wig` *`driver`*	Specifies the in-guard control source pin or port name.
`-wint` *`driver`*	Specifies the control source pin or port name for inward facing test mode. **Note:** This option is ignored if you specified the `-decoded_select_cfi` option.
`-wext` *`driver`*	Specifies the control source pin or port name for outward facing test mode. **Note:** This option is ignored if you specified the `-decoded_select_cfi` option.
`-wog` *`driver`*	Specifies the out-guard control source pin or port name.
`-wsen` *`driver`*	Specifies the shift-enable source pin or port name.

Examples

```
insert_dft wrapper_cell -location /top/core/in[0] -wsen /top/SEN \
-wint /top/WINT -wck /top/clk1
```

Related Information

Advanced DFT Topics in the *Design for Test in Encounter RTL Compiler* manual.

Affects these commands:	connect_scan_chains on page 523
Sets this attribute:	core_wrapper

read_dft_abstract_model

```
read_dft_abstract_model
     [-ctl] [-segment_prefix string]
     [-instance instance]
     [-assume_connected_shift_enable] file
```

Reads in the scan abstract model of a design that is used as a core or IP block in the current design. The scan abstract model defines the scan chain architecture of the subdesign and is used as scan chain segments in the configuration of the top-level scan chains of the current design.

The extracted scan chain information is stored in:

`/designs/`*`top_design`*`/dft/scan_segments`

Options and Arguments

`-assume_connected_shift_enable`

Indicates that the shift-enable port specified in the DFT abstract model for the block being read in is already connected to logic external to this block. Therefore the scan configuration engine does not need to modify the existing connection.

Note: If you specify this option and the shift-enable pin is *not* connected, the scan configuration engine will *not* make the connection.

If you do not specify this option, the scan configuration engine will make the connection to the shift-enable port specified in the DFT abstract model. If a connection already existed, it will be first removed.

`-ctl`

Specifies that the scan abstract model was written using the Core Test Language (CTL) format (IEEE format `P1450.6`).

If you omit this option, the scan abstract model is assumed to consist of a list of `define_dft abstract_segment` commands, one scan segment per scan chain in the subdesign.

`file`

Specifies the file that contains the abstract model description.

`-instance` *`instance`*	Applies the scan abstract model to the specified hierarchical instance. If you read in an abstract model written in native RC format, this instance must be an instantiation of the subdesign specified through the `-module` option in the abstract model. If you read in an abstract model written in CTL format, this instance must be an instantiation of the subdesign specified in the `Environment` section of the CTL file. If this option is omitted, the scan abstract model is applied to all instances of the subdesign.
`-segment_prefix` *`string`*	
	Adds the specified string as a prefix to the ■ Segment name defined in the native RC format file ■ Chain name defined in the CTL file

Related Information

Affected by these commands:	check_dft_rules on page 501
	connect_scan_chains on page 523
	synthesize on page 294
Related commands:	write_dft_abstract_model on page 666
	write_hdl on page 212 (`-abstract`)
Sets these attributes:	Scan Segment Attributes

read_io_speclist

`read_io_speclist` *`iospeclist_file`*

Reads in the specified IOSpecList input file to be used for boundary scan insertion.

The IOSpecList input file is only required to provide information that cannot be inferred from the design, and the command-line options of the `insert_dft boundary_scan` command.

You need an IOSpecList input file if

- The I/O pad cells in your library do not use the standard pin names
- Your design has pin sharing logic to shared functional output signals that was inserted before you insert boundary scan logic
- You want to customize the location of the boundary cells in the boundary register

You can also use an IOSpecList input file if

- You want to use custom boundary cells
- You want to use user-defined TAP instructions (such as those required for MBIST or PTAM) and use specific opcodes specified using `JTAG_Inline` syntax

  Note: You can also use the `define_dft jtag_instruction` command to enter user-defined instructions.

Options and Arguments

`iospeclist_file`	Specifies the IOSpecList input file.

Related Information

Inserting Boundary Scan Logic in *Design for Test in Encounter RTL Compiler*

Affects these commands:	insert_dft boundary_scan on page 607
	write_io_speclist on page 691

replace_scan

```
replace_scan [-to_non_scan] [-dont_check_dft_rules] [design]
```

This command either

- Replaces non-scan flops with their scan-equivalent flip-flops if the design was previously mapped.

  In this case, the `dft_scan_map_mode` design attribute must be set to either `tdrc_pass` or `force_all`. If set to `tdrc_pass`, you must have run the DFT rule checker.
- Replaces all scan flops that are part of shift register segments with non scan flops except for the first element of each shift register segment.

Options and Arguments

`design`	Specifies the design in which you want to replace regular flip-flops.
`-dont_check_dft_rules`	Prevents the DFT rules from being automatically checked.
`-to_non_scan`	Replaces all scan flops that are part of shift register segments to non scan flops except for the first element in the segments.

Related Information

Controlling Mapping to Scan in a Mapped Netlist in the *Design for Test in Encounter RTL Compiler* manual.

Affected by these commands:	check_dft_rules on page 501
	identify_test_mode_registers on page 595
	reset_scan_equivalent on page 652
	set_scan_equivalent on page 654
Affected by this attribute:	dft_scan_map_mode

report_scan_compressibility

`report_scan_compressibility -directory` *`atpg_directory`*

Reports the scan compressibility of a design. This command can only be used with a directory which has been created by the `analyze_scan_compressibility` command.

Options and Arguments

`-directory` *`atpg_directory`*

Specifies the directory where the ATPG compression runs created by the `analyze_scan_compressibility` command are stored.

Examples

- The following command reports the compressibility analysis resulting from the ATPG directory `temp_dir`.

```
rc:>  report_scan_compressibility -directory temp_dir

Design-DLX_CORE
Compressor-mimic_bidi_misr
Decompressor-xor
Mask-wide2
####################################
Analyze_dft_compressibility Results
####################################

Achieved compression table with fullscan topup vectors
##############################################################
IC    TATR   TDVR   Cov     CL      Cycles        Runtime
##############################################################
fs    1      1      99.86%  161     57318         1:20
10    7      17     99.88%  17      8159          1:50
20    10     19     99.85%  9       5581          2:00

Achieved compression table without fullscan topup vectors
##############################################################
IC    TATR   TDVR   Cov     CL      Cycles
##############################################################
fs    1      1      99.86%  161     57318
10    7      23     98.02%  17      7193
20    14     43     97.28%  9       3971
Total atpg runtime for exp. 5:10 hrs.

IC      - Inserted compression
TATR    - Test application time reduction
TDVR    - Test data volume reduction
Cov     - Atpg coverage
CL      - Channel Length
Cycles - Total no. of cycles for test
Runtime- Atpg runtime
fs      - fullscan run
```

Related Information

Analyzing and Reporting Scan Compressibility in *Design for Test in Encounter RTL Compiler.*

Compressing Scan Chains in *Design for Test in Encounter RTL Compiler*

Affected by this command	analyze_scan_compressibility on page 490

report dft_chains

Refer to `report dft_chains` in the Chapter 8, “Analysis and Report.”

report dft_registers

Refer to `report dft_registers` in the Chapter 8, “Analysis and Report.”

report dft_setup

Refer to `report dft_setup` in Chapter 8, "Analysis and Report."

report dft_violations

Refer to `report dft_violations` in Chapter 8, “Analysis and Report.”

report scan_power

Refer to `report scan_power` in the Chapter 8, “Analysis and Report.”

reset_scan_equivalent

```
reset_scan_equivalent [libcell]...
```

Removes the specified non-scan library cells from the scan-equivalency table which was previously defined using a (number of) `set_scan_equivalent` command(s).

If you do not specify any library cells, the command removes all scan-equivalent mappings.

Options and Arguments

`libcell`	Specifies a non-scan library cell to be removed from the scan-equivalency table.

Example

The following example removes the `snl_ffqx1` cell from the scan-equivalency table.

```
reset_scan_equivalent snl_ffqx1
```

Related Information

Affects this command:	replace_scan on page 644
Related command:	set_scan_equivalent on page 654

set_compatible_test_clocks

```
set_compatible_test_clocks
     {-all | list_of_test_clocks} [-design design]
```

Specifies the compatible test clocks whose related scan flip-flops can be merged into a single scan chain using lockup elements in between. By default, no test clocks (including different phases of the same clock) are assumed compatible.

Note: This command applies only to the muxed scan style.

Test clocks that are declared compatible belong to the same DFT clock domain.

Important

Test clocks with different clock periods cannot be made compatible.

Options and Arguments

`-all`	Specifies that all test clocks are compatible.
`design`	Specifies the design for which you want to specify compatible test clocks.
`list_of_test_clocks`	Specifies the compatible test clocks. You must specify the test clock object name. **Note:** To allow combining flip-flops from the same DFT domain—which are triggered by either edge of the same test clock—on the same scan chain, you need to set the `dft_mix_clock_edges_in_scan_chains` root attribute to `true`. By default, only same edge clocks are mixed.

Related Information

Mixing Different Test Clocks in the Same Scan Chain in *Design for Test in Encounter RTL Compiler*

Affects this command:	connect_scan_chains on page 523
Related attribute:	dft_mix_clock_edges_in_scan_chains

set_scan_equivalent

```
set_scan_equivalent
     -non_scan_cell libcell -scan_cell libcell
     [-tieoff_pins string] [-pin_map list_of_pin_groups]
```

Controls the scan-equivalent cell type that is used during the conversion of a non-scan flip-flop which passes the DFT rule checks to a scan flop. Use the `replace_scan` command to perform the actual conversion to scan.

Note: The RC-DFT engine automatically derives the scan data input, scan data output, and other test signals from the `test_cell` description of the scan flop in the target library.

Options and Arguments

`-non_scan_cell` *`libcell`*

Specifies a non-scan flip-flop library cell.

`-pin_map` *`list_of_pin_groups`*

Indicates how to map a pin from the non-scan flop to a pin in the scan flop when the pin names in the cells do not match.

The *`list_of_pin_groups`* has the following format:

`{{`*`non_scan_pin scan_pin`*`} {`*`non_scan_pin scan_pin`*`}...}`

`-scan_cell` *`libcell`*

Specifies a scan flip-flop library cell.

`-tieoff_pins` *`string`*

Specifies the tie-off value for extra scan cell pins.

The *`string`* has the following format:

`{{`*`pin value`*`} {`*`pin value`*`} ...}`

The value can be a logic `0` or `1`.

Example

The following example assumes that the pin names in the non-scan and scan flip-flops match, and that there are no extra pins in the scan flop to be tied off.

```
set_scan_equivalent -non_scan_cell snl_ffqx1 -scan_cell snl_sffqx1
```

Related Information

Affects this command:	replace_scan on page 644
Related command:	reset_scan_equivalent on page 652

write_atpg

```
write_atpg
     { -cadence [-compression | > file]
     | -mentor [> file]
     | -stil [-dft_configuration_mode dft_config_mode_name]
       [> file]}
     [-decimals_ok] [-picoseconds]
     [-test_clock_waveform test_clock]
     [-apply_inputs_at integer]
     [-apply_bidirs_at integer]
     [-dft_configuration_mode dft_config_mode_name]
     [-strobe_outputs_at integer]
     [-strobe_width integer] [design]
```

Writes out the scan-chain information for an Automatic Test Pattern Generator (ATPG) tool in a format readable by the designated ATPG tool.

The ATPG tool uses this information to generate appropriate test patterns. The file extension given to the interface file(s) is determined by the selected tool.

The interface file is useful only to the third-party tool if the test synthesis tool has connected the scan chain. Therefore, you should use this command only if the test synthesis tool has connected the scan chains.

Options and Arguments

`-apply_bidirs_at` *integer*

Specifies when in the test clock cycle to apply the bidirectional signals. Specify this time as a percentage of the test clock period.

Default: Same value as specified for `apply_inputs_at`

`-apply_inputs_at` *integer*

Specifies when in the test clock cycle to apply the input signals. Specify this time as a percentage of the test clock period.

Default: 0

`-cadence`	Creates pin-assignment files that capture the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) for use by the Encounter Test ATPG tool. If you use this option without the `-compression` option, the command writes out the pin-assignment information for full scan mode only. The information is written to the specified file.
`-compression`	Creates pin-assignment files for full scan mode, compression mode and XOR decompression mode. The following files are generated: *`topmodulename`*`.FULLSCAN.pinassign`, *`topmodulename`*`.COMPRESSION.pinassign,`and *`topmodulename`*`.COMPRESSION_DECOMP.pinassign.` **Note:** This option is only valid with the `-cadence` option.
`-decimals_ok`	Writes out decimal numbers. By default, time values are rounded off to integer numbers because many ATPG tools do not accept decimal numbers for test waveform time values. Use the `-picoseconds` option to minimize round-off errors.
`design`	Specifies the top module for which to write ATPG.
`-dft_configuration_mode` *`dft_configuration_mode_name`*	Writes ATPG information for the specified scan mode name. **Note:** This option is only valid with the `-stil` option.
`file`	Specifies the file to which the output must be written. If no file is specified, the output is written to standard out (`stdout)` and to the log file. **Note:** This argument is only valid with the `-stil` and `-cadence` options.
`-mentor`	Creates an interface file in the format used by the Mentor Graphics ATPG tool. Files generated: ■ `top_module.testproc` ■ `top_module.dofile`
`-picoseconds`	Specifies to use picoseconds for the time unit. Use this option to minimize the round-off errors when rounding-off test waveform time values to integers. *Default*: `nanoseconds`

`-stil`	Creates an interface file in the IEEE Standard Test Interface Language (STIL) format (IEEE format 1450.1). **Note:** The generated STIL format is TetraMAX compatible.
`-strobe_outputs_at` *`integer`*	Specifies when in the test clock cycle to strobe the outputs. Specify this time as a percentage of the test clock period. *Default*: 40
`-strobe_width` *`integer`*	Specifies how long the outputs are valid during the test clock cycle. Specify this time as a percentage of the test clock period. *Default*: 0
`-test_clock_waveform` *`test_clock`*	Specifies to use the clock waveform of the specified test clock. *Default*: first test clock object found

Related Information

Creating an Interface File for ATPG Tool in *Design for Test in Encounter RTL Compiler*

Affected by this command:	compress_scan_chains on page 508
	connect_scan_chains on page 523
	define_dft scan_clock_a on page 566
	define_dft scan_clock_b on page 569
	define_dft test_clock on page 577
Affected by these attributes:	Actual Scan Chain attributes

write_bsdl

```
write_bsdl
     [-pinmap_file file]
     [-bsdl_package_name files]
     [-bsdlout file]
     [-include_private_instructions]
     [-expose_ports_with_pinmap]
     -directory string
```

Generates a file describing the boundary scan architecture of the design in Boundary Scan Description Language (BSDL), along with two VHDL package files, `STD_1149_1_2001` and `CDNDFT_1149_1_2001`, which contain the supported boundary cell descriptions that were used during boundary scan insertion.

Note: Dedicated test-related signals (such as shift-enable, and test-mode signals defined without the `-shared_in` option) are also written to the BSDL file along with their respective compliance enable values.

Options and Arguments

`-bsdl_package_name` *`files`*	
	Specifies the name of a VHDL file or a comma-separated list of VHDL files, each containing one or more custom boundary cell descriptions that were used during boundary scan insertion.
	The name of each package file is added to the BSDL file in a `use` statement.
	Note: This option is required if you used custom boundary cells in the design.
`-bsdlout` *`string`*	Specifies the name of the BSDL file to be generated.
	If you omit this option, the output file is named using the *`topmodulename`*`.bsdl`.
`-directory` *`string`*	Specifies the directory to which the output files must be written.

`-expose_ports_with_pinmap`

Specifies to only expose functional and test ports with package pinmap information to the output BSDL file.

Additionally, this option prevents the writing of other ports connected in the boundary-scan register without package pinmap information. The names of these other ports will not appear in the BOUNDARY_REGISTER section of the BSDL file. Rather, these port names will be represented with an asterisk (`*`) and their associated boundary-scan cells will be represented as `INTERNAL` as shown in the following BSDL snippet:

```
attribute BOUNDARY_REGISTER of test: entity is
                    ...
"8  (BC_OUT, *, INTERNAL, X)," &
```

When the `-expose_ports_with_pinmap` is not specified, all ports in the design will be written to each relevant section of the BSDL file, regardless of whether any pinmap information has been provided for any ports.

Package pinmap information may be provided during boundary-scan insertion or when writing the BSDL file using the `-pinmap_file` option.

`-include_private_instructions`

Specifies to include register access information for private instructions in the BSDL file.

`-pinmap_file` *`file`* Specifies the name of the pinmap file to be used to create a BSDL file.

Note: This file can have fewer pin-to-pad bonding requirements than the pinmap file specified for the boundary scan insertion.

Refer to Pinmap File Format for more information.

Example

- The following command creates a BSDL file named `bsdlout.`The name of the package file for the custom boundary cells is added to the BSDL file.

  ```
  write_bsdl -directory .-bsdl_package_name MY_BIDIR_PKG -bsdlout bsdloutfile
  ```

  This causes the following line to be added to file `bsdlout`:

  ```
  use MY_BIDIR_PKG.all ;
  ```

Related Information

Inserting Boundary Scan Logic in *Design for Test in Encounter RTL Compiler*

Affected by this command:	insert_dft boundary_scan on page 607
Related constraints:	define_dft shift_enable on page 572
	define_dft test_clock on page 577
	define_dft test_mode on page 581

write_compression_macro

```
write_compression_macro
     -chains integer
     -sub_chains integer
     [-decompressor {broadcast | xor}]
     [-compressor {xor | opmisr | hybrid}]
     [-mask {wide0 | wide1 | wide2}]
     [-mask_sharing_ratio integer
     [-no_fullscan_muxing]
     [-jtag_control]
     [-serial_misr_read]
     [-file string]
     [-info_file string]
```

Generates the RTL for a customized scan compression macro.

Note: To use this command you need to have a license for the Encounter Test Architect tool.

Options and Arguments

`-chains` The number of top level scan data input/scan data output pairs. Typically, this is the number of uncompressed scan chains. For MISR-based compression, the `-chains` option must be greater than or equal to 16.

`-compressor {xor | opmisr| hybrid}`

Specifies the type of compression logic to be built:

- `xor` specifies to build an XOR-based compressor
- `opmisr` specifies to build a `MISR`-based compressor
- `hybrid` specifies to build `MISR` compression with `MISR` bypass capability to effectively result in an XOR-based compressor.

Default: `xor`

`-decompressor {broadcast | xor}`

Specifies the type of decompression logic to be built:

- `xor` specifies to build an XOR-based spreader network in addition to the broadcast-based decompression logic
- `broadcast` specifies to build a broadcast-based decompression logic (simple scan fanout).

`-file` Specifies the filename where the compression macro RTL will be written. If not specified, the RTL will be written to `stdout`.

`-info_file` Specifies a file containing more detailed information about the compression macro. This script can be sourced into the current session to provide more information about the compression macro to commands such as `write_et` so they can generate accurate input files for Encounter Test.

`-jtag_control` Specifies to include a JTAG-controlled test data register (TDR) which generates compression test signals to configure the compression testmode.

`-mask {wide0 |wide1 | wide2}`

Inserts scan channel masking logic of the specified type.

The masking types that can be used depend on the compressor type specified with the -compressor option.

By default, no masking logic is inserted.

`-mask_sharing_ratio` *`integer`*

Specifies the number of internal scan channels sharing a mask register. The specified integer may not exceed the ratio of the number of subchains to the number of chains.

Note: This option is only valid with `wide1` and `wide2` masking.

`-no_fullscan_muxing` Specifies to exclude additional muxing logic to the compression macro. By default, additional muxing is added to the compression macro to concatenate the compressed scan channels into uncompressed fullscan chains. If such muxing exists outside the compression macro, specify this option to exclude this logic from the compression macro.

`-serial_misr_read` Specifies to include support for reading MISR bits serially through the scan data pins.

`-sub_chains` *`integer`*

Specifies the number of compressed scan channels that exist in the design or that you will build. The specified integer must be an even multiple of the integer specified for the `-chains` option unless the `-no_fullscan_muxing` option is also specified.

Examples

- The following command writes an XOR-based compression macro without masking to the file `xor1.v`.

  ```
  rc:/> write_compression_macro -compressor xor. -chains 8 -sub_chains 88 -file \
   xor1.v
  Checking out license 'Encounter_Test_Architect'... (1 seconds elapsed)
  ...
  ```

- The following command writes an XOR-based compression macro with masking logic of type `wide1`. The `-no_fullscan_muxing` option is specified so the logic to concatenate the compressed `sub_chains` into fullscan chains will be excluded.

  Note: Note, since the `-no_fullscan_muxing` option is specified, the number of sub_chains is no longer required to be evenly divisible by the number of chains.

  ```
  rc:/> write_compression_macro -compressor xor -mask wide1 -chains 8
  -sub_chains 85 -no_fullscan_muxing -file xor2.vChecking out license
  'Encounter_Test_Architect'... (1 seconds elapsed)....
  ```

-

- The following command writes an MISR-based compression macro with masking logic of type `wide1`, with decompression logic of type `xor`. The compressor type is `hybrid` which means the MISR can be bypassed resulting in XOR compression.

  ```
  rc:/> write_compression_macro -compressor hybrid -decompressor xor \
  -mask wide1 -chains 16 -sub_chains 512 -file hybrid1.v
  Checking out license 'Encounter_Test_Architect'... (2 seconds elapsed)
  ...
  ```

- The following command writes a MISR-based compression macro with masking logic of type `wide2`. The `-info_file` option is also specified.

  ```
  rc:/> write_compression_macro -compressor opmisr -mask wide2 -chains 20 \
   -sub_chains 500 -file opmisr1.v -info_file opmisr1.info
  ....
  ```

Related Information

Compressing Scan Chains in *Design for Test in Encounter RTL Compiler*

Manually Inserting a Scan Compression Macro in *Design for Test in Encounter RTL Compiler*

write_dft_abstract_model

```
write_dft_abstract_model [-ctl]  [design] [> file]
     [-dft_configuration_mode dft_config_mode_name]
```

Writes a scan abstract model for all the top-level scan chains configured in the design.

Note: Currently, this command is not supported for the clocked LSSD scan style with the `-ctl` option.

Options and Arguments

`-ctl`	Writes out a scan abstract model in the Core Test Language (CTL) format (IEEE format `P1450.6`). If you omit this option, the scan abstract model is written in native RC format that consists of a list of `define_dft abstract_segment` commands, one per top-level scan chain.
`design`	Specifies the design for which to write out the scan abstract model of the scan chains. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-dft_configuration_mode` *`dft_configuration_mode_name`*	Writes scan chain information related to the specified scan mode name.
`file`	Specifies the file to which the output must be written. You can write out the CTL file in compressed format by specifying a file name with the `.gz` extension. *Default*: output is written to the screen

Examples

■ In the following example, the different active edges of the different test clocks in the same test clock domain are allowed to be mixed on the same scan chains. Following shows the configuration result and the scan abstract models for the scan chains:

```
rc:> connect_scan_chains
    Configuring 1 chains for 27 scan f/f
  Configured 1 chains for Domain: 'clkAll', edge: 'mixed'
         AutoChain_1 (DFT_sdi_1 -> DFT_sdo_1) has 27 registers; Domain:clkAll,
edge: mixed
```

```
  Processing 1 scan chains in ’muxed_scan’ style.
  Using default shift enable signal ’SE’: ’/designs/test/ports_in/SE’ active
high.
    Connecting scan chain ’AutoChain_1’ with 27 flip-flops.
Mapping DFT logic introduced by scan chain connection...
Mapping DFT logic done.
Reporting 1 scan chain

Chain 1: AutoChain_1
  scan_in:      DFT_sdi_1
  scan_out:     DFT_sdo_1
  shift_enable: SE (active high)
  clock_domain: clkAll (edge: mixed)
  length: 27
    bit 1       out1_reg_4 <test_clk1/fall>
    ...
    bit 5       out1_reg_8 <test_clk1/fall>
    llatch 5    DFT_lockup_g1
    bit 6       out2_reg_4 <test_clk2/fall>
    ...
    bit 10      out2_reg_8 <test_clk2/fall>
    llatch 10   DFT_lockup_g348
    bit 11      out3_reg_4 <test_clk3/fall>
    ...
    bit 18      out1_reg_2 <test_clk1/rise>
    bit 19      out1_reg_3 <test_clk1/rise>
    llatch 19   DFT_lockup_g349
    bit 20      out2_reg_0 <test_clk2/rise>
    ...
    bit 23      out2_reg_3 <test_clk2/rise>
    llatch 23   DFT_lockup_g350
    bit 24      out3_reg_0 <test_clk3/rise>
    ...
    bit 27      out3_reg_3 <test_clk3/rise>
------------------------

rc:/> write_dft_abstract_model
scan style is muxed_scan

# writing abstract model for 1 scan chain

  define_dft abstract_segment -module test \
    -name test_AutoChain_1 \
    -sdi DFT_sdi_1 -sdo DFT_sdo_1 \
    -shift_enable_port SE -active high \
    -clock_port clk1 -fall \
    -tail_clock_port clk3 -tail_edge_rise \
    -length 27
```

To avoid naming collisions when reading in a scan abstract model, the segment names are prefixed with the module name.

Related Information

Creating a Scan Abstract Model in *Design for Test in Encounter RTL Compiler*

Affected by this command:	connect_scan_chains on page 523

Affected by these attributes:	Actual Scan Chain attributes
Related command:	read_dft_abstract_model on page 641
	write_hdl on page 212 (`-abstract`)

write_et_atpg

```
write_et_atpg
     [-compression] [-effort string] [-force]
     [[-configuration_mode_order dft_configuration_mode+]...|
      [-dft_configuration_mode dft_config_mode_name+]...]
     [-build_model_options string] [-build_testmode_options string]
     [-atpg_options string] [-verify_test_structures_options string]
     [-library string] [-directory string] [design]
```

Writes out the necessary files and the template run scripts to run Automatic Test Pattern Generator (ATPG) using the Encounter Test software.

This command generates the following files:

- `et.exclude`—A file listing objects to be excluded from the ATPG analysis in an assumed scan mode.
- `et.modedef`––A mode definition file, a text file that describes the test mode in `assumed scan mode`
- *`topmodulename`*`.ASSUMED.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock) and their test function used to build the testmode before actual scan chains exist in the design
- *`topmodulename`*`.FULLSCAN.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- If the `write_et_atpg` command is run with the `-compression` option, the following pin-assignment files are generated in addition to the *`topmodulename`*`.FULLSCAN.pinassign` file. In this case, all three files include the compression test signals with their appropriate test functions to validate their specific test mode:
  - *`topmodulename`*`.COMPRESSION_DECOMP.pinassign`—A file generated *only* when inserting XOR-based decompression logic
  - *`topmodulename.`*`COMPRESSION`*`.pinassign`*—A file generated when inserting broadcast-based decompression logic
- `runet.atpg ––` A template script file to run the requested testability analysis
- *`topmodulename`*`.et_netlist.v`––A netlist for Encounter Test
- `run_compression_decomp_sim`––An NCVerilog run file used to simulate the test patterns created by Encounter Test for compression logic built using XOR-based decompression logic

- `run_compression_sim`––An NCVerilog run file used to simulate the test patterns created by Encounter Test for compression logic built using broadcast-based decompression logic
- `run_fullscan_sim`––An NCVerilog run file used to simulate the test patterns created by Encounter Test in full scan mode.

Note: Some files can be customized according to the setup requirements.

For more information on the exact Encounter Test product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Options and Arguments

`-atpg_options {`*`option1=value option2=value`*`}`

Specifies extra ATPG analysis options.

`-build_model_options {`*`option1=value option2=value`*`}`

Specifies extra options to apply when building the Encounter Test model.

`-build_testmode_options {`*`option1=value option2=value`*`}`

Specifies extra options to apply when building the test mode for Encounter Test.

`-compression` Instructs to write out the files needed to run ATPG-based testability analysis for compression mode.

`-configuration_mode_order` *`dft_configuration_mode...`*

Specifies to write Encounter Test script files for a compression mode.Valid compression mode names are:

`COMRESSION`, `COMRESSION_DECOMP`, `OPMISRPLUS`, `OPMISRPLUS_DECOMP`, `FULLSCAN`

Note: If specified, the `FULLSCAN` compression mode must be specified last.

`design`

Specifies the design for which to write out the Encounter Test input files.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dft_configuration_mode` *`dft_configuration_mode_name...`*

Writes scan chain information related to the specified scan mode name(s).

`-directory` *`string`*

Specifies the directory to which the output files must be written.

Default: *`current_working_directory/`*`et_scripts`

`-effort {low | medium | high}`

Specifies the ATPG effort level to expend in resolving faults. Increasing effort will generally result in resolving more faults, but will require more processing time, sometimes significantly more.

Default: `low`

`-force`

Specifies to continue even when the DFT logic appears to be incomplete.

`-library` *`string`*

Specifies the list of Verilog structural library files.

The Verilog libraries required to run Encounter Test ATPG and NC-Verilog simulation of the generated vectors must be provided using the `-library` option of the `write_et_atpg` command. These libraries must be in a structural format. The files can be specified separately on the `write_et_atpg` command line or can be referenced using an *include* file. Directories of Verilog files can also be specified but they cannot be referenced in the include file.

For example, if the Verilog files required to run ATPG and NC-Verilog simulation are:

```
./padcells.v
./stdcells.v
./memories/*.v
./ip_blocks/*.v
```

`write_et_atpg` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_atpg -library "./padcells.v ./stdcells.v \
 ./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
‘include "./padcells.v"
‘include "./stdcells.v"
```

And then specify the following:

```
write_et_atpg -library "include_libraries.v
./memories \
 ./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

`-verify_test_structures_options {`*`option1`*`=`*`value option2`*`=`*`value`*`}`

Specifies extra options to apply when performing test structure verification for Encounter Test.

Examples

- The following command generates the files to run an ATPG-based testability analysis.

  ```
  write_et_atpg -directory atpg -library $sim/mylib.v
  Examining the atpg directory that was generated shows the following files:
  rc:/>  shell ls atpg
  run_compression_decomp_sim
  run_compression_sim
  runet.atpg
  run_fullscan_sim
  test.COMPRESSION_DECOMP.pinassign
  test.COMPRESSION.pinassign
  test.et_netlist.v
  test.FULLSCAN.pinassign
  test.rc_netlist.v
  ```

- The following command uses the `-configuration_mode_order` option to generate the files to run an ATPG-based testability analysis first using the `OPMISRPLUS_DECOMP` compression mode and then with the `FULLSCAN` mode.

  ```
  write_et -atpg -configuration_mode_order {OPMISRPLUS_DECOMP FULLSCAN} \
   -directory rc_et
  ```

write_et_bsv

```
write_et_bsv
     [-bsdl file] [-bsdl_package_path string] [-bsdl_package_name files]
     [-build_model_options string] [-library string]
     [-directory string] [design]
```

Writes out the necessary files and the template run scripts to run boundary scan verification.

This command generates the following files:

- `topmodulename.bsdl`––A BSDL file
- `topmodulename.et_netlist.v`––A netlist for Encounter Test
- `runet.bsv`—An Encounter Test run file to run Boundary Scan Verification (BSV).

Note: Some files can be customized according to the setup requirements.

For more information on the exact Encounter Test product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Options and Arguments

`-bsdl` *file*	Specifies the name of the BSDL file to be used for the boundary scan verification. If you omit this option but you specified the `-bsv` option, this command will automatically run the `write_bsdl` command to generate the BSDL file. *Important* You must use this option if you inserted custom boundary cells in the design. Additionally, the BSDL file should be written using the `write_bsdl` command with the `-bsdl_package_name` option to list the custom package file to be used during boundary scan verification.

`-bsdl_package_name` *`files`*

Specifies a package file or a comma-separated list of package files that describe the custom boundary cells used in the design.

Note: This option is only required if you used custom boundary cells in the design. This option cannot be specified without the `-bsdl` option.

`bsdl_package_path` *`string`*

Specifies the UNIX directory or a comma-separated list of directories that indicate(s) where to find the package file(s).

You can use dot (.) to refer to the current working directory.

Note: This option is only required if you used custom boundary cells in the design. This option cannot be specified without the `-bsdl` option.

`-build_model_options {`*`option1`*`=`*`value option2`*`=`*`value`*`}`

Specifies extra options to apply when building the Encounter Test model.

`design` Specifies the design for which to write out the Encounter Test input files.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-directory` *`string`* Specifies the directory to which the output files must be written.

Default: *`current_working_directory/`*`et_scripts`

`-library` *`string`*

Specifies the list of Verilog structural library files.

The Verilog libraries required to run Encounter Test ATPG and NC-Verilog simulation of the generated vectors must be provided using the `-library` option of the `write_et_bsv` command. These libraries must be in a structural format. The files can be specified separately on the `write_et_bsv` command line or can be referenced using an *include* file. Directories of Verilog files can also be specified but they cannot be referenced in the include file.

For example, if the Verilog files required to run ATPG and NC-Verilog simulation are:

```
./padcells.v
./stdcells.v
./memories/*.v
./ip_blocks/*.v
```

`write_et_bsv` can be used in either of the following ways:

1. If specifying files separately on the command line:

   ```
   write_et_bsv -library "./padcells.v ./stdcells.v \
    ./memories ./ip_blocks" ...
   ```

2. If using an include file. Create an include file named `include_libraries.v` containing:

   ```
   ‘include "./padcells.v"
   ‘include "./stdcells.v"
   ```

   And then specify the following:

   ```
   write_et_bsv \
    -library "include_libraries.v ./memories \
    ./ip_blocks" ...
   ```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

Examples

- The following command generates the files to run an ATPG-based testability analysis.

  ```
  write_et_bsv -directory bsv -library $sim/mylib.v
  ```

  Examining the `atpg` directory that was generated shows the following files:

  ```
  rc:/>  shell ls bsv
  topmodulename.bsdl
  runet.bsv
  test.et_netlist.v
  test.rc_netlist.v
  ```

Related Information

Generating Script for Boundary Scan Verification in *Design for Test in Encounter RTL Compiler*

write_et_dfa

```
write_et_dfa
     [-library string] [-directory string] [design] [-effort string]
     [-build_model_options string] [-build_testmode_options string]
     [-atpg_options string] [-dfa_options string]
     [-verify_test_structures_options string]
```

Writes out the necessary files and the template run scripts to run Deterministic Fault Analysis using the Encounter Test software. The template script will only be written if actual scan chains exist in the design. DFA analysis is not supported in assumed scan mode.

This command generates the following files:

- `*topmodulename*.FULLSCAN.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- `runet.dfa` –– A template script file to run the requested deterministic fault analysis
- `*topmodulename*.et_netlist.v`––A netlist for Encounter Test
- `TestPointInsertion.*testmode_name*.dfa`–– A file containing test point locations.
- `run_fullscan_sim`––An NCVerilog run file used to simulate the test patterns created by Encounter Test in full scan mode.

Note: Some files can be customized according to the setup requirements.

For more information on the exact Encounter Test product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Options and Arguments

`-atpg_options {`*`option1=value option2=value`*`}`

Specifies extra ATPG analysis options.

`-build_model_options {`*`option1=value option2=value`*`}`

Specifies extra options to apply when building the Encounter Test model.

`-build_testmode_options {`*`option1=value option2=value`*`}`

Specifies extra options to apply when building the test mode for Encounter Test.

`design`

Specifies the design for which to write out the Encounter Test input files.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dfa_options {`*`option1=value option2=value`*`}`

Specifies extra options for deterministic fault analysis.

`-directory` *`string`*

Specifies the directory to which the output files must be written.

Default: *`current_working_directory/`*`et_scripts`

`-effort {low | medium | high}`

Specifies the ATPG effort level to expend in resolving faults. Increasing effort will generally result in resolving more faults, but will require more processing time, sometimes significantly more.

Default: `low`

`-library` *`string`*

Specifies the list of Verilog structural library files.

The Verilog libraries required to run Encounter Test ATPG and NC-Verilog simulation of the generated vectors must be provided using the `-library` option of the `write_et_dfa` command. These libraries must be in a structural format. The files can be specified separately on the `write_et_dfa` command line or can be referenced using an *include* file. Directories of Verilog files can also be specified but they cannot be referenced in the include file.

For example, if the Verilog files required to run ATPG and NC-Verilog simulation are:

```
./padcells.v
./stdcells.v
./memories/*.v
./ip_blocks/*.v
```

`write_et_dfa` can be used in either of the following ways:

1. If specifying files separately on the command line:

   ```
   write_et_dfa -library "./padcells.v ./stdcells.v \
    ./memories ./ip_blocks" ...
   ```

2. If using an include file. Create an include file named `include_libraries.v` containing:

   ```
   ‘include "./padcells.v"
   ‘include "./stdcells.v"
   ```

   And then specify the following:

   ```
   write_et_dfa -library "include_libraries.v
   ./memories \
    ./ip_blocks" ...
   ```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

`-verify_test_structures_options {`*`option1`*`=`*`value option2`*`=`*`value`*`}`

Specifies extra options to apply when performing test structure verification for Encounter Test.

Examples

- The following command generates the files to run deterministic fault analysis.

  ```
  write_et_dfa -directory dfa -library $sim/mylib.v
  Examining the dfa directory that was generated shows the following files:
  rc:/>  shell ls dfa
  runet.dfa
  run_fullscan_sim
  test.et_netlist.v
  test.FULLSCAN.pinassign
  test.rc_netlist.v
  ```

write_et_mbist

```
write_et_mbist
     -mbist_interface_file_dir string -mbist_interface_file_list string
     [-build_model_options string] [-create_embedded_test_options string]
     [-bsv [-bsdl file [-bsdl_package_path string]
              [-bsdl_package_name files]]
     -library string [-directory string]  [design]
```

Writes out the necessary files and the template run scripts to run Create Embedded Test using the Encounter Test software.

This command generates the following files:

- `*topmodulename*.ASSUMED.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock) and their test function used to build the testmode before actual scan chains exist in the design
- `*topmodulename*.FULLSCAN.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- *topmodulename*.`bsdl`––A BSDL file produced when specifying the `-bsv` and BSDL-related options.
- `runet.mbist`—An Encounter Test run file to run Boundary Scan Verification (BSV).when specifying the `-bsv` option.
- *topmodulename*`.et_netlist.v`––A netlist for Encounter Test
- `runet.mbist_interface`––A template script file to run Create Embedded Test in Encounter Test

Note: Some files can be customized according to the setup requirements.

For more information on the exact Encounter Test product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Options and Arguments

`-bsdl` *`file`*

Specifies the name of the BSDL file to be used for the boundary scan verification.

If you omit this option but you specified the `-bsv` option, this command will automatically run the `write_bsdl` command to generate the BSDL file.

Important

You must use this option if you inserted custom boundary cells in the design. Additionally, the BSDL file should be written using the `write_bsdl` command with the `-bsdl_package_name` option to list the custom package file to be used during boundary scan verification.

`-bsdl_package_name` *`files`*

Specifies a package file or a comma-separated list of package files that describe the custom boundary cells used in the design.

Note: This option is only required if you used custom boundary cells in the design. This option cannot be specified without the `-bsdl` option.

`-bsdl_package_path` *`string`*

Specifies the UNIX directory or a comma-separated list of directories that indicate(s) where to find the package file(s).

You can use dot (.) to refer to the current working directory.

Note: This option is only required if you used custom boundary cells in the design. This option cannot be specified without the `-bsdl` option.

`-bsv`

Writes out the files needed for boundary scan verification.

Note: This option prints a pin assignment file if differential PAD pairs are detected in the design.

`-build_model_options {`*`option1=value option2=value`*`}`

Specifies extra options to apply when building the Encounter Test model.

`-create_embedded_test_options {`*`option1`*`=`*`value option2`*`=`*`value`*`}`	
	Specifies extra options to apply when running Create Embedded Test in Encounter Test.
`design`	Specifies the design for which to write out the Encounter Test input files.
	If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-directory` *`string`*	Specifies the directory to which the output files must be written.
	Default: *`current_working_directory/`*`et_scripts`

`-library` *`string`*

Specifies the list of Verilog structural library files.

The Verilog libraries required to run Encounter Test ATPG and NC-Verilog simulation of the generated vectors must be provided using the `-library` option of the `write_et_mbist` command. These libraries must be in a structural format. The files can be specified separately on the `write_et_mbist` command line or can be referenced using an *include* file. Directories of Verilog files can also be specified but they cannot be referenced in the include file.

For example, if the Verilog files required to run ATPG and NC-Verilog simulation are:

```
./padcells.v
./stdcells.v
./memories/*.v
./ip_blocks/*.v
```

`write_et` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_mbist -library "./padcells.v ./stdcells.v \
 ./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
‘include "./padcells.v"
‘include "./stdcells.v"
```

And then specify the following:

```
write_et_mmbist -library "include_libraries.v \
./memories  ./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

`-mbist_interface_file_dir`

Specifies the MBIST interface file directories. Separate the directory names with blank spaces.

`-mbist_interface_file_list`

Specifies a list of MBIST interface files. Separate the file names with commas, for example, `file1,file2`.

Examples

- The following command generates the files to run Boundary Scan Verfication and Create Embedded Test in Encounter Test..

  ```
  write_et_mbist -mbist_interface_file_dir directory \
  -mbist_interface_file_list file1,file2 -bsv -library $sim/mylib.v
  ```

  Examining the `atpg` directory that was generated shows the following files:

  ```
  rc:/>  shell ls mbist
  test.COMPRESSION_DECOMP.pinassign
  test.COMPRESSION.pinassign
  test.et_netlist.v
  test.FULLSCAN.pinassign
  test.rc_netlist.v
  runet.mbist
  runet.mbist_interface
  ```

write_et_rrfa

```
write_et_rrfa
     [-atpg] [-force] [-effort string]
     [-build_model_options string] [-build_testmode_options string]
     [-atpg_options string] [-verify_test_structures_options string]
     [-library string] [-directory string] [design]
```

Writes out the necessary files and the template run scripts to run either an Automatic Test Pattern Generator (ATPG) or Random Resistance Fault Analysis (RRFA) based testability analysis, generate test patterns using the Encounter Test software.

This command generates the following files:

- `et.exclude`—A file listing objects to be excluded from the ATPG analysis in an assumed scan mode.
- `et.modedef`––A mode definition file, a text file that describes the test mode in `assumed scan mode`
- *`topmodulename`*`.ASSUMED.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock) and their test function used to build the testmode before actual scan chains exist in the design
- *`topmodulename`*`.FULLSCAN.pinassign`––A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- `run_fullscan_sim`––An NCVerilog run file used to simulate the test patterns created by Encounter Test in full scan mode.

Note: Some files can be customized according to the setup requirements.

For more information on the exact Encounter Test product requirements, refer to Encounter Test Product Requirements for Advanced Features in *Design for Test in Encounter RTL Compiler.*

Options and Arguments

`-atpg`	Writes out the files needed to run Automatic Test Pattern Generation using the Encounter Test software.
`-atpg_options {`*`option1=value option2=value`*`}`	Specifies extra ATPG analysis options.
`-build_model_options {`*`option1=value option2=value`*`}`	Specifies extra options to apply when building the Encounter Test model.
`-build_testmode_options {`*`option1=value option2=value`*`}`	Specifies extra options to apply when building the test mode for Encounter Test.
`design`	Specifies the design for which to write out the Encounter Test input files. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-directory` *`string`*	Specifies the directory to which the output files must be written. *Default*: *`current_working_directory/`*`et_scripts`
`-effort {low | medium | high}`	Specifies the ATPG effort level to expend in resolving faults. Increasing effort will generally result in resolving more faults, but will require more processing time, sometimes significantly more. *Default*: `low`
`-force`	Specifies to continue even when the DFT logic appears to be incomplete.

`-library` *`string`*

Specifies the list of Verilog structural library files.

The Verilog libraries required to run Encounter Test ATPG and NC-Verilog simulation of the generated vectors must be provided using the `-library` option of the `write_et_rrfa` command. These libraries must be in a structural format. The files can be specified separately on the `write_et_rrfa` command line or can be referenced using an *include* file. Directories of Verilog files can also be specified but they cannot be referenced in the include file.

For example, if the Verilog files required to run ATPG and NC-Verilog simulation are:

```
./padcells.v
./stdcells.v
./memories/*.v
./ip_blocks/*.v
```

`write_et_rrfa` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_rrfa -library "./padcells.v ./stdcells.v \
 ./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
‘include "./padcells.v"
‘include "./stdcells.v"
```

And then specify the following:

```
write_et_rrfa \
-library "include_libraries.v ./memories \
 ./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

`-verify_test_structures_options {`*`option1`*`=`*`value option2`*`=`*`value`*`}`

Specifies extra options to apply when performing test structure verification for Encounter Test.

Examples

- The following command generates the files to run an ATPG-based testability analysis.

  ```
  write_et_rrfa -atpg -directory atpg -library $sim/mylib.v
  ```

  Examining the `atpg` directory that was generated shows the following files:

  ```
  rc:/>  shell ls rrfa
  run_compression_decomp_sim
  run_compression_sim
  runet.atpg
  run_fullscan_sim
  test.COMPRESSION_DECOMP.pinassign
  test.COMPRESSION.pinassign
  test.et_netlist.v
  test.FULLSCAN.pinassign
  test.rc_netlist.v
  ```

write_io_speclist

```
write_io_speclist > iospeclist_file
     [-supplemental_file file]
```

Writes out the IOSpecList output file.

The IOSpeclist output file describes the boundary scan architecture of the design. The file contains all ports (functional, test, and TAP) in the design, specifies the type and location of the boundary cells to be inserted on all the functional ports, and lists the instructions (both mandatory and user-defined) to be built in the JTAG Macro.

Options and Arguments

iospeclist_file Specifies the name of the file to be written.

`-supplemental_file` *file*

Specifies the name of the supplemental file to write out. This file and its corresponding IOspeclist file are used to define the boundary scan objects prior to inserting boundary scan logic. The supplemental file lists the boundary scan segments and the JTAG-instruction definitions for its related objects written to the IOspeclist file. When using an IOspeclist file to define the order of the boundary scan register, the supplemental file should be included into the RTL Compiler session before reading the IOspeclist input file. Both the supplemental and the IOspeclist files need only be written if your intention is to insert boundary-scan logic in a new RTL Compiler session.

Note: The recommended approach to completely restoring the DFT setup (including the boundary scan objects) in a new RTL Compiler session is to use the `write_script`/`read_netlist` approach.

Related Information

Inserting Boundary Scan Logic in *Design for Test in Encounter RTL Compiler*

Affected by these commands:	define_dft jtag_instruction on page 543
	define_dft jtag_instruction_register on page 547
	insert_dft boundary_scan on page 607
	insert_dft mbist on page 618
	insert_dft ptam on page 622
Related command:	read_io_speclist on page 643

write_scandef

```
write_scandef
     [-partition partition -chains chain [chain]...]...
     [-version {5.4|5.5}]
     [-end_chains_before_lockups]
     [-dft_configuration_mode dft_config_mode_name]
     [-dont_split_by_library_domains]
     [-dont_split_by_power_domains]
     [-dont_use_timing_model_pins] [design] [ > file]
```

Writes the scanDEF description of the top-level scan chains configured in the design for reordering using a physical design tool.

Options and Arguments

`-chains` *`chain`*	Specifies the scan chains that must be grouped in the same partition by the physical design tool. Use the `report dft_chains` command to obtain a list of valid chain names. The tool ensures that chains or chain segments that are not compatible are not added to the same partition, but are further partitioned by adding the test clock name and test clock edge to the final partition name. **Note:** Requires version 5.5.
`design`	Specifies the design for which to write out the scanDEF description. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-dft_configuration_mode` *`dft_configuration_mode_name`*	Specifies the configuration mode for which to write the scan definition
`-dont_split_by_library_domains`	Indicates not to split the chains at the scan data input pin of the last flop in the originating (or from) library domain and at the scan data output pin of the first flop in the destination (or to) library domain. By default, the chains will be split based on the library domains. If the design has no library domains, the chains will not be split.

`-dont_split_by_power_domains`

Indicates not to split the chains at the scan data input pin of the last flop in the originating (or from) power domain and at the scan data output pin of the first flop in the destination (or to) power domain.

By default, the chains will be split based on the power domains. If the design has no power domains, the chains will not be split.

`-dont_use_timing_model_pins`

Prevents using the user-designated libcell timing model pins as the scanDEF chain `START` and `STOP` points. Instead an outward trace is performed to identify and use the first flip-flop scan data output pin and last flip-flop scan data input pin and use these pins as `START` and `STOP` pins in the scanDEF chains.

`-end_chains_before_lockups`

Terminates the scan segment at the scan data input pin of the scan flop which precedes the lockup element in the scan DEF chain.

`file` Specifies the file to which the output must be written. To write out the scanDEF file in compressed format, specify a file name with the `.gz` extension.

Default: output is written to the screen

`-partition` *`partition`*

Specifies the name of a user-defined partition.

Note: Requires version 5.5.

`-version {5.4|5.5}` Specifies which DEF version to write out. Version 5.5 writes out the `MAXBITS` and `PARTITION` keywords as regular statements (that is, uncommented).

Default: `5.4`

Example

- The following example writes out the scanDEF information to the screen:

```
rc:/> write_scandef

VERSION 5.4 ;
NAMESCASESENSITIVE ON ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN top ;

SCANCHAINS 2 ;
 - chain_1
 + START u_a/out_reg_1 Q
 + FLOATING
  u_a/out_reg_2 ( IN SI ) ( OUT Q )
  u_a/out_reg_3 ( IN SI ) ( OUT Q )
 + STOP buf_2 A
 ;

 - chain_2
 + START buf_1 Y
 + FLOATING
  u_b/out_reg_0 ( IN SI ) ( OUT Q )
  u_b/out_reg_1 ( IN SI ) ( OUT Q )
 + STOP u_b/out_reg_3 SI
 ;

END SCANCHAINS
END DESIGN
```

Related Information

Creating a scanDEF File in *Design for Test in Encounter RTL Compiler*

Affected by this command:	connect_scan_chains on page 523
Affected by these attributes:	Actual Scan Chain attributes

12

Low Power Synthesis

build_rtl_power_models

```
build_rtl_power_models
     [-clean_up_netlist]
     [-clock_gating_logic]
     [-relative instance_list]
     [-design design]
```

Builds detailed power models for more accurate RTL power analysis. The models are used in subsequent RTL power analysis reports.

The power models are MSV and PSO aware.

If you have super-threading enabled, it will be used for power model building.

Options and Arguments

`-clean_up_netlist`	Requests to remove unreachable logic in the netlist as this can affect the accuracy of the estimation. **Note:** Unreachable logic is removed from the input netlist, without changing the logic functionality.
`-clock_gating_logic`	Requests to include power estimation for clock-gating logic. **Note:** Power estimation of clock-gating components can increase runtime considerably.
`-design` *`design`*	Specifies the design for which to build the power models.
`-relative` *`instance_list`*	Builds separate power models for each of the specified hierarchical instances. Models for the top design are built separately at the end.

Example

The following example shows an extract of the messages that are printed in the log file when you build the RTL power models.

```
rc:/> build_rtl_power_models -clean_up_netlist -clock_gating_logic
  Cleaning up the design /designs/mult_bit_muxed_add ...
  Starting building RTL power analysis models ...
  Preprocessing the netlist for building RTL power models ...
  Building RTL power models for top-level design /designs/mult_bit_muxed_add ...
  Building power models for clock gating logic ..
  Done building models for power analysis.
  RTL power modeling has finished.  Use command ’report power’ to see power report.
```

The following example shows the messages that are printed in the log file when you build separate RTL power models for hierarchical instances.

```
rc:/> build_rtl_power_models -clean_up_netlist -relative {mult_1 mult_2}
  Cleaning up the design /designs/test ...
  Starting building RTL power analysis models ...
  Preprocessing the netlist for building RTL power models ...
  Building RTL power models for domain /designs/test/instances_hier/mult_1 ...
  Building RTL power models for domain /designs/test/instances_hier/mult_2 ...
  Building RTL power models for top-level design /designs/test ...
  Done building models for power analysis.
  RTL power modeling has finished.  Use command 'report power' to see power report.
```

Related Information

RTL Power Analysis in *Low Power in Encounter RTL Compiler*

Related command:	report power on page 732
Affected by this attribute:	lp_insert_clock_gating

clock_gating

```
clock_gating
     { connect_test | declone| import | insert_in_netlist
     | insert_obs | join | remove | share | split}
```

Manipulates a netlist for clock gating.

Important

The `clock_gating` commands only work on a mapped netlist.

Options and Arguments

`connect_test`	Connects the test input of all clock-gating logic.
`declone`	Merges clock-gating instances driven by the same inputs.
`import`	Processes clock-gating instances that were either manually inserted or inserted by third-party tools to make them recognizable as clock-gating instances by the RC-LP engine.
`insert_in_netlist`	Inserts clock-gating logic on a mapped netlist.
`insert_obs`	Inserts and connects observability logic.
`join`	Joins multiple-stage clock-gating logic into a single clock-gating instance with a complex enable function.
`remove`	Removes the specified clock-gating logic.
`share`	Extracts the enable function shared by clock-gating logic and inserts shared clock-gating logic with the common enable sub function as the enable signal.
`split`	Splits a single clock-gating instance with a complex enable function into multiple stages of clock-gating logic.

Related Information

Related commands:

clock_gating connect_test on page 703

clock_gating declone on page 704

clock_gating import on page 705

clock_gating insert_in_netlist on page 707

clock_gating insert_obs on page 708

clock_gating join on page 710

clock_gating remove on page 712

clock_gating share on page 714

clock_gating split on page 716

clock_gating connect_test

`clock_gating connect_test`

Globally connects the test input of all clock-gating logic to the test signal specified through the `lp_clock_gating_test_signal` attribute and marks this network as ideal. This command applies to the current design or the current hierarchical instance.

If the clock-gating test input is already connected, the command has no effect.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Example

- The following command connects the test inputs connected to Test1.

  ```
  synthesize
  ...
  set_att lp_clock_gating_test_signal Test1
  ...
  clock_gating connect_test
  ```

Related Information

Clock Gating with DFT in *Low Power in Encounter RTL Compiler*

Scan Insertion after Clock-Gating Insertion in *Low Power in Encounter RTL Compiler*

Related command:	report clock_gating on page 730
Affected by the attribute:	lp_clock_gating_test_signal

clock_gating declone

```
clock_gating declone
     [-hierarchical] [-no_clock_tree_traversal]
```

Merges clock-gating instances driven by the same inputs. The RC-LP engine automatically removes any dangling ports.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Options and Arguments

`-hierarchical`	Allows traversing the design hierarchy to search for clock-gating instances that can be merged. As a result, new ports can be added for the gated-clock signal. By default, this command only affects instances at the current level of the hierarchy.
`-no_clock_tree_traversal`	Prevents traversing through buffers and inverter pairs on the clock signal. If you do not set this option, RTL Compiler can remove buffers or inverter pairs on the clock path during optimization unless the buffers or inverter pairs are marked preserved.

Related Information

Decloning Clock-Gating Instances in *Low Power in Encounter RTL Compiler*

Related command:	report clock_gating on page 730
Affected by this attribute:	lp_clock_gating_max_flops

clock_gating import

```
clock_gating import
     [-start_from instance] [-hierarchical] [-detail]
```

Processes clock-gating instances that were either manually inserted or inserted by third-party tools to make them recognizable as clock-gating instances by the RC-LP engine.

The command returns the total number of instances imported.

Currently, the RC-LP engine recognizes the following structures as clock-gating instance:

- Two-input AND or NAND gates that have a clock signal driving one of the inputs

  In this case, the other pin is assumed to be the enable pin.

  Note: If the other pin is part of the test network, the RC-LP engine does not recognize the gate as a clock-gating instance.

- Integrated clock-gating cells

Currently, the RC-LP engine does not recognize these structures as clock-gating instances if they were defined in a separate module that is instantiated in the netlist.

The RC-LP engine creates a new hierarchical instance (RC_CG_HIER_INST) for each clock-gating instance it recognizes and adds it to the list of clock-gating instances that the RC-LP engine has created.

Options and Arguments

`-detail`	Prints a summary with the names of the imported clock-gating modules and the number of instances found for each clock-gating module.
`-hierarchical`	Allows traversing the design hierarchy to process clock-gating instances that were not inserted by RTL Compiler. By default, this command only affects instances at the current level of the design hierarchy.
`-start_from` *`instance`*	Starts processing clock-gating instances that were not inserted by RTL Compiler from the specified hierarchical instance. By default, the process starts from the current location in the design hierarchy.

Examples

- The following example shows the minimum information listed when clock-gating instances are successfully imported. The number “2” is the total number of instances imported.

```
rc:/> clock_gating import -start_from /designs/top -hier
Importing clock_gating logic from /designs/top
Imported 2 Clock Gating instances

2
```

- The following example shows the additional information listed when the `-detail` option is specified.

```
rc:/> clock_gating import -start_from /designs/top -hier -detail
Importing clock_gating logic from /designs/top

               Detailed report for clock_gating import
 --------------------------------------------------------------------
                   Module name                    |  Import count
 --------------------------------------------------------------------
 a_1                                              |    1
 a                                                |    1
 --------------------------------------------------------------------

Imported 2 Clock Gating instances

2
```

Related Information

Related command:	report clock_gating on page 730

clock_gating insert_in_netlist

```
clock_gating insert_in_netlist
```

Inserts clock-gating logic in a mapped netlist if the D-input of a flip-flop is driven by a two-input MUX and there is a feedback loop from the Q-output to the D-input through one of the data pins of the two-input MUX.

You should only use this command on a netlist that was already mapped (possibly by a third-party tool).

If you set the `lp_clock_gating_test_signal` attribute before you enter this command, the RC-LP engine can connect the test-control signal to the test pins of the clock-gating logic during clock-gating insertion.

Note: This command allows the flip-flops and MUX logic to be in different hierarchies.

Related Information

Inserting Clock Gating in a Mapped Netlist in *Low Power in Encounter RTL Compiler*

Related command:	report clock_gating on page 730
Affected by this attribute:	lp_clock_gating_test_signal

clock_gating insert_obs

```
clock_gating insert_obs
     [-hierarchical] [-make_obs_module]
     [-max_cg integer]
     [-ignore_clock_constraint]
     [-exclude instance...]
     [-disable_clock -libcell libcell]
```

Inserts and connects circuitry to improve the observability of the design after clock-gating logic is inserted. This command applies to the current design or the current hierarchical instance.

Note: To make sure that the enable signal of the clock-gating logic is observable, set the `lp_clock_gating_add_obs_port` design attribute to `true` before you insert the clock-gating logic.

Observability logic is inserted based on clock information. The clock information is required because only clock-gating logic driven by the same clock can share an observation flip-flop. The clock information can be derived from clock constraints or from the physical connectivity.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Options and Arguments

`-disable_clock`	Specifies to gate the clock of the observability flip-flops. **Note:** One gating cell is inserted per flip-flop. The RC-LP engine creates a separate subdesign for each gating cell.
`-exclude` *`instance`*	Prevents insertion of observability logic in the specified hierarchical instances.
`-hierarchical`	Allows insertion and connection of observability logic in the current level of the hierarchy and all its children. By default, this command only affects the current level of the hierarchy.
`-ignore_clock_constraint`	Inserts observability logic based on physical connectivity. By default, observability logic is inserted based on clock constraints defined with the `define_clock` command.

`-libcell` *`libcell`*	Specifies the name of a library cell to be used for gating. **Note:** You can only specify an AND cell to gate the observability logic.
`-make_obs_module`	Creates a separate hierarchy (module) for each observation flip-flop and its associated XOR tree.
`-max_cg` *`integer`*	Specifies the maximum number of clock-gating cells that can be observed per observation flip-flop. Specify an integer between 1 and 32. *Default*: 8

Related Information

Clock Gating with DFT in *Low Power in Encounter RTL Compiler*

Other Flows in *Low Power in Encounter RTL Compiler*

Related commands:	define_clock on page 240
	report clock_gating on page 730
Affected by this attribute:	lp_clock_gating_add_obs_port

clock_gating join

```
clock_gating join
     [-hierarchical] [-max_level integer]
     [-multi_fanouts] [-start_from instance]
```

Combines multiple stages of clock-gating logic into a single clock gating instance with a complex enable function.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Options and Arguments

`-hierarchical`	Allows joining of clock-gating logic down the hierarchy starting from the current directory. By default, this command only affects the current level of the hierarchy.
`-max_level` *`integer`*	Specifies the maximum levels of clock-gating instances that can be combined. If you specify n, n+1 stages can be combined. *Default*: `1` allowing 2 levels to be combined.
`-multi_fanouts`	Allows joining even if the root stage clock-gating instance is driving multiple clock-gating instances.
`-start_from` *`instance`*	Joins the clock-gating logic starting from the specified hierarchical instance.

Examples

- The following command allows joining clock-gating instances across the hierarchy of hierarchical instance `i1`.

  ```
  clock_gating join -hierarchical -start_from [find / -inst i1]
  ```

- The following command allows joining three stages of clock-gating instances across the hierarchy starting from the current directory.

  ```
  clock_gating join -hierarchical -max_level 2
  ```

Related Information

Multi-Stage Clock Gating in *Low Power in Encounter RTL Compiler*

Related command:	report clock_gating on page 730

clock_gating remove

```
clock_gating remove
     [ -hierarchical [-obs_only]
     | -cg_list instance_list [-obs_only]
     | -flops flops]
     [-no_verbose]
     [-effort {low|high}]
```

Removes clock-gating logic inserted by the RC-LP engine from the current design or the current hierarchical instance.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Options and Arguments

`-cg_list` *`instance_list`*

Specifies a list of clock-gating instances to be removed. Use a full path name to identify these instances.

Note: If you specify a clock-gating instance with an incomplete path, the tool searches for that instance from the root of the design hierarchy and might select multiple instances with the same name from different hierarchies.

`-effort {high|low}`	Specifies the effort level.
	Choosing low effort results in better runtime performance but at the cost of an area increase. In this case, the RC-LP engine reconstructs the original MUX and feedback loop from the flip-flop to the MUX.
	For large designs high effort can result in long runtimes, but the feedback logic is optimized.
	Default: `low`
`-flops` *`flops`*	Removes clock gating from the specified flops (that is, recreates the feedback loop for those flops).
	If you specified all flops that are gated by the same clock-gating instance, the clock-gating instance will also be removed.

`-hierarchical`	Removes all clock-gating logic in the hierarchy of the current design or subdesign. By default, this command only affects the current level of the hierarchy.
`-no_verbose`	Suppresses info and warning messages.
`-obs_only`	Removes only the observability logic.

Example

- The following command removes all clock-gating instances in the hierarchy of subdesign `sub1`.

  ```
  rc:/designs/alu/subdesigns/sub1> clock_gating remove -hier
  ```

- The following command removes the clock-gating instances `RC_CG_HIER_INST_121` and `RC_CG_HIER_INST_122` from the current design hierarchy.

  ```
  rc:/> clock_gating remove -cg_list \
  /designs/top/instances_hier/RC_CG_HIER_INST_121 \
  /designs/top/instances_hier/RC_CG_HIER_INST_122

  Clock-gating instance removed   /designs/top/instances_hier/RC_CG_HIER_INST_121
  Clock-gating instance removed   /designs/top/instances_hier/RC_CG_HIER_INST_122
  ```

Related Information

Removing Clock-Gating Instances in *Low Power in Encounter RTL Compiler*

Related command:	report clock_gating on page 730

clock_gating share

```
clock_gating share
     [-hierarchical] [-max_level integer]
     [-max_stage {integer|string}]
```

Extracts the enable function shared by clock-gating logic and inserts shared clock-gating logic with the common enable sub function as the enable signal. The resulting netlist has multiple stages of clock-gating logic.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Options and Arguments

`-hierarchical`	Inserts shared clock-gating logic down the hierarchy starting from the design or current hierarchical instance. By default, this command only affects the current level of the hierarchy.
`-max_level` *`integer`*	Specifies the maximum levels of logic (buffers and inverters excluded) to traverse in the enable fanin of clock-gating instances to extract the common enable function. *Default*: 5
`-max_stage {`*`integer`*`|`*`string`*`}`	Specifies the maximum number of stages of shared clock-gating logic. To specify the same maximum number of stages for all clocks, specify an integer. To specify the maximum number of stages per clock, use a string. The string must have the following format: {{*`clock integer`*}{*`clock integer`*} ...} If this option is not specified, no limit is applied to the number of stages.

Examples

- The following command allows sharing clock-gating logic across the hierarchy starting from the current directory and allows traversing two levels of logic to extract the common enable function.

  ```
  clock_gating share -hierarchical -max_level 2
  ```

- The following command will insert a maximum of 2 stages of shared clock-gating logic for clock `clk1` and a maximum of 3 stages of clock-gating logic for clock `clk2`.

  ```
  clock_gating share -max_stage { {clk1 2} {clk2 3} }
  ```

Related Information

Multi-Stage Clock Gating in *Low Power in Encounter RTL Compiler*

Related command:	report clock_gating on page 730

clock_gating split

```
clock_gating split
     [-hierarchical] [-max_level integer]
     [-power_driven] [-start_from instance]
```

Splits a single clock gating instance with a complex enable function into multiple stages of clock-gating logic.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Options and Arguments

`-hierarchical`	Allows splitting of clock-gating logic down the hierarchy starting from the current directory. By default, this command only affects the current level of the hierarchy.
`-max_level` *`integer`*	Specifies how many times a complex enable function can be split. *Default*: `1` allowing the complex enable function to be split into two stages.
`-power_driven`	Forces to use the signal with smallest toggle rate as the root-level enable. By default, the RC-LP engine considers timing first and uses the late signal as the root-level enable.
`-start_from` *`instance`*	Splits the clock-gating logic starting from the specified hierarchical instance.

Examples

- The following command allows splitting clock-gating instances across the hierarchy of hierarchical instance `i1`.

  ```
  clock_gating split -hierarchical -start_from [find / -inst i1]
  ```

- The following command allows splitting a single clock-gating instance with a complex enable function into three stages of clock-gating instances across the hierarchy starting from the current directory.

  ```
  clock_gating split -hierarchical -max_level 2
  ```

Related Information

Multi-Stage Clock Gating in *Low Power in Encounter RTL Compiler*

Related command:	report clock_gating on page 730

read_saif

```
read_saif [-scale scale_factor]
     [-update [-weight weight_factor]]
     [-verbose] [-instance instance] file
```

Reads switching activity information in Synopsys switching activity interchange format (SAIF) and converts it internally to the Toggle Count Format (TCF) for power estimation.

The `read_saif` command can read files that have been compressed with gzip (`.gz` extension). The `.gz` file is unzipped in memory while the file is read in.

Note: If you read in subsequent SAIF files without the `-update` option, only the probability values and toggle counts of the pins and nets in the current SAIF file are overwritten. The other net values remain unchanged.

The following applies when *updating* the probability values and toggle counts:

- If the probability values and toggle rates were not previously user asserted, the updated probability and toggle rates are determined by the values specified in the SAIF file.
- If the probability and toggle rates were previously user asserted, the new probability and toggle rates are calculated as follows:

  ```
  prob_new = (prob_old + w * prob_spec)/(1+w)
  tr_new = (tr_old + w * tc_spec/duration_spec)/(1+w)
  ```

  where `prob_old` and `tr_old` are the stored values, and `prob_spec`, `tc_spec,` and `duration_spec` are the probably, toggle count, and duration values derived from the new SAIF file.

Options and Arguments

`file`	Specifies the name of the SAIF file. The file can have any name, suffix, or length.
`-instance` *`instance`*	Reads in the switching activities for the specified instance. The instance name can refer to an instance in the design loaded in RC, or can refer to an instance name in the SAIF file.

- If the instance name refers to an instance in RTL Compiler, the RC-LP engine asserts switching activities on that instance in the loaded design.

  In this case, the SAIF file is incomplete and contains only switching activities for the specified instance.
- If the instance name refers to an instance in the SAIF file, the RC-LP engine asserts switching activities on the design loaded in RTL Compiler.

  In this case, a *partial* design is loaded in RTL Compiler. However, the SAIF file contains switching activities for the full design. This SAIF file must be in hierarchical SAIF format.

  Note: The name of the instance in the SAIF file does not need to match the name of the design.

`-scale` *`scale_factor`*

Scales the toggle counts in the SAIF file by dividing them by the specified factor. Use a positive (non-zero) floating number.

Default: `1.0`

`-update` Indicates that you are updating the probability values and toggle counts.

`-verbose` Prints a message for each net that is asserted.

Default: Silent mode. Prints the percent completion messages.

`-weight` *`weight_factor`*

Specifies the relative weight of the probability values and toggle rates in the new SAIF file with respect to the probability values and toggle rates currently stored in the design. Use a positive floating number. This option is only valid with the `-update` option.

Default: `1.0`

Examples

For the following examples, consider the following SAIF file (`and2.saif`):

```
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN "a")
(DATE "date")
(VENDOR "Cadence Design Systems Inc.")
(PROGRAM_NAME "program")
(VERSION "version")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 1000.00)
(INSTANCE a
    (NET
        (in2
            (T0 700) (T1 300) (TC 16)
        )
        ("in1"
            (T0 900) (T1 100) (TC 9)
        )
        (out
            (T0 100) (T1 900) (TC 7)
        )
  )
)
)
```

- The following command reads the SAIF file without any additional options:

  ```
  rc:/> read_saif and2.saif
  ```

  Check the asserted toggle rates on nets in1 and in2:

  ```
  rc:/> get_attr lp_asserted_toggle_rate nets/in1
  0.009000
  ```

  ```
  rc:/> get_attr lp_asserted_toggle_rate nets/in2
  0.016000
  ```

- The following command scales the toggle counts in the SAIF file by a factor 2:

  ```
  rc:/> read_saif and2.saif -scale 2.0
  ```

  Check the asserted toggle rates on nets `in1` and `in2`:

  ```
  rc:/> get_attr lp_asserted_toggle_rate nets/in1

  0.004500

  rc:/> get_attr lp_asserted_toggle_rate nets/in2

  0.008000
  ```

- In the following example, assume you have read in the `and2.saif` file, and you read in the following `and2_new.saif` file:

  ```
  (SAIFILE
  ...
  (TIMESCALE 1 ns)
  (DURATION 1000.00)
  (INSTANCE a
      (NET
          (in1
              (T0 900) (T1 100) (TC 5)
          )
      )
  )
  )
  ```

  The following command updates the stored switching activites with the data in the `and2_new.saif and` gives a two times higher weight on the values in the `and2_new.saif` file.

  ```
  read_saif -update -weight 2 and2_new.saif
  ```

  Check the asserted toggle rates on nets `in1`:

  ```
  rc:/> get_attr lp_asserted_toggle_rate nets/in1
  0.006333
  ```

  This can be caluclated as follows:

  ```
  tr_new = (tr_old + w * tc_spec/duration_new)/(1+w)
         = (0.009000 + 2 * 0.005000)/ (1+2) = 0.006333
  ```

Related Information

Providing Switching Activity Information in *Low Power in Encounter RTL Compiler.*

Affects this command:	report power on page 732
Sets these attributes:	lp_asserted_probability
	lp_asserted_toggle_rate
Related attributes:	lp_probability_type
	lp_toggle_rate_type

read_tcf

```
read_tcf [-scale scale_factor]
     [-update [-weight weight_factor]]
     [-verbose] [-instance instance]
     [-ignorecase] file
```

Reads or updates probability values and toggle counts of the pins and nets in the specified Toggle Count Format (TCF) file and stores the assertions as pin or net attributes, so they can be used for power estimation and optimization.

The `read_tcf` command can read files that have been compressed with gzip (`.gz` extension). The `.gz` file is unzipped in memory while the file is read in.

Note: If you read in subsequent TCF files without the `-update` option, only the probability values and toggle counts of the pins and nets in the current TCF file are overwritten. The other net values remain unchanged.

The following applies when *updating* the probability values and toggle counts:

- If the probability values and toggle rates were not previously user asserted, the updated probability and toggle rates are determined by the values specified in the TCF file.
- If the probability and toggle rates were previously user asserted, the new probability and toggle rates are calculated as follows:

  ```
  prob_new = (prob_old + w * prob_spec)/(1+w)
  tr_new = (tr_old + w * tc_spec/duration_spec)/(1+w)
  ```

  where `prob_old` and `tr_old` are the stored values, and `prob_spec`, `tc_spec,` and `duration_spec` are the probably, toggle count, and duration values specified in the new TCF file.

Options and Arguments

`file`	Specifies the name of the TCF file. The file can have any name, suffix, or length.
`-ignorecase`	Ignores the case of net and pin names in the TCF file when searching for the matching net or pin in the design. By default, case is taken into account. **Note:** Using this option might result in increased run time.

`-instance` *`instance`*

Reads in the switching activities for the specified instance.

The instance name can refer to an instance in the design loaded in RC, or can refer to an instance name in the TCF file.

- If the instance name refers to an instance in RTL Compiler, the RC-LP engine asserts switching activities on that instance in the loaded design.

  In this case, the TCF file is incomplete and contains only switching activities for the specified instance.

- If the instance name refers to an instance in the TCF file, the RC-LP engine asserts switching activities on the design loaded in RTL Compiler.

  In this case, a *partial* design is loaded in RTL Compiler. However, the TCF file contains switching activities for the full design. This TCF file must be in hierarchical TCF format.

  Note: The name of the instance in the TCF file does not need to match the name of the design.

`-scale` *`scale_factor`*

Scales the toggle counts in the TCF file by dividing them by the specified factor. Use a positive (non-zero) floating number.

Default: `1.0`

`-update` Indicates that you are updating the probability values and toggle counts.

`-verbose` Prints a message for each net that is asserted.

Default: Silent mode. Prints the percent completion messages.

`-weight` *`weight_factor`*

Specifies the relative weight of the probability values and toggle rates in the new TCF file with respect to the probability values and toggle rates currently stored in the design. Use a positive floating number. This option is only valid with the `-update` option.

Default: `1.0`

Examples

- Consider the following TCF file (`example1.tcf`):

```
tcffile () {
   tcfversion   :       "1.0";
   duration     :       "1.500000e+05";
   unit         :       "ns";
   instance () {
     pin () {
         "i_12/Z"       :       "0.566 747";
         "n_n1/B"       :       "0.516 475";
         "hier1/i_0/Z"  :       "0.5 500";
         "hier1/n_n0/Z" :       "0.5 500";
         "hier1/n_n0/A" :       "0.5 500";
         "hier1/i_0/A"  :       "0.5 500";
         "hier1/i_0/B"  :       "0.5 500";
         "n_n1/A"       :       "0.61 516";
      }
    }
 }
```

  To read this TCF file (`example1.tcf`), use the following command:

```
rc:/> read_tcf example1.tcf
```

- In the following TCF file (`example2.tcf`), the only difference with the previous TCF file is that the duration in `example2.tcf` is half of the duration in `example1.tcf`.

```
tcffile () {
   tcfversion   :       "1.0";
   duration     :       "0.75000e+05";
   unit         :       "ns";
   instance () {
     pin () {
         "i_12/Z"       :       "0.566 747";
         "n_n1/B"       :       "0.516 475";
         "hier1/i_0/Z"  :       "0.5 500";
         "hier1/n_n0/Z" :       "0.5 500";
         "hier1/n_n0/A" :       "0.5 500";
         "hier1/i_0/A"  :       "0.5 500";
         "hier1/i_0/B"  :       "0.5 500";
         "n_n1/A"       :       "0.61 516";
      }
    }
 }
```

  To make the toggle rates on all pins the same as in the previous example, use the following command:

```
rc:/> read_tcf -scale 2.0 example2.tcf
```

- Consider the following TCF file (`example1.tcf`):

```
tcffile () {
   tcfversion   :       "1.0";
   duration     :       "1.000000e+05";
   unit         :       "ns";
   instance () {
     pin () {
         "i_0/A"        :       "0.5 500";
         "i_0/B"        :       "0.6 600";
         "i_0/Z"        :       "0.7 700";
      }
    }
 }
```

  Assume you read this TCF file with the following command:

```
rc:> read_tcf example1.tcf
```

  Now consider the following TCF file (`example2.tcf`):

```
tcffile () {
   tcfversion   :       "1.0";
   duration     :       "1.500000e+05";
   unit         :       "ns";
   instance () {
     pin () {
        "i_0/A"        :       "0.5 600";
        "i_0/B"        :       "0.6 750";
        "i_0/Z"        :       "0.7 900";
      }
    }
 }
```

  Assume you read this TCF file with the following command:

```
rc:> read_tcf -update -weight 0.5 example2.tcf
```

  You would get the same result by:

  a. Creating the following TCF file (`example3.tcf`)

```
tcffile () {
   tcfversion   :       "1.0";
   duration     :       "1.500000e+05";
   unit         :       "ns";
   instance () {
     pin () {
        "i_0/A"        :       "0.5 700";
        "i_0/B"        :       "0.6 850";
        "i_0/Z"        :       "0.7 1000";
      }
    }
 }
```

  b. Reading the `example3.tcf` file using the following command:

```
read_tcf example3.tcf
```

Related Information

Reading Switching Activity Information from a TCF File in *Low Power in Encounter RTL Compiler*

Checking System Messages when Reading Switching Activities in *Low Power in Encounter RTL Compiler*

TCF Syntax in *Toggle Count Format Reference.*

Affects this command:	report power on page 732
Sets these attributes:	lp_asserted_probability
	lp_asserted_toggle_rate
Related attributes:	lp_probability_type
	lp_toggle_rate_type

read_vcd

```
read_vcd
     [-static
     | -activity_profile [-time_window time]
       [-simvision] [-write_sst2 file] ]
     [-start_time start_monitoring_time]
     [-end_time end_monitoring_time]
     [-module {design|subdesign}] [-vcd_module module]
      vcd_file
```

Reads a Value Change Dump (VCD) file for power analysis. You can

- Perform static power analysis
- Build an activity profile

Note: If no options are specified, static power anlaysis is performed by default.

The `read_vcd` command can read files that have been compressed with gzip (`.gz` extension). The `.gz` file is unzipped while the file is read in.

Options and Arguments

`-activity_profile`

Builds a profile of the activities for the set scope.

By default, profiling is done for the whole design. You can limit the scope to a portion of the design by setting the `lp_dynamic_analysis_scope` attribute to `true` on those instances for which you want to build the profile.

The RC-LP engine captures the toggle count of objects within a given time window. The object can be a net, pin or a hierarchical instance. For a hierarchical instance, the activity will be the sum of the activities of the objects in that instance.

Note: By default, the `read_vcd` command performs static power analysis if neither the `-static` or `-activity_profile` option was specified.

`-end_time` *`end_monitoring_time`*

Specifies the time you want to end monitoring the switching activities or events. Specify a value larger than zero in picoseconds.

By default, the activities or events are monitored till the end (last timestamp of the VCD file).

`-module {`*`design | subdesign`*`}`

Specifies the name of the design or subdesign in the RTL Compiler hierarchy to which the parsed VCD hierarchy (specified through the `-vcd_module` option) corresponds.

For example, if a *partial* design is loaded in RTL Compiler but you have a VCD file that contains switching activities for the full design, the top design in RTL Compiler will correspond to an instance in the VCD hierarchy.

You can also have a full design loaded in RTL Compiler, but only have a partial VCD. In that case you need to specify the name of the subdesign in the RTL Compiler hierarchy to which the VCD file applies.

By default, the VCD file applies to the top design in the RTL Compiler hierarchy. If multiple top designs exists, you must specify the name of the top design.

`-simvision` Invokes SimVision to display the activity profile.

Note: To use this option you need to have SimVision installed and your operating system `PATH` environment variable must include the path to SimVision.

`-start_time` *`start_monitoring_time`*

Specifies the time you want to start monitoring the switching activities or events. Specify a value larger than zero in picoseconds.

By default, the first timestamp in the VCD file is considered as the start time to monitor.

`-static` Calculates the switching activities of each of the nets and pins from the time you want to start monitoring the switching activities to the time you want to stop monitoring, and then stores the information as assertions on the nets and pins.

By default, the `read_vcd` command performs static power analysis if neither the `-static` or `-activity_profile` or option was specified.

`-time_window` *`window`*

Specifies the time increment, in picoseconds, in which you want the RC-LP engine to divide the period during which the events are monitored. The specified time window must be larger than zero.

By default, the time window is calculated based on the setting of the `lp_power_analysis_effort` root attribute and the values of the `-start_time` and `-end_time` options.

`vcd_file` Specifies the name of the value change dump (VCD) file.

`-vcd_module` *`module`*

Starts parsing the VCD hierarchy from the specified module. You can specify to start parsing from the top or from a particular module in the VCD hierarchy.

Default: first scope encountered is used as the top scope

`-write_sst2` *`file`* Specifies the prefix of the SST2 database files to generate to view the data in other waveform viewers.

Examples

- The following command reads `my_vcd.vcd`, generates an activity profile from 10 to 100 ps based on a time window of 10ps, and invokes SimVision to display the activity profile.

```
read_vcd -vcd_module mid2 -activity_profile -start_time 10 -end_time 100 \
-time_window 10 -simvision my_vcd.vcd
```

Related Information

Reading Switching Activity Information from a VCD File in *Low Power in Encounter RTL Compiler*

Affects this command:	report power on page 732
Related attributes:	lp_dynamic_analysis_scope
	lp_power_analysis_effort

report clock_gating

Refer to `report clock_gating` in Chapter 8, “Analysis and Report.”

report operand_isolation

Refer to `report operand_isolation` in Chapter 8, “Analysis and Report.”

report power

Refer to `report power` in Chapter 8, “Analysis and Report.”

state_retention

```
state_retention
     {connect_power_gating_pins | swap}
```

Defines the aspects of mapping to state retention cells.

Options and Arguments

`connect_power_gating_pins`	Connects the power gating pins in a mapped netlist.
`swap`	Replaces sequential cells with their equivalent state retention power gating cells in a mapped netlist.

Related Information

Related commands:	state_retention connect_power_gating_pins on page 734
	state_retention swap on page 735

state_retention connect_power_gating_pins

```
state_retention connect_power_gating_pins
```

Connects the power gating pins according to the driver specifications.

Use this command if you

- Replaced the sequential cells with state-retention cells after mapping and did not specify to hook up the power gating pins at that time (used `state_retention swap` command without the `-connect_power_gating_pins` option).
- Specified the driver specifications (`state_retention define_driver` commands) after mapping (although the mapping instructions were given before mapping)

Related Information

State-Retention Cell Replacement when Starting with Mapped Netlist in *Low Power in Encounter RTL Compiler*

Related attributes:	power_gating_pin_class
	power_gating_pin_phase

state_retention swap

```
state_retention swap
     [-hierarchical]
     [-start_from instance]
     [-connect_power_gating_pins]
```

Replaces sequential cells with their equivalent state retention power gating cells in a mapped netlist.

Options and Arguments

`-connect_power_gating_pins`

Hooks up the power gating pins with their drivers.

The drivers are specified through the `lp_srpg_pg_driver` instance attribute.

`-hierarchical`

Allows traversing the design hierarchy to map.

By default, this command only affects instances at the current level of the design hierarchy.

`-start_from` *`instance`*

Starts replacing sequential cells from the specified hierarchical instance.

By default, the process starts from the current location in the design hierarchy.

Related Information

State-Retention Cell Replacement when Starting with Mapped Netlist in *Low Power in Encounter RTL Compiler*

Affected by these attributes:	hdl_enable_proc_name
	hdl_proc_name
	lp_map_to_srpg_type

write_forward_saif

```
write_forward_saif
     [-library library_path...
     |-library_domain library_domain]
     [ > file ]
```

Prints the library forward SAIF file. This file contains the state-dependent and path-dependent (SDPD) directives needed to generate backward SAIF files during simulation.

Note: You do not need to have any designs loaded to write out a forward SAIF file. You only need to have the libraries loaded.

Options and Arguments

`file`	Specifies the file to which to write the library forward SAIF information. If not specified, the information is written to the screen.
`-library` *`library_path...`*	Specifies the paths to the libraries for which to generate the forward SAIF information. If not specified, the information is generated for all libraries that are loaded.
`-library_domain` *`library_domain`*	Specifies the path to the library domain containing the libraries for which to generate the forward SAIF information. If not specified, the information is generated for all libraries in all library domains. **Note:** This option only applies if you created library domains using the `create_library_domain` command.

Examples

- The following example redirects the forward SAIF information for the `cg` library to the `my.saif` file:

  ```
  write_forward_saif -library /libraries/cg > my.saif
  ```

- The following example redirects the forward SAIF information for the libraries in library domain `d1` to the screen:

  ```
  write_forward_saif -library_domain /libraries/library_domains/d1
  ```

Related Information

Related commands:	create_library_domain on page 753
	read_saif on page 718

write_saif

```
write_saif [-duration simulation_period] [-computed]
     [-boundary_only] [-include_hier_ports] [> file]
```

Writes a hierarchical SAIF file containing the user-asserted or computed (if requested) probability and toggle count of the pins in the design.

By default, the RC-LP engine writes out the user-asserted switching activities of all leaf instance output pins and primary inputs ports.

The `write_saif` command writes out a compressed SAIF file if you add the `.gz` extension to the file name, but is not removed from the directory.

Options and Arguments

`-boundary_only`	Writes out the switching activities of the primary inputs ports and the leaf sequential instance output pins.
`-computed`	Adds the computed probability and toggle count of the pins and nets to the SAIF file. By default only the asserted values are written out.
`-duration` *`simulation_period`*	Modifies the duration for which the toggle count is written in the SAIF file. By default, toggle counts are given for a duration of one second. By modifying the duration, smaller toggle counts (numbers) can be written. For example, if the toggle rate is `3e-3/ns`, the default printed toggle count is 300000. If the simulation period is set to`1e+5 ns`, the printed toggle count will be 300. **Note:** Do not choose the duration too small, otherwise the toggle count will be rounded to 0, because only integer numbers are written out. *Default*: `1e+9 ns`
`file`	Specifies the name of the file to which to write the probability values and toggle count values.
`-include_hier_ports`	Includes the (computed) switching activities for the hierarchical output ports.

Examples

- The following example writes out a SAIF file with the user-asserted switching activities.

  ```
  write_saif > my.saif
  ```

  Note: If you did not read in a TCF or SAIF file, and you did not set toggle rate or probability values on any nets, this SAIF file will not contain any switching activities because you did not request to write out the computed values.

Related Information

Affected by these attributes:	lp_asserted_probability
	lp_asserted_toggle_rate

By default, the command writes out the toggle count for a duration of 1s. For example, if the toggle rate is 3e-3/ns and the simulation period is 1e5 ns, the printed toggle count will be 300

write_tcf

```
write_tcf [-duration simulation_period] [-computed]
     [-hierarchical] [-include_hier_ports] [> file]
```

Writes a TCF file containing the user-asserted or computed (if requested) probability and toggle count of the pins in the design.

By default, the RC-LP engine writes out the user-asserted switching activities of all leaf instance output pins and primary inputs ports.

The `write_tcf` command writes out a compressed TCF file if you add the `.gz` extension to the file name.

Options and Arguments

`-boundary_only`	Writes out the switching activities of the primary inputs ports and the leaf sequential instance output pins.
`-computed`	Adds the computed probability and toggle count of the pins and nets to the TCF file. By default only the asserted values are written out.
`-duration` *`simulation_period`*	Modifies the duration for which the toggle count is written in the TCF file. By default, toggle counts are given for a duration of 1s By modifying the duration, smaller toggle counts can be written. For example, if the toggle rate is 3e-3/ns, the default printed toggle count is 300000. If the simulation period is 1e5 ns, the printed toggle count will be 300. **Note:** Do not choose the period too small, otherwise the toggle count will be rounded to 0, because only integer numbers are written out. *Default*: `1e+9 ns`
`file`	Specifies the name of the file to which to write the probability values and toggle count values.
`-hierarchy`	Writes out a TCF file in hierarchical format. By default, the RC-LP engine writes out a flat TCF file.

`-include_hier_ports`

Includes the (computed) switching activities for the hierarchical output ports.

Examples

- The following example writes out a flat TCF file with the user-asserted switching activities.

  ```
  write_tcf > my.tcf
  ```

  Note: If you did not read in a TCF or SAIF file, and you did not set toggle rate or probability values on any nets, this TCF file will not contain any switching activities because you did not request to write out the computed values.

Related Information

TCF Syntax in *Toggle Count Format Reference*

Related command:	read_tcf on page 722
Affected by these attributes:	lp_asserted_probability
	lp_asserted_toggle_rate

13

Advanced Low Power Synthesis

check_cpf

```
check_cpf
     [-isolation | -level_shifter | -retention | -all]
     [-detail] [-continue_on_error] [> file]
```

Checks the validity of the CPF rules against the RTL of the design. This enables designers to capture any violations of the low power intent of the design early in the design cycle.

- If no low power rule check errors are detected, the command returns `1`.
- If rule check errors are detected, the command returns `0`. In this case, you need to make the necessary changes to your CPF file before proceeding further with synthesis.

To run this command you need to have access to the Encounter ® Conformal ® Low Power software.

Options and Arguments

`-all`	Reports all problems with the CPF file. By default, all problems will be reported.
`-continue_on_error`	Allows the tool to continue when low power rule check errors are encountered.
`-detail`	Provides a detailed report.
`file`	Redirects the report to the specified file.
`-isolation`	Reports only problems with the isolation rules.
`-level_shifter`	Reports only problems with the level-shifter rules.
`-retention`	Reports only problems with the state retention rules.

Examples

The following command reports problems with the level shifter rules.

```
rc:/designs/top> check_cpf -level_shifter
```

```
Using Conformal version xxx.
=================================================================
 CPF LEVEL SHIFTER VIOLATIONS
=================================================================
CPF_LS1: No level shifter rule specified for power domain crossing.
    Severity: Error      Occurrence: 8

Error   : Low Power rule check did not finish successfully. [RCLP-203] [check_cpf]
        : Fix the errors before proceeding further or set the attribute
’clp_treat_errors_as_warnings’ appropriately.
0
```

Related Information

Using CPF for Multiple Supply Voltage Designs in *Low Power in Encounter RTL Compiler*

Using CPF for Designs Using Power Shutoff Methodology in *Low Power in Encounter RTL Compiler*

Interfacing with Conformal Low Power in *Interfacing between Encounter RTL Compiler and Encounter Conformal*

Common Power Format Rule Checks in *Encounter ® Conformal ® Low Power Reference Manual*

Related commands:	read_cpf on page 760

check_library

```
check_library
     [-isolation_cell] [-level_shifter_cell]
     [-retention_cell]
     [-library_domain library_domain_list]
     [-libcell libcell_list]
     [> file]
```

Allows you to check specific information in the loaded libraries with regards to level shifters, isolation cells, and state retention cells. The report also lists the unusable cells. The information returned can be fine tuned by combining several options.

Without any options specified, this command list the number of level shifters, isolation cells, and state retention cells available in each of the library domains. If no library domains exist, the report lists the library names instead.

Options and Arguments

`file`

Specifies the name of the file to which to write out the library information.

`-isolation_cell`

Returns two lists of library cells:

- A list of pure isolation cells with their isolation type
- A list of combo cells with for each cell
  - The isolation type
  - The voltage ranges that they support
  - Whether the combo cell can be used between a lower and higher voltage, or vice versa
  - The valid location for the cell

`-level_shifter_cell`

Returns a list of level shifter cells found in each of the library domains and specifies for each cell

- The supported input and output range
- Whether the level shifter can be used between a lower and higher voltage, or vice versa
- The valid location for the cell

`-library_domain` *`library_domain_list`*

List the number of level shifters, isolation cells, and state retention cells available in each of the specified library domains.

`-libcell` *`libcell_list`*

If not combined with any other option, indicates for each of the specified library cells to which library domain it belongs, whether it is a level shifter, isolation cell, retention cell, always on cell, and the function of the cell.

`-state_retention_cell`

Returns for each library domain the following information:

- A list of sequential elements that have no state-retention equivalent
- A list of state-retention cells available in that domain

Examples

- The following command requests a general check of the libraries that were loaded. When requesting a general check, the report lists the number of usable and unusable level shifters, isolation cells, combo cells, and state retention cells in each library or library domain.

```
rc:/designs/Design2> check_library
```

```
============================================================
  ...
  Module:                 Design2
  Library domain:         lib_074v
    Domain index:         0
    Technology libraries: ....
    Operating conditions: _nominal_ (balanced_tree)
  Library domain:         lib_090v
    Domain index:         1
    Technology libraries: ....
    Operating conditions: _nominal_ (balanced_tree)
  Library domain:         lib_110v
    Domain index:         2
    Technology libraries: ....
    Operating conditions: _nominal_ (balanced_tree)
  Library domain:         lib_120v
    Domain index:         3
    Technology libraries: ...
    Operating conditions: _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================
============================================================================
Library Domain   Total cells   LS cell   ISO cell   Combo (LS+ISO)   SR Flops
----------------------------------------------------------------------------
lib_074v(0.74)   345           2         25         2                5
lib_120v(1.2)    341           0         13         0                5
lib_090v(0.9)    345           2         25         2                5
lib_110v(1.1)    343           2         17         2                5
----------------------------------------------------------------------------

Unusable libcells
=================

============================================================================
Library Domain   Total cells   LS cell   ISO cell   Combo (LS+ISO)   SR Flops
----------------------------------------------------------------------------
lib_074v(0.74)   249           34        45         16               49
lib_120v(1.2)    232           15        47         8                49
lib_090v(0.9)    249           34        45         16               49
lib_110v(1.1)    239           22        47         10               49
----------------------------------------------------------------------------
```

- The following command checks the libraries in library domain `lib_120v`.

```
rc:/> check_library -level_shifter_cell -library_domain lib_120v
============================================================
......
  Library domain:         lib_074v
    Domain index:         0
    Technology libraries: ...
    Operating conditions: BALANC_TREE (balanced_tree)
  Library domain:         lib_120v
    Domain index:         1
    Technology libraries: ...
    Operating conditions: BALANC_TREE (balanced_tree)
  Wireload mode:          enclosed
============================================================


============================================================================
Library Domain   Total cells   LS cell   ISO cell   Combo (LS+ISO)   SR Flops
----------------------------------------------------------------------------
lib_120v(1.2)    11            3         10         2                0
----------------------------------------------------------------------------

Unusable libcells
=================

============================================================================
Library Domain   Total cells   LS cell   ISO cell   Combo (LS+ISO)   SR Flops
----------------------------------------------------------------------------
lib_120v(1.2)    0             0         0          0                0
----------------------------------------------------------------------------
```

- The following command checks the libraries for isolation cells only. In the example below, the cell names in the Combo cell column were abbreviated for documentation purposes to fit the report. The cell names are not abbreviated by the tool.

```
rc:/> check_library -isolation_cell
============================================================
......
  Library domain:         lib_074v
    Domain index:         0
    Technology libraries: ...
    Operating conditions: BALANC_TREE (balanced_tree)
  Library domain:         lib_120v
    Domain index:         1
    Technology libraries: ...
    Operating conditions: BALANC_TREE (balanced_tree)
  Wireload mode:          enclosed
============================================================

Pure isolation cells
====================
========================================================
Library Domain      Isolation Type      Isolation cells
--------------------------------------------------------
lib_074v         enable_high_out_high   OR2XCP
                                        OR2XHP
                                        ....
                                        AND2XCP
                                        AND2XH
                                        ....
lib_120v         enable_high_out_high   OR2XCPPP
                                        OR2XHPPP
                                        ....
                                        AND2XCPPP
                                        AND2XHPPP
                                        ....
--------------------------------------------------------


Combo cells
===========

=========================================================================================
Library   Isolation           Combo         Input       Output        Direction  Location
Domain      type              cell          Range(V)    Range(V)
------------------------------------------------------------------------------------------
lib_074v enable_high_out_high ......BH1XH    0.74-0.74   1.2-1.2       up        to
                              ......BH1XL    0.74-0.74   1.2-1.2       up        to
         enable_low_out_low   ......1CLXH    1.2-1.2     0.74-0.74     down      to
                              ......1CLXL    1.2-1.2     0.74-0.7      down      to
                              ......PPPAD    0.74-0.74   1.2-1.2       up        to
                              ......PPPAD    0.74-0.74   1.2-1.2       up        to
lib_120v enable_high_out_high ......BH1XH    0.9-0.9     1.2-1.2       up        to
         enable_low_out_low   ......1CLXH    0.9-0.9     1.2-1.2       up        to
------------------------------------------------------------------------------------------
```

Note: If the libraries would have unusable isolation cells, the report would list the names of the isolation cells per library or library domain.

- The following command checks the function of library cell `LSHLL1CLXL`.

  ```
  rc:/> check_library -libcell LSHLL1CLXL
  ```

  ```
  ============================================================================
   Library      Libcell      Level   Isolation  Retention  Always   Function
   domain                    shifter  cell       flop       ON
  ----------------------------------------------------------------------------
  lib_074v      LSHLL1CLXL    true    true       false      false    Y = A * B
  ----------------------------------------------------------------------------
  ```

- The following command checks the level shifter characteristics of the specified cell.

  ```
  rc:/> check_library -libcell LSHLL1CLXL -level_shifter_cell
  ```

  ```
  ============================================================
  ......
    Library domain:          lib_074v
      Domain index:          0
      Technology libraries: ...
      Operating conditions: BALANC_TREE (balanced_tree)
    Library domain:          lib_120v
      Domain index:          1
      Technology libraries: ...
      Operating conditions: BALANC_TREE (balanced_tree)
    Wireload mode:           enclosed
  ============================================================


  Level shifter report
  ====================

  =======================================================
   Library     Level shifter      Input      Output    Direction   Location
   Domain          cell          Range(V)    Range(V)
  ---------------------------------------------------------------------------
  lib_074v     LSHLL1CLXL         1.2-1.2    0.74-0.74   down        to
  ---------------------------------------------------------------------------
  ```

- The following command checks the libraries for state retention cells. The report distinguishes between flip-flops and latches. In the example below the library has no latches.

  `rc:/> check_library -retention_cell`

```
============================================================
......
============================================================

Flops without corresponding state-retention flops
--------------------------------------------------

===========================================================================
Library Domain                               Flops
---------------------------------------------------------------------------
110_lib         HD65_LS_DFPHQNX5     HD65_LS_DFPRQNX10    HD65_LS_SDFNRX5
                HD65_LS_SDFPHQNX10   HD65_LS_SDFPSQNX20
---------------------------------------------------------------------------
Usable state-retention flops
--------------------------------------------------


==========================================================================================
Library Domain                                      Flops
---------------------------------------------------------------------------------------
090_lib        HD65_LS_SDFNRX10_SRPG     HD65_LS_SDFNRX5_SRPG
110_lib        HD65_LS_SDFPHQNX10_SRPG   HD65_LS_SDFPHQX10_SRPG    HD65_LS_SDFPQNX10_SRPG
               HD65_LS_SDFPQX10_SRPG     HD65_LS_SDFPRQNX10_SRPG   HD65_LS_SDFPRQX10_SRPG
               HD65_LS_SDFPRSQX10_SRPG   HD65_LS_SDFPSQNX10_SRPG   HD65_LS_SDFPSQX10_SRPG
------------------------------------------------------------------------------------------

Latches without corresponding state-retention latches
-----------------------------------------------------
==========================================================================================
Library Domain                                   Latches
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------

Usable state-retention latches
-----------------------------------------------------
==========================================================================================
Library Domain                                   Latches
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
```

Related Information

Related commands:	read_cpf on page 760

commit_cpf

```
commit_cpf
     [-level_shifter_only]
     [-isolation_only]
```

Inserts level-shifter logic and isolation logic as requested based on the rules specified in previously read in CPF file(s). If specified without any options, both level-shifter logic and isolation logic are inserted.

Options and Arguments

`-level_shifter_only`	Inserts only level-shifter logic according the rules specified in CPF file(s).
`-isolation_only`	Inserts only isolation logic.

Related Information

Using CPF for Multiple Supply Voltage Designs in *Low Power in Encounter RTL Compiler*

Using CPF for Designs Using Power Shutoff Methodology in *Low Power in Encounter RTL Compiler*

Related commands:	read_cpf on page 760
	reload_cpf on page 762

create_library_domain

`create_library_domain` *`domain_list`*

Creates the specified library domains. To use dedicated libraries with portions of the design, you must use this command before you read in any libraries for the specified library domains. The command returns the directory path to the library domains that it creates.

You can find the objects created by the `create_library_domain` command in:

`/libraries/library_domains`

Note: There is no limitation on the number of library domains you can create.

Options and Arguments

`domain_list`	Specifies the names of the library domains to be created. Specify the library domains as a Tcl list.

Examples

- The following example creates three library domains:

```
rc:/> create_library_domain {dom_1 dom_2 dom_3}
```

```
/libraries/library_domains/dom_1 /libraries/library_domains/dom_2 /libraries/
library_domains/dom_3
```

Related Information

Create Library Domains in *Encounter RTL Compiler Synthesis Flows.*

Related attribute:	library

isolation_cell remove

```
isolation_cell remove
     [ iso_hier_instance_list | -hierarchical ]
     [-from_power_domain power_domain_list]
     [-to_power_domain power_domain_list]
```

Removes hierarchical isolation cell instances from the design or the current hierarchical instance. When you combine options only those hierarchical isolation cell instances that meet all criteria are removed. The command returns the number of (hierarchical and leaf) isolation cell instances that were removed.

RTL Compiler can remove any isolation logic that was *inserted* by RTL Compiler.

If the isolation logic was *imported*, RTL Compiler can only remove the hierarchical instance if the following conditions are met:

1. The individual cells in the hierarchy are *all*
   - **a.** Mapped
   - **b.** Of the same type
     - ❍ Pure isolation cells
     - ❍ Pure isolation cells with at most one inverter at their enable
     - ❍ *Combo* cells that combine level shifting and isolation
     - ❍ Discrete cells—`AND` or `OR` cells with or without an inverter at one of the inputs.
2. The enable, data and out pins of each leaf isolation cell contained in isolation hierarchy can be clearly distinguished.

Important

This command does not remove any ports that were created during isolation logic insertion.

Options and Arguments

`-from_power_domain` *`power_domain_list`*

Removes all isolation cell instances whose drivers are output pins of instances in the specified power domains.

`-hierarchical` Removes all isolation cell instances (that meet the from and to criteria) down the hierarchy starting from the current directory.

`iso_hier_instance_list`

Specifies the names of the hierarchical isolation cell instances to be removed.

`-to_power_domain` *`power_domain_list`*

Removes all isolation cell instances whose output pins are driving instances in the specified power domains.

Examples

- The following command removes all isolation cell instances that connect to the top-level of the design.

```
rc:/> isolation_cell remove -hier
Removing isolation cells from '/designs/top' ...
Status
======
Number of isolation cell hierachical instances removed: 8
Isolation cells removed: 8
8
```

- The following command removes all isolation cell instances driving instances in power domain p4.

```
rc:/> isolation_cell remove -to_power_domain p4
Removing isolation cells from '/designs/top' ...
Status
======
Number of isolation cell hierachical instances removed: 3
Isolation cells removed: 3
3
```

- The following command removes two specific isolation cell instances.

```
rc:/> isolation_cell remove {mux_10_14/RC_ISO_HIER_INST_23  \
mux_10_14/RC_ISO_HIER_INST_24}

Removing isolation cells from '/designs/top' ...
Status
======
Number of isolation cell hierachical instances removed: 2
Isolation cells removed: 2
2
```

- The following command removes isolation cell instances that are driven by instances in power domain p1 and that are driving instances in both power domains p2 and p4.

```
rc:/> isolation_cell remove -from_power_domain p1 -to_power_domain {p2 p4}
Removing isolation cells from '/designs/top' ...
Status
======
Number of isolation cell hierachical instances removed: 1
Isolation cells removed: 1
1
```

level_shifter remove

```
level_shifter remove
     { [instance]...
     | [-from_library_domain library_domain...]
       [-to_library_domain library_domain...]
       [-instance_from instance_list]
       [-instance_to instance_list]
       [-hierarchical] }
       [-invalid_only]
```

Removes leaf or hierarchical level-shifter instances. Without any options specified, the command can remove level shifters from the design or the current hierarchical instance. The command returns the number of leaf level-shifter instances that were removed.

The RC-LP engine can remove invalid *imported* level shifters only if you specify the `-invalid_only` option and the output of the imported level shifter is unconnected.

Important

If the `preserve` attribute on a leaf level shifter or its module is set to `true`, or `size_ok`, or `map_size_ok`, it will not be removed. It can only be removed if the `preserve` attribute was set to `delete_ok`, `size_delete_ok`, or `false`.

Options and Arguments

`-from_library_domain` *`library_domain`*

Removes all level shifters whose drivers are output pins of instances in the specified library domain.

`-hierarchical`	Removes all level shifters down the hierarchy starting from the current directory.
`instance`	Specifies the name of a leaf or hierarchical level-shifter instance to be removed.

`-instance_from` *`instance_list`*

Removes all level shifters whose input pin is connected to

- a specified leaf instance
- an instance that is an immediate child of a specified hierarchical instance.

`-instance_to` *`instance_list`*

Removes all level shifters whose output pin is connected to

- a specified leaf instance
- an instance that is an immediate child of a specified hierarchical instance.

`-invalid_only` Removes only level shifters that have become invalid.

`-to_library_domain` *`library_domain`*

Removes all level shifters whose output pins are driving instances in the specified library domain.

Examples

- The following example removes all individual level shifters that connect to the top-level of the design.

```
rc:/> level_shifter remove
Info    : Total level shifter hierarchical instances removed. [LS-102]
        : 1
Info    : Total level shifter cells removed. [LS-103]
        : 12
12
```

- The following example removes all individual level shifters in the design across the hierarchy.

```
rc:/designs/top> level_shifter remove -hier
  level_shifter remove: remove level shifters
Info    : Total level shifter hierarchical instances removed. [LS-102]
        : 2
Info    : Total level shifter cells removed. [LS-103]
        : 13
13
```

- The following example removes the level-shifter instance `RC_LS_HIER_INST_56`.

```
rc:/> level_shifter remove [find / -inst RC_LS_HIER_INST_56]
Info    : Total level shifter hierarchical instances removed. [LS-102]
        : 1
Info    : Total level shifter cells removed. [LS-103]
        : 12
12
```

Note: You can find the list of hierarchical level-shifter instance names in a detailed hierarchical level-shifter report.

- The following example removes all level shifters that have drivers in library domain `07v`.

```
rc:/> level_shifter remove -from [find / -library_domain 07v]
Info    : Total level shifter hierarchical instances removed. [LS-102]
        : 1
Info    : Total level shifter cells removed. [LS-103]
        : 12
12
```

- The following example removes all individual level shifters that connect to instance `outa_reg[6]`.

```
rc:/> level_shifter remove -instance_to [find / -inst outa_reg[6]]
Info    : Total level shifter hierarchical instances removed. [LS-102]
        : 0
Info    : Total level shifter cells removed. [LS-103]
        : 1
1
```

Related Information

Related commands:	report level_shifter on page 765

read_cpf

```
read_cpf [-library]
     [-name_mapping_files file]
     file [file]...
```

Reads the power intent for the design from the specified Common Power Format (CPF) file(s).

Note: In general the `read_cpf` command does not change the netlist. However, the netlist is changed if the CPF file contains

- A `set_instance` command specified with the `-port_mapping` option

  The `-port_mapping` option specifies the mapping of the virtual ports.

- A `set_design` command specified with the `-ports` option

  The `-ports` option specifies a list of virtual ports. These ports do not exist in the definition of the module but will be needed for the control signals of the low power logic such as isolation logic, state-retention logic, and so on.

Options and Arguments

`file`	Specifies the name of the CPF file. The file can have any name, suffix, or length.

`-library`	Instructs to process the following statements in the CPF file:
	`define_isolation_cell`
	`define_level_shifter_cell`
	`define_library_set`
	`define_state_retention_cell`

`-name_mapping_files` *`file`*	Specifies the name of the file in which the changes to instance and pin names have been recorded.

Example

The following example script shows where to use the `read_cpf` command in the flow:

```
...
read_cpf -library my_design.cpf
read_hdl design2.v
elaborate
set_attr output_name_mapping_file name_mapper2.nmf design2
change_names -restricted "\[ \]" -replace_str "_"
read_cpf -name_mapping_files name_mapper2.nmf my_design.cpf
...
commit_cpf
...
```

Related Information

Using CPF for Multiple Supply Voltage Designs in *Low Power in Encounter RTL Compiler*

Using CPF for Designs Using Power Shutoff Methodology in *Low Power in Encounter RTL Compiler*

Using CPF for Designs Using Dynamic Voltage Frequency Scaling in *Low Power in Encounter RTL Compiler*

Related commands:	commit_cpf on page 752
	reload_cpf on page 762
Related attribute:	output_name_mapping_file

reload_cpf

```
reload_cpf
     [-name_mapping_files file]
     [-design design]
```

Applies previously loaded constraints as needed to new design objects (ports/pins/nets/instances) that are created during synthesis.

Options and Arguments

`-design design`	Specifies the design for which you want to reapply the previously loaded constraints. If you omit this option, the constraints will be reapplied to the current design.
`-name_mapping_files file`	Specifies the name of the file in which the changes to instance and pin names have been recorded.

Example

The following example script shows where to use the `reload_cpf` command in the flow:

```
...
read_cpf -library my_design.cpf
read_hdl design2.v
elaborate
...
set_attr output_name_mapping_file name_mapper2.nmf design2
change_names -restricted "\[ \]" -replace_str "_"
read_cpf -name_mapping_files name_mapper2.nmf my_design.cpf
...
synth -to_map -eff medium -no_incr
reload_cpf -name_mapping_files name_mapper2.nmf
commit_cpf
...
```

Related Information

Using CPF for Multiple Supply Voltage Designs in *Low Power in Encounter RTL Compiler*

Using CPF for Designs Using Power Shutoff Methodology in *Low Power in Encounter RTL Compiler*

Related commands:	commit_cpf on page 752
	read_cpf on page 760

report isolation

For more information, refer to report isolation in Chapter 8, “Analysis and Report.”

report level_shifter

For more information, refer to report level_shifter in Chapter 8, “Analysis and Report.”

report state_retention

For more information, refer to report state_retention in Chapter 8, “Analysis and Report.”

verify_power_structure

```
verify_power_structure
     [-isolation] [-level_shifter] [-retention]
     [-pre_synthesis | -post_synthesis]
     [-all] [-detail] [continue_on_error] [> file]
```

Verifies whether the low power cells in the design conform to the rules and specifications in the loaded CPF file. Specifically, RTL Compiler will flag if there are any missing isolation, level shifter, or state retention cells or if the low power cells are not connected appropriately.

To run this command you need to have access to the Encounter ® Conformal ® Low Power software.

Options and Arguments

`-all`	Reports all violations.
`-continue_on_error`	Allows the tool to continue when low power rule check errors are encountered.
`-detail`	Provides a detailed report.
`file`	Redirects the report to the specified file.
`-isolation`	Reports only isolation related violations.
`-level_shifter`	Reports only level-shifter related violations.
`-post_synthesis`	Runs Conformal Low Power, after synthesis, on low power cells.
`-pre_synthesis`	Runs Conformal Low Power, before synthesis, to check the existing low power cells.
`-retention`	Reports only state retention related violations.

Related Information

Using CPF for Multiple Supply Voltage Designs in *Low Power in Encounter RTL Compiler*

Using CPF for Designs Using Power Shutoff Methodology in *Low Power in Encounter RTL Compiler*

Interfacing with Conformal Low Power in *Interfacing between Encounter RTL Compiler and Encounter Conformal*

Related commands:	commit_cpf on page 752
	reload_cpf on page 762

14

Design Exploration

dex_create_exploration_scenarios

```
dex_create_exploration_scenarios
     -design design
```

Creates the exploration scenarios for the specified design. An exploration scenario represents a specific set of operating voltages for the exploration power domains of an MSV design.

You can find the objects created by the `create_exploration_scenarios` command in:

`/designs/`*`design`*`/dex2/scenarios`

The scenarios are named `scenario_`*`n`*.

Options and Arguments

`-design` *`design`*	Specifies the design for which to create the exploration scenarios.

Example

Assume you defined 3 exploration power domains. For two of the power domains, you defined a voltage range that contains 4 voltages, while you keep one power domain at the same voltage. In this case 16 scenarios will be created.

Related Information

Design Exploration in *Low Power in Encounter RTL Compiler*

Related command:	dex_define_exploration_power_domain on page 771

dex_define_exploration_power_domain

```
dex_define_exploration_power_domain
     -name name [-default]
     [-voltage_range string]
     [instance_list] [-design design]
```

Defines an exploration power domain.

An exploration power domain is a power domain for which you try to find the best voltage.

Options and Arguments

`-default`	Indicates whether this is the default power domain.
`-design` *`design`*	Specifies the design to which the power domain belongs.
`instance_list`	Specifies the hierarchical instances that belong to this power domain.
`-name` *`name`*	Specifies the name of the power domain.
`-voltage_range` *`string`*	Specifies the voltage range to be explored for this power domain. Use the following format: `{`*`library_domain1 library_domain2`*`}` In CPF, a library set corresponds to an operating voltage of the design. In RTL Compiler, a library domain corresponds to a library set. To specify the voltage range, you reference the corresponding library domains for the lower and upper voltages. To keep a power domain at a specific voltage, specify the corresponding library domain twice.

Examples

```
dex_define_exploration_power_domain -name default_dom -default \
    -voltage_range {ld0 ld0}
dex_define_exploration_power_domain -name cpu -voltage_range {ld0 ld3} \
    [find / -inst cpu*]
dex_define_exploration_power_domain -name cntrl -voltage_range {ld0 ld3} \
    [find / -inst cntrl*]
```

Related Information

Design Exploration in *Low Power in Encounter RTL Compiler*

Related command:	dex_create_exploration_scenarios on page 770

dex_execute_exploration_scenarios

```
dex_execute_exploration_scenarios -design design
     [-sdc string]
     [-pre_load file] [-post_load file]
```

Executes the different scenarios.

Note: This requires that you can perform super-threading.

Options and Arguments

`-design` *`design`*	Specifies the design for which to execute the exploration scenarios.
`-post_load` *`file`*	Specifies the name of the script that contains any attribute settings or commands you want to use after elaborating the design.
`-pre_load` *`file`*	Specifies the name of the script that contains any attribute settings or commands you want to use before elaborating the design.
`-sdc` *`file`*	Specifies the name of the SDC file to be used for the exploration scenarios.

Example

```
dex_execute_exploration_scenarios -design $DESIGN \
    -sdc cpu.sdc \
    -pre_load dex.preload.tcl \
    -post_load dex.postload.tcl
```

The following is an example of a pre_load script:

```
set_attr super_thread_batch_command {bsub -q normal} /
set_attr super_thread_kill_command {bkill} /
set_attr super_thread_servers {batch batch} /
set_attribute lp_insert_clock_gating true /
set_attr tns_opto true /
set_attr lp_multi_vt_optimization_effort medium /
```

The following is an example of a post_load script:

```
edit_netlist uniquify /designs/*
set_attribute lp_clock_gating_max_flops 64 [find / -design *]
set_attribute lp_clock_gating_min_flops 2 [find / -design *]
set_attr max_leakage_power 0.900 /designs/*
set_attr max_dynamic_power 1.25 /designs/*
```

Related Information

Design Exploration in *Low Power in Encounter RTL Compiler*

Related command:	dex_write_scenario on page 778

dex_report qor_summary

```
dex_report qor_summary
     -design design [> file]
```

Reports the summary for all scenarios that were run. The report shows for each scenario or thread the design area, the number of instances, the slack, the total negative slack, the total leakage power, the net power, the internal power, and the total power.

Options and Arguments

`-design` *`design`*	Specifies the design for which to write the report.
`file`	Specifies the name of the file to which the report must be written.

Example

The following command shows the summary report for 16 threads or scenarios that were run for design cpu_top.

```
rc:/> dex_report qor_summary -design cpu_top
Thread                                      Leakage      Net     Internal    Dynamic      Total
 Num.    Area  Instances   Slack   TNS  Power (nW) Power (nW) Power (nW) Power (nW) Power (nW)
-------------------------------------------------------------------------------------------------
     1 6234.16       632 -1443.60 37425    3405.34   57115.81  346969.48  404085.28  407490.62
     2 6234.16       632 -1443.60 37425    3503.82   57255.88  347965.60  405221.48  408725.30
     3 6234.16       632 -1443.60 37425    3609.77   57426.12  349111.69  406537.80  410147.57
     4 6234.16       632 -1443.60 37425    3820.82   57750.55  351550.87  409301.42  413122.24
     5 6569.52       677  -446.20 10353    4176.43   65126.80  473143.87  538270.66  542447.09
     6 6569.52       677  -446.20 10353    4274.91   65275.51  474143.06  539418.58  543693.48
     7 6569.52       677  -446.20 10353    4380.86   65455.54  475290.63  540746.17  545127.03
     8 6569.52       677  -446.20 10353    4591.91   65800.52  477732.04  543532.57  548124.48
     9 6187.09       582     6.70     0    4678.78   69764.79  608270.96  678035.75  682714.53
    10 6187.09       582     6.70     0    4777.26   69916.19  609271.30  679187.50  683964.76
    11 6187.09       582     6.70     0    4883.21   70099.26  610420.10  680519.37  685402.58
    12 6187.09       582     6.70     0    5094.26   70450.61  612862.58  683313.19  688407.45
    13 5316.33       417    20.10     0    5076.22   62415.03  840851.26  903266.29  908342.51
    14 5316.33       417    20.10     0    5174.71   62556.72  841846.63  904403.35  909578.06
    15 5316.33       417    20.10     0    5280.66   62728.78  842994.57  905723.35  911004.01
    16 5316.33       417    20.10     0    5491.71   63057.05  845436.11  908493.16  913984.87
```

Related Information

Design Exploration in *Low Power in Encounter RTL Compiler*

Related command:	dex_write_scenario on page 778

dex_report thread_info

```
dex_report thread_info
     -design design
     -thread thread_number
     [-depth integer] [> file]
```

Reports the detailed information for the specified thread.

Options and Arguments

`-depth` *`number`*	Specifies the number of hierarchy levels to descend in the report. *Default*: `infinite` (all levels of the hierarchy)
`-design` *`design`*	Specifies the design for which to write the report.
`file`	Specifies the name of the file to which the report must be written.
`-thread thread_`*`number`*	Specifies the specific thread name for which the detailed report must be generated.

Example

The following command shows the detailed report for thread `thread_16` for design cpu_top. The report is shown for all levels of the hierarchy.

```
rc:/> dex_report thread_info -design /des*/* -thread thread_16
-------------------
"thread_16" Summary
----------------------------------------------
Library_domain(Voltage)       Instances
----------------------------------------------
---------------------------------------------------------------------------------------------------------------------------
        Design   Voltage(LD)      Area   Instances   Slack  TNS         Net     Leakage    Internal     Dynamic       Total
                     (volts)                          (ps) (ps)   Power(nW)   Power(nW)   Power(nW)   Power(nW)   Power(nW)
---------------------------------------------------------------------------------------------------------------------------
cpu_top             0.7(ld0)   5316.33         417    20.1    0    63057.05     5491.71   845436.11   908493.16   913984.87
 cntrl_1           1.08(ld3)    368.31          27       -    -      356.04      449.34     2986.74     3342.78     3792.12
 cntrl_2           1.08(ld3)    368.31          27       -    -      574.89      459.52     4569.86     5144.75     5604.27
 cpu_1             1.08(ld3)    840.16          64       -    -     2093.25      922.35   166717.86   168811.12   169733.47
  mul_83_20        1.08(ld3)    429.50          48       -    -     1432.19      457.85     3139.77     4571.96     5029.81
 cpu_2             1.08(ld3)    840.16          64       -    -      277.60      789.41   168920.56   169198.16   169987.57
  mul_83_20        1.08(ld3)    429.50          48       -    -        0.00      383.65        0.00        0.00      383.65
 cpu_3             1.08(ld3)    840.16          64       -    -      723.17      941.78   162172.90   162896.07   163837.85
  mul_83_20        1.08(ld3)    429.50          48       -    -      296.56      466.25      389.98      686.53     1152.78
 cpu_4             1.08(ld3)    840.16          64       -    -      660.33      820.10   168977.64   169637.96   170458.06
  mul_83_20        1.08(ld3)    429.50          48       -    -        0.00      396.92        0.00        0.00      396.92
 cpu_5             1.08(ld3)    840.16          64       -    -     2684.62      915.73   169153.63   171838.24   172753.97
  mul_83_20        1.08(ld3)    429.50          48       -    -     1753.39      448.18     3910.71     5664.10     6112.28
 dma                0.7(ld0)    378.90          43       -    -      918.38      193.48     1936.92     2855.30     3048.78
---------------------------------------------------------------------------------------------------------------------------
```

Related Information

Design Exploration in *Low Power in Encounter RTL Compiler*

Related command:	dex_write_scenario on page 778

dex_write_scenario

```
dex_write_scenario -scenario scenario
     [-prefix string]
```

Generates a basic run script and CPF file for the specified scenario:

- You can use the run script for a complete synthesis run
- The CPF file contains the library set definitions, power domain definitions, nominal condition information and a power mode definition.

Options and Arguments

`-prefix` *`string`*	Specifies the prefix for the script and CPF file that will be generated.
`-scenario` *`scenario`*	Specifies the name of the exploration scenario for which to generate the script and CPF file.

Example

The following command specifies to use `chip_top_` as the prefix for the files to generate for scenario `scenario_3.`

```
dex_write_scenario -prefix chip_top_ -scenario scenario_3
```

Related Information

Design Exploration in *Low Power in Encounter RTL Compiler*

Related command:	dex_execute_exploration_scenarios on page 773

15

Design Manipulation

change_link

```
change_link -instances instance_list
     { -design_name {instance | subdesign | design}
     | -libcell libcell }
     [-pin_map string] [-lenient]
     [-change_in_non_uniq_subd] [-retain_exceptions]
```

Changes the reference of a hierarchical instance to the specified subdesign or design. The command also supports libcell to libcell reference changes as well as a hierarchical instance to libcell changes.

Options and Arguments

`-change_in_non_uniq_subd`	
	Changes the link of the instance(s) in all instantiations of a non-uniquified subdesign.
`-design_name` {*`instance`* | *`subdesign`* | *`design`*}	
	Specifies the design, subdesign, or hierarchical instance to which the link has to be changed.
`-instances` *`instance_list`*	
	Specifies the instance(s) whose reference has to be changed.
`-lenient`	Leaves the pins floating if a pin map is not found.
`-libcell` *`libcell`*	Specifies the library cell to which the instance link has to be changed.
`-pin_map` *`string`*	Specifies, as a Tcl list of lists, the required pin mapping for swapping.
`-retain_exceptions`	Keeps the original exceptions of the instance after replacing it with another link.

Examples

- The following command changes the reference of the hierarchical instances `top/A`, `top/B`, and `top/C` to `patch.`

  ```
  change_link -instances {/designs/top/instances_hier/A \
  /designs/top/instances_hier/B /designs/top/instances_hier/C} \
  -design_name /designs/patch
  ```

- The following example changes the reference of the hierarchical instance `add_0` to the design `add`:

  ```
  rc:/> change_link -design_name add \
      -instance /designs/test/instances_hier/add_0

  CHLNK INFO : Changing link of instance /designs/test/instances_hier/add_0 to
  design /designs/add
  Warning : Uniquifying instance /designs/test/instances_hier/add_0. New
  subdesign is add_1
  ```

- In the following example, `A1`, `A2`, and `A3` are instances of subdesign `A` which is not uniquified. The following command changes a leaf instance in all instances of subdesign `A`.

  ```
  change_link -instances {/designs/top/instances_hier/A1/instances_comb/g1}
  -libcell buf1 -change_in_non_uniq_subd
  ```

  This command changes not only leaf instance `A1/g1` but also `A2/g1` and `A3/g1` with `buf1`.

  Note: If you omit the `-change_in_non_uniq_subd` option, the tool will issue an error.

- The following command replaces hierarchical instance `A1` with `patch` and tries to retain all exceptions of `A1`.

  ```
  change_link -instances {/designs/top/instances_hier/A1} \
  -design_name /designs/patch -retain_exceptions
  ```

  Exceptions defined for instance `A1` are copied to `patch` if the object for which the exception was originally defined is also found in `patch`. For example, an exception defined on `A1/d_reg` will be retained if `d_reg` also exists in `patch`.

- The following command specifies how to map the pins of the hierarchical instance `A` to the pins of design `new`. Instance `A` has pins `a`, `b`, `c`, and `d`. Design `new` has pins `e`, `f`, `g`, and `h`.

  ```
  change_link -instances /designs/top/instances_hier/A \
  -design_name /designs/new -pin_map {{a e} {b f} {c g} {d h}}
  ```

change_names

```
change_names [ -net | -instance | -design | -subdesign
     | -port_bus | -subport_bus]...
     [-local] [-force] [-vhdl] [-verilog]
     [-prefix string [-name_collision]]
     [-suffix string [-name_collision]]
     [-first_restricted string] [-restricted string]
     [-last_restricted string] [-replace_str string]
     [-reserved_words string] [-max_length number]
     |-map string] [-allowed string... [-regexp]]
     [-check_internal_net_name] [-collapse_name_space]
     [-dummy_net_prefix string]
     [-case_insensitive] [-lowertoupper] [-uppertolower]
     [-log_changes file [-append_log]] [hier_instance]
```

Changes names of nets, busses, instances, designs, subdesigns, ports, port buses, and subport buses. You can specify one of more objects. If no object is specified, the requested change applies to all objects unless otherwise specified. By default, all changes are global: changes are made to all objects. There is no restriction on the length of the name.

Options and Arguments

`-allowed` *`string`*	Specifies the characters that are allowed in names. Any characters that are not in the allowed list will be ignored in the resulting names. The minimum specification is 10 characters. To allow all the letters from a to z in capital and lower case letters, you must specify all of them. That is, you cannot use a dash (“-”) to indicate inclusion.
`-append_log`	Appends the information of the last `change_names` command to the logfile specified with the `-log_changes` option If you omit this option, the information of the last `change_names` command overwrites the current information in the logfile. **Note:** You can only specify this option if you specified the `log_changes` option.
`-case_insensitive`	Does not take case sensitivity into account.
`-check_internal_net_name`	Adds the suffix `_int` to any net whose name matches that of a port or subport but is not connected to that port or subport.

`-collapse_name_space`	
	Adds the suffix `_design` to either the port, subport, net, or subdesign in a hierarchy only if they have similar Verilog names.
`-design`	Changes the names of design objects.
`-dummy_net_prefix` *`string`*	
	Specifies the prefix to use for the names of dummy nets when writing out the netlist or HDL.
	Note: This option does not change the names in the design hierarchy.
`-first_restricted` *`string`*	
	Specifies the characters that cannot be used as first character in a name.
`-force`	Forces the name change even if the object name is preserved.
`hier_instance`	Specifies the name of the hierarchical instance to which the changes must be applied.
`-instance`	Changes the names of instance objects.
`-last_restricted` *`string`*	
	Specifies the characters that cannot be used as last character in a name.
`-local`	Restricts the changes to the current directory.
	Default: global changes
`-log_changes` *`file`*	
	Specifies the name of the logfile that shows which names were changed using the `change_names` command and the result of the changes.
`-lowertoupper`	Changes the names of all objects from lowercase to uppercase.
`-map {{"`*`from`*`" "`*`to`*`"}...}`	
	Maps the specified *`from`* character to the specified *`to`* character.
	Enclose each character in double quotes and separate the characters with a space. Enclose each set in braces. If you specify several sets, separate them with spaces and enclose the list of all sets with braces.

`-max_length` *`integer`*	Limits the length of the changed name to the specified number. If the resulting names are longer than the specified integer, the last letters will be truncated.
`-name_collision`	Indicates to only use the specified prefix or suffix values to change object names if a name clash would occur while executing the `change_names` command using other options.
`-net`	Changes the names of net objects.
`-port_bus`	Changes the names of the top-level port bus objects. **Note:** You cannot change the left bracket, "[", and the right bracket, "]", because they are a part of the bus name when referencing individual bits of the bus.
`-prefix` *`string`*	Adds a prefix to the names of the objects to be changed.
`-regexp`	Allows you to specify character ranges with the `-allowed` option.
`-replace_str` *`string`*	Specifies the replacement string. *Default*:_
`-reserved_words` *`string`*	Specifies words to be avoided, such as "begin end".
`-restricted` *`string`*	Specifies the list of strings that cannot be used in a name. Separate independent patterns to be replaced with a “space.” These strings are replaced with the `-replace_str` string.
`-subdesign`	Changes the names of subdesign (object).
`-subport_bus`	Changes the names of subport bus objects. **Note:** You cannot change the left bracket, “[“, and the right bracket, “]”, because they are a part of the bus name when referencing individual bits of the bus.
`-suffix`	Adds a suffix to the names of the objects to be changed.
`-uppertolower`	Changes the names of all objects from uppercase to lowercase.
`-verilog`	Replaces Verilog reserved words.

`-vhdl`	Replaces VHDL reserved words as well as strings that start with a digit, an underscore, continuous (two or more) underscores, and end with an underscore. You do not need to specify the `-case_insensitive`, `-instance`, or `-subdesign` option if you specify the `-vhdl` option.

Examples

- The following example adds a suffix `_t` to the `design` object names:

  ```
  rc:/>change_names -design -suffix _t
  ```

  All `design` objects will now have the `_t` suffix:

  ```
  module top_newName (
      mid m1_t (....)
      mid1 m2_t (....)
      mid3 m3_t (....)
      INVXL g5_t
  endmodule
  ```

  ```
  module mid (...)
      out_reg_7__t (....)
  endmodule
  ```

- The following example replaces all lowercase “n” with uppercase “N”, and all underscores “_” with hyphens “-” in all instance names.

  ```
  rc:/> change_names -instance -map {{"n" "N"} {"_" "-"}}
  ```

- The following example replaces all lowercase “a″ with uppercase “A″ on all subdesigns and subports.

  ```
  rc:/> change_names -map {{"a" "A"}} -subdesign -subport_bus
  ```

- The following example replaces any instances of `ab`, `bc`, or `ca` with “@” in all object names.

  ```
  rc:/> change_names -restricted "ab bc ca" -replace_str "@"
  ```

- The following example specifies the maximum length of all subdesign names to be 12 characters. Issue this command after elaboration or before writing out the netlist.

  ```
  rc:/> change_names -max_length 12 -subdesign
  ```

- The following example ignores case sensitivity. Because the design contains nets `n_73` and `N_73`, RTL Compiler renames one of the nets to avoid a naming conflict.

  ```
  rc:/> mv n_73 N_73
  /designs/alu/nets/N_73
  ```

  ```
  rc:/> mv n_72 n_73
  /designs/alu/nets/n_73
  ```

  ```
  rc:/> change_names -case_insensitive -net
  Info    : Change names is successful [CHNM-102]
           : /designs/alu/nets/n_73 moved to /designs/alu/nets/n_73_1
  ```

- The following example illustrates that you cannot change the brackets (“[“ and “]”) when they are a part of the bus name referencing individual bits of the bus:

```
rc:/designs/test/ports_in> ls
./       SI2      clk1     in1[0]   in2[0]   in2[3]   in3[2]   in3[5]
rc:/designs/test/ports_in> change_names -port_bus -map {{"[" "("} {"]" \
")"}}
rc:/designs/test/ports_in> ls
./       SI2      clk1     in1[0]   in2[0]   in2[3]   in3[2]   in3[5]
```

- The following two commands both change the brackets (“[” and “]”) in the instance name to “x”s:

```
change_names -instance -restricted {[ ]} -replace_str "x"
change_names -instance -restricted "\[ \]" -replace_str "x"
```

Note: There is no need to escape special characters when enclosing them in braces ({}). If you added the escape character inside the braces, the tool would try to replace this character as well with "x".

- The following example allows all capital and lower case letters, numbers, underscores, backslashes, and brackets:

```
rc:/> change_names -allowed \
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_\[\]
```

Note: You cannot use the dash “-” to indicate inclusion. That is, the following example is not allowed:

```
rc:/> change_names -allowed a-zA-Z0-9_[]
```

- The following example creates a separate `change_names` log file, `danni.log`, that reflects the subdesign name change:

```
rc:/> change_names -map {{“SUB” “HERO_SUB”}} -subdesign -log_changes danni.log
```

- The following example shows a part of a netlist that includes VHDL reserved words:

```
generate u1(.A (eg1[0] ), .B (B[0]), .Y (Y[0]));
open u2(.A (eg1[1] ), .B (B[1]), .Y (Y[1]));
endmodule  u3(.A (eg1[2] ), .B (B[2]), .Y (Y[2]));
```

To change, or eliminate, the names of the VHDL reserved words, use the `-vhdl` option.

```
rc:/> change_names -vhdl
```

Now the netlist does not contain any VHDL keywords:

```
generate_cn u1(.A (eg1[0] ), .B (B[0]), .Y (Y[0]));
open_cn u2(.A (eg1[1] ), .B (B[1]), .Y (Y[1]));
endmodule  u3(.A (eg1[2] ), .B (B[2]), .Y (Y[2]));
```

- The following example changes all the object names from lowercase to uppercase:

```
rc:/> ls /designs/areid/ports_in
./    in1[0]    in1[1]    in1[2]    in1[3]    in2[0]    in2[1]    in2[2]
```

```
rc:/> change_names -lowertoupper
Setting in1[3] --> IN1[3
Setting in1[2] --> IN1[2]
Setting in1[1] --> IN1[1]
...
rc:/> ls /designs/AREID/ports_in
./    IN1[0]    IN1[1]    IN1[2]    IN1[3]    IN2[0]    IN2[1]
```

Notice how the lowercase design name change to uppercase as well. The `-uppertolower` option works similarly, except that it changes all uppercase letters to lowercase.

- The following example indicates that the d cannot be used as the first character in a net name and that when a name collision would occur, the prefix `xx` can be used.

```
change_names -name_collision -first_res d -prefix xx
```

- The following example specifies the character ranges that are allowed when renaming instances.

```
change_names -regexp -allowed "a-zA-Z0-9" -instance
```

- The following example uses the `MY_UNCONN_` prefix for dummy nets when writing out the netlist.

```
write_hdl
change_names -dummy_net_prefix MY_UNCONN_
write_hdl
```

Netlist before net name changes:

```
wire [1:0] in1, in2, in3;
wire [5:0] out1, out2;
wire UNCONNECTED, n_0, t;
assign out2[3] = 1’b0;
assign out2[4] = 1’b0;
assign out2[5] = 1’b0;
assign out1[1] = 1’b0;
a a_1(.in1 ({in1[1], 1’b0}), .in2 ({in2[1], 1’b0}), .out1
({out1[5:2], UNCONNECTED, out1[0]}));
```

Netlist after the net name changes:

```
wire \MY_UNCONN_, n_0, t;
assign out2[3] = 1’b0;
assign out2[4] = 1’b0;
assign out2[5] = 1’b0;
assign out1[1] = 1’b0;
a a_1(.in1 ({in1[1], 1’b0}), .in2 ({in2[1], 1’b0}), .out1
       ({out1[5:2], \MY_UNCONN_, out1[0]}));
```

- The following example specifies that inside hierarchical instance `m2,` any changed names cannot end with the character 2.

```
change_names -last_restricted 2 /designs/m1/instances_hier/m2
```

Related Information

Related commands:	write_hdl on page 212

clock_gating

Refer to `clock_gating` in Chapter 12, "Low Power Synthesis."

delete_unloaded_undriven

```
delete_unloaded_undriven
     [-disconnect] [-force_bit_blast] [-all]
     [design] [> file]
```

Disconnects subports and hierarchical pins connected to constants and that do not fanout to anything, and deletes unloaded and undriven subports from the design. Use this command as a post-processing step to remove any unused subports from the netlist.

By default the command skips individual bits of a bus that are connected to constants or that are unused.

Options and Arguments

`-all`	Disconnects subports and hierarchical pins connected to constants and that do not fanout to anything, and deletes unloaded and undriven subports and top-level ports from the design.
`design`	Specifies the design from which to remove the unused ports or subports.
`-disconnect`	Only disconnects the subports and hierarchical pins that are connected to constants and that do not fanout to anything.
`file`	Specifies the name of the file to which the output of the command should be redirected.
`-force_bit_blast`	Bitblasts modules that have individual bus bits that are unused or connected to constants and deletes the unused bus bits.

edit_netlist

```
edit_netlist { bitblast_all_ports | connect
     | dedicate_subdesign | disconnect
     | group | new_design | new_instance
     | new_port_bus | new_primitive | new_subport_bus
     | ungroup | uniquify}
```

Edits a gate-level design.

Options and Arguments

`bitblast_all_ports`	Bitblasts all ports of a design or subdesign
`connect`	Connects a pin, port or subport to another pin, port or subport.
`dedicate_subdesign`	Replaces a subdesign of instances with a dedicated copy.
`disconnect`	Disconnects a pin, port or subport.
`group`	Builds a level of hierarchy around instances.
`new_design`	Creates a new design.
`new_instance`	Creates a new instance.
`new_port_bus`	Creates a new `port_bus` on a design.
`new_primitive`	Creates a new unmapped primitive instance.
`new_subport_bus`	Creates a new `subport_bus` on a hierarchical instance.
`ungroup`	Flattens a level of hierarchy
`uniquify`	Eliminates sharing of subdesigns between instances

Related Information

Related commands:	edit_netlist bitblast_all_ports on page 794
	edit_netlist connect on page 795
	edit_netlist dedicate_subdesign on page 797
	edit_netlist disconnect on page 798
	edit_netlist new_design on page 802

	edit_netlist new_instance on page 803
	edit_netlist new_port_bus on page 805
	edit_netlist new_primitive on page 806
	edit_netlist new_subport_bus on page 808
	edit_netlist uniquify on page 810
	edit_netlist group on page 800
	edit_netlist ungroup on page 809
Affected by this attribute:	ui_respects_preserve

edit_netlist bitblast_all_ports

```
edit_netlist bitblast_all_ports {design|subdesign}...
```

Bitblasts all ports of the specified design or subdesign. This command is available after elaboration. The name of the bitblasted ports will follow the nomenclature specified by the `bit_blasted_port_style` attribute. The default style is:

```
%s_%d
```

Options and Arguments

{*design*|*subdesign*}	Specifies the design or subdesign in which the ports should be bitblasted.

Example

In the following example, the Verilog design `top` has a four-bit input port named `AI`:

```
AI[0:3]
```

The `edit_netlist bitblast_all_ports` command is issued on the design `top`, bitblasting the `AI` port:

```
...
rc:/> synthesize
...
rc:/> edit_netlist bitblast_all_ports top
rc:/> ls /designs/top/ports_in

AI_0 AI_1 AI_2 AI_3
```

Related Information

Affected by this attribute:	bit_blasted_port_style

edit_netlist connect

```
edit_netlist connect
     {pin|port|subport}
     {pin|port|subport}
     [-net_name string]
```

Connects the two specified objects, and anything to which they might already be connected.

You can create nets that have multiple drivers, and you can create combinational loops.

The `logic0` and `logic1` pins are visible in the directory so that you can connect to them and disconnect from them. They are in a directory called `constants` and are called `1` and `0`. The following example shows how the top-level `logic1` pin appears in a design called `add`:

```
/designs/add/constants/1
```

The following example shows how a `logic0` pin appears deeper in the hierarchy:

```
/designs/add/instances_hier/bad/constants/0
```

Each level of hierarchy has its own dedicated logic constants that can only be connected to other objects within that level of hierarchy.

The command has a number of limitations. Violation of the following limitations will generate error messages and cause the command to fail. You cannot connect

- Pins, ports, or subports that are in different levels of hierarchy.
- Pins, ports, or subports that are already connected
- An object to itself.
- An object that is driven by a logic constant to an object that already has a driver. This prevents you from shorting the logic constant nets together.
- Objects if it would require a change to a preserved module.

Options and Arguments

`-net_name` *`string`*	Specifies the user-defined name of the net.
`pin`	Specifies the name of an instance pin to connect.
`port`	Specifies the name of a design port to connect.

`subport`	Specifies the name of a subport (port on a hiearchical instance) to connect.

Examples

- In the following example, `A` and `B` are already connected and `C` and `D` are already connected. When you connect `A` and `C`, the result is a net connecting `A`, `B`, `C`, and `D`.

  ```
  rc:/designs/alu/ports_in> edit_netlist connect A C
  /designs/alu/nets/A_
  ```

Related Information

Related commands:	edit_netlist disconnect on page 798

edit_netlist dedicate_subdesign

```
edit_netlist dedicate_subdesign instance [instance]...
```

Creates a new subdesign by copying the subdesign that is common to the listed hierarchical instances. The command returns the path to the newly created subdesign.

The creation of a new subdesign allows you to make changes that affect a limited set of instances instead of all instances of the original subdesign.

Options and Arguments

instance	Specifies the name of a hierarchical instance for which you want a dedicated subdesign. You must specify a list of instances that share the same subdesign.

Examples

- In the following example the design top contains a module *sub* that has been instantiated five times in the design. The instance names are `sub1`,`sub2`, `sub3`, `sub4`, and `sub5`. To create a separate subdesign for instances `sub1` and `sub2`, enter the following command:

  ```
  rc:/> edit_netlist dedicate_subdesign {/designs/top/instances_hier/sub1 \
      /designs/top/instances_hier/sub2}
  ```

  Note: If you would execute the `edit_netlist dedicate_subdesign` command on the remaining three instances, no new subdesign would be created because they already share a subdesign that is not used by any other instances.

edit_netlist disconnect

`edit_netlist disconnect {`*`pin`*`|`*`port`*`|`*`subport`*`}`

Disconnects a single pin, port, or subport from each object it is connected to. For example, if `A`, `B`, and `C` are connected together and you disconnect `A`, then `B` and `C` remain connected to each other, but `A` is now left unconnected.

You cannot disconnect an object that would require changes to a preserved module.

You cannot disconnect a generic constant (1 or 0) pin of the module but you can disconnect the loads from that pin.

You can disconnect an object that is not currently connected to anything else. In that case nothing happens.

Options and Arguments

`pin`	Specifies the name of an instance pin to disconnect.
`port`	Specifies the name of a design port to disconnect.
`subport`	Specifies the name of a subport (port on a hiearchical instance) to disconnect.

Examples

- The following example disconnects input port `data[4]`.

```
rc:/designs/alu/ports_in> edit_netlist disconnect data[4]
```

- The following example shows how you can disconnect a constant pin 1 from all its loads.

```
set cnet [get_attr net /designs/test/constants/1]
set all_loads [get_attr loads $cnet]
foreach load $all_loads {
    edit_netlist disconnect $load
}
```

  Note: If the constant pin has a large number of loads, disconnecting each of these loads may impact runtime.

Related Information

Related commands:	edit_netlist connect on page 795

edit_netlist group

```
edit_netlist group -group_name group_name
     instance [instance]...
```

Creates a level of the design hierarchy by grouping the specified instances. You can only group instances that belong to the same hierarchy.

Options and Arguments

`-group_name` *`group_name`*	
	Specifies the name of the module that groups the specified instances. The resulting module will have an instance name consisting of the specified group name with the suffix `i`, and is placed in the `instances_hier` directory. The suffix is used to indicate that this hierarchy is the result of a group command.
`instance`	Specifies the name of an instance to add to the specified group. You need to specify at least one instance.

Examples

- The following example groups instances `accum_1` and `averg_1` into one module `my_module`.

  ```
  rc:/> edit_netlist group -group_name my_module accum_1 averg_1
  /designs/alu/instances_hier/my_modulei
  ```

- The following command returns an error because the specified instances do not belong to the same hierarchy.

  ```
  rc:/> edit_netlist group [find / -instance m5] [find / -instance m3_0]
  Error   : Not all instances belong to the same hierarchy. [TUI-234] [edit_netlist
  group]
          : Instance '/designs/m1/instances_hier/m2/instances_hier/m3/
  instances_hier/m4/instances_hier/m5' is part of (sub)design 'm4'. Instance '/
  designs/m1/instances_hier/m2/instances_hier/m3/instances_hier/m3_0' is not part of
  (sub)design 'm4'.
          : The 'edit_netlist group' command can only group instances contained
  within the same hierarchy.
  ```

Related Information

Grouping and Ungrouping Objects in *Using Encounter RTL Compiler*

Related commands:	edit_netlist ungroup on page 809
Related attribute:	group_generate_portname_from_netname

edit_netlist new_design

```
edit_netlist new_design -name string [-quiet]
```

Creates a new design at the same level as the existing top-level design.

The new design is created in the `/designs` directory. Once the design is created, you can specify instances, `port_bus`, and so on.

Options and Arguments

`-name` *`string`*	Specifies the name of the new design.
`-quiet`	Suppresses the warning messages regarding naming conflicts.

Examples

- The following example creates a new design called `DESIGNA`.

  ```
  rc:/> edit_netlist new_design -name DESIGNA
  ```

  The new design `DESIGNA`, will be created in the `/designs` directory. Once the design is created, the instances, `port_bus`, and so on can be specified.

- The following example tries to create a new design called TEST. However, a design by that name already exists. In such cases, a number will be appended to the end of the specified name and an error message indicating the naming conflict will be printed. The following example specifies the `-quiet` option to suppress this warning.

  ```
  rc:/> edit_netlist new_design -name TEST -quiet
  /designs/TEST1
  ```

  The name given to the new design in this case is `TEST1`.

Related Information

Related commands:	edit_netlist new_port_bus on page 805

edit_netlist new_instance

```
edit_netlist new_instance [-name string]
     {design|subdesign|libcell}
     {subdesign|design} [-quiet]
```

Creates a specified instance type in a specified level of design hierarchy. You can instantiate inside a top-level design or a subdesign.

- You cannot instantiate objects that require a change to a preserved module.
- You cannot create a hierarchical loop.

  If subdesign `A` contains subdesign `B`, you cannot instantiate `A` underneath `B`.

Options and Arguments

`-name` *`string`*	Specifies the name of the new instance.
`{`*`design`*`|`*`subdesign`*`|`*`libcell`*`}`	Specifies the object to instantiate.
`-quiet`	Suppresses the warning messages regarding naming conflicts.
`{`*`subdesign`*`|`*`design`*`}`	Specifies the name of the design or subdesign in which you want to instantiate the object.

Examples

- The following example creates a new instance called `TEST_SUB` under the `TEST` design:

  ```
  rc:/> edit_netlist new_instance -name TEST_SUB /designs/TEST \
      /designs/TEST
  rc:/> ls /designs/TEST/instances_hier/
  /designs/TEST/instances_hier/TEST_SUB
  ```

- The following example tries to create a new subdesign called `TEST_SUB` under the `TEST` design. However, a subdesign by that name already exists. In such cases, a number will be appended to the end of the specified name and an error message indicating the naming conflict will be printed. The following example specifies the `-quiet` option to suppress this warning.

  ```
  rc:/> edit_netlist new_instance -name TEST_SUB /designs/TEST -quiet \
          /designs/TEST
  /designs/TEST/instances_hier/TEST_SUB3
  ```

  The name given to the new subdesign in this case is `TEST_SUB3`.

Related Information

Related commands:	edit_netlist new_subport_bus on page 808

edit_netlist new_port_bus

```
edit_netlist new_port_bus -name string
     [-left_bit integer] [-right_bit integer]
     {-input|-output|-input -output}
     [design]
```

Creates a `port_bus` object in a design. The command can also create a single port by omitting both the `-left_bit` and `-right_bit` options.

Options and Arguments

`design`	Specifies the name of the design for which to create the port_bus. The design name can be omitted if there is only one top-level design.
`-input`	Creates an input port_bus.
`-input -output`	Creates a bidirectional port_bus.
`-left_bit integer`	Specifies the leftmost bit index of the bus.
`-name string`	Specifies the name of the new port_bus.
`-output`	Creates an output port_bus.
`-right_bit integer`	Specifies the rightmost bit index of the bus.

Example

- The following example creates a single input port named `a_in`:

  ```
  rc:/> edit_netlist new_port_bus -name a_in -input
  ```

Related Information

Related commands:	edit_netlist new_design on page 802

edit_netlist new_primitive

```
edit_netlist new_primitive [-name string] [-inputs integer]
     [-quiet] logic_function {design|subdesign}
```

Creates an unmapped primitive cell in a design or a subdesign.

Options and Arguments

`{design|subdesign}`	Specifies the name of the design or subdesign in which you want to instantiate the primitive cell.
`-inputs integer`	Specifies the number of input pins to create for the primitive cell.
`logic_function`	Specifies the logic function of the primitive cell. You can specify any of the following:

`and`	`latch`	`notif1`
`buf`	`nand`	`or`
`bufif0`	`nor`	`xnor`
`bufif1`	`not`	`xor`
`d_flop`	`notif0`	

`-name string`	Specifies the name of the new primitive cell.
`-quiet`	Suppresses the warning messages regarding naming conflicts.

Examples

- The following example creates a new buffer instance called `I101` in design `DESIGNA`:

  ```
  rc:/> edit_netlist new_primitive -name I101 buf DESIGNA
  ```

  The new instance, `I101`, is created in the `/designs/DESIGNA/instances_comb` directory.

  A sequential primitive (`d_flop` or `latch`) will be created in the `instances_seq` directory.

- The following example tries to create a new buffer instance called `I101`. However, a buffer by that name already exists. In such cases, a similar name will be chosen and an error message indicating the naming conflict will be printed. The following example specifies the `-quiet` option to suppress this warning:

```
rc:/> edit_netlist new_primitive -name I101 buff TEST -quiet
/designs/TEST/instances_comb/I1
```

The name given to the new buffer in this case is `I1`.

edit_netlist new_subport_bus

```
edit_netlist new_subport_bus -name string
     [-left_bit integer] [-right_bit integer]
     {-input|-output|-input -output}
     instance
```

Creates a `subport_bus` in a design. The command can also create a single subport by omitting both the `-left_bit` and `-right_bit` options.

Options and Arguments

`design`	Specifies the name of the instance for which to create the `subport_bus`.
`-input`	Creates an input `subport_bus`.
`-input -output`	Creates a bidirectional `subport_bus`.
`-left_bit` *`integer`*	Specifies the leftmost bit index of the `subport_bus`.
`-name` *`string`*	Specifies the name of the new `subport_bus`.
`-output`	Creates an output `subport_bus`.
`-right_bit` *`integer`*	Specifies the rightmost bit index of the `subport_bus`.

Examples

- The following example creates a single input subport named `a_in`:

  ```
  rc:/> edit_netlist new_subport_bus -name a_in -input
  ```

Related Information

Related commands:	edit_netlist new_instance on page 803

edit_netlist ungroup

```
edit_netlist ungroup [-prefix string] instance...
```

Removes a level of the design hierarchy.

Large numbers of small hierarchical blocks in a design can sometimes limit optimization since the hierarchical boundaries must be preserved. Many hierarchical blocks may also increase the memory usage since information about each block and port names must be stored. The `ungroup` command provides a mechanism to remove these unwanted levels of hierarchy.

Options and Arguments

`instance`	Specifies the hierarchical instance for which to remove one level of the hierarchy. The components of the specified instance then become instances in the parent block.
`-prefix`	Specifies a prefix for the ungrouped instances.

Examples

- The following example ungroups all hierarchical instances whose names end in `_little`:

  ```
  rc:/> edit_netlist ungroup [find / -instance *_little]
  ```

- The following example ungroups the instance `inst1` and specifies that the resulting ungrouped instances of `inst1` have a prefix of `inst1_test`. Using the prefix allows you to identify from which instance the ungrouped instances originated:

  ```
  rc:/designs/test/instances_hier> edit_netlist ungroup -prefix inst1_test \
      inst1
  rc:/designs/test/instances_comb> ls

  inst1_test_g1/  inst1_test_g2/  inst1_test_g3/
  ```

Related Informations

Grouping and Ungrouping Objects in *Using Encounter RTL Compiler*

Related commands:	edit_netlist group on page 800

edit_netlist uniquify

`edit_netlist uniquify {`*`subdesign`*`|`*`design`*`}`

Uniquifies the instances under the specified design or subdesign. Uniquification is the process of creating a new subdesign for an instance or a group of instances. The newly created subdesign is merely a copy of the subdesign to which the original instance or group of instances were associated. That is, an instance or a group of instances will now be a part of their own, *unique* subdesign. The newly created subdesign will usually maintain the original design or subdesign name followed by a number suffix.

Note: Preserved modules cannot be uniquified.

Options and Arguments

`{`*`design`*`|`*`subdesign`*`}`

The instances under the specified design or subdesigns will be uniquified.

Example

- The following example has a top level design called `top` with two subdesigns named `A` and `B`:

  ```
  rc:/designs/top/subdesigns> ls
  ./     A/     B/
  ```

  In order to uniquify the subdesigns, issue the `edit_netlist uniquify` command on `top`:

  ```
  rc:/designs/top/subdesigns> edit_netlist uniquify /designs/top
  rc:/designs/top/subdesigns> ls
  ./     A/     A_1/     B/     B_1/
  ```

- The following example shows how to uniquify all subdesigns except for subdesign `mysubdesign`.

  ```
  set_attribute preserve true [find / -subdesign mysubdesign]
  foreach el [find / -subdesign *] {
    edit_netlist uniquify $el
  }
  ```

group

```
edit_netlist group -group_name group_name
     instance [instance]...
```

Creates a level of the design hierarchy by grouping the specified instances. You can only group instances that belong to the same hierarchy.

Alias for edit_netlist group.

insert_tiehilo_cells

```
insert_tiehilo_cells [-hilo libcell] [-hi libcell]
     [-lo libcell] [-all] [-maxfanout integer]
     [-verbose] [-skip_unused_hier_pins]
     [subdesign | design]
```

Ties the constants `1’b0` and `1’b1` in the netlist to tie high and tie low cells, respectively. In multiple supply voltage (MSV) designs, this command inserts cells by domain. It skips scan pins, preserved pins, preserved nets, and modules by default. Scan pins can be connected by using the `-all` option.

Use the root level `ui_respects_preserve` attribute to override preserve settings.

Options and Arguments

`-all`	Inserts tie hi/lo cells without skipping scan pins.
`-hi` *`libcell`*	Specifies a cell for constant `1`s. By default, the first appropriate cell will be used.
`-hilo` *`libcell`*	Specifies a high-low cell to connect constants. By default, the first appropriate cell will be used.
`-lo` *`libcell`*	Specifies a cell for constant `0`s. By default, the first appropriate cell will be used.
`-maxfanout` *`integer`*	Specify the maximum fanout allowed per tie cell. If this option is not specified, there is no contraint on the fanout.
`-skip_unused_hier_pins`	Skips hierarchical constant connected pins which are not used inside the module.
{*`subdesign`* | *`design`*}	Specifies the design or subdesign in which to insert constants. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
`-verbose`	Provides detailed information of the preserved and scan pins that were skipped in the tie hi/lo cell insertion process.

Examples

- The following example ties the constant `1`s and `0`s to the cells named `TIEHI` and `TIELOW`, respectively. The maximum fanout per tie cell is 10. Using the `-verbose` option shows that two scan pins were skipped:

```
rc:/> insert_tiehilo_cells -hi TIEHI -lo TIELO  -maxfanout 10 -verbose
```

```
pin: /libraries/slow/libcells/TIELO/Y function: 0
pin: /libraries/slow/libcells/TIEHI/Y function: 1
Connecting all 1'b0 and 1'b1 to TIELO/TIEHI cells.
tielo_cell is /libraries/slow/libcells/TIELO , tiehi_cell is /libraries/slow/
libcells/TIEHI

Info : 2 scan pins which are loads of '0' in /designs/m1 are skipped.
Use the '-all' option to avoid skipping of scan pins.
/designs/m1/instances_seq/foo/bx_reg/pins_in/SE
/designs/m1/instances_seq/tm_reg/pins_in/SE

Done connecting 1'b0 and 1'b1 to TIELO/TIEHI cells
```

- The following example shows two modules, `UI` and `top`. When the `insert_tiehilo_cells -skip_unused_hier_pins` command is used, the pin `B` of the instantiation of `U1` in module `top` will be skipped.

```
module U1(A, B, C, Sel, Z);
    input A, B, C;
    input [1:0] Sel;
    output Z;
    wire A, B, C;
    wire [1:0] Sel;
    wire Z;
    wire n_0, n_1, n_2, n_3, n_4, n_5;
    NAND2X1 g27(.A (n_5), .B (n_4), .Y (Z));
    NAND2X1 g28(.A (n_3), .B (n_2), .Y (n_5));
    NAND2X1 g29(.A (n_1), .B (A), .Y (n_4));
    NOR2X1 g30(.A (n_0), .B (Sel[0]), .Y (n_3));
    INVX1 g31(.A (n_1), .Y (n_2));
    NOR2X1 g32(.A (Sel[1]), .B (Sel[0]), .Y (n_1));
    INVX1 g33(.A (C), .Y (n_0));
endmodule

module top(a, b, c, sel, z);
    input a, b, c;
    input [1:0] sel;
    output z;
    wire a, b, c;
    wire [1:0] sel;
    wire z;
    U1 inst_U1(.A (a), .B (1'b0), .C (c), .Sel (sel), .Z (z));
endmodule
```

Related Information

Removing Assign Statements in *Using Encounter RTL Compiler*

Related attributes:	ui_respects_preserve
	use_tiehilo_for_const

mv

```
mv object new_name [-flexible] [-slash_ok]
```

Renames an object in the design hierarchy. This command is similar to its UNIX counterpart.

You can rename the following objects:

- design
- instance
- isolation_rule
- level_shifter_group
- level_shifter_rule
- library_domain
- net
- port_bus
- power_domain
- scan_chain
- scan_segment
- subdesign
- subport_bus
- test_clock
- test_clock_domain
- test_signal

Options and Arguments

`-flexible`	Indicates to be flexible for renaming when there is a collision.
`new_name`	Specifies the new name for the specified object.
`object`	Specifies the object to rename.
`-slash_ok`	Indicates that the destination name can have embedded slashes.

Examples

- The following example changes the name of design `comp` to `comp_test`.

```
rc:/designs> ls
comp
rc:/designs> mv comp comp_test
rc:/designs> ls
comp_test
```

- The following example changes the subdesign `mux` to `muxYYY`. You do not have to specify the pathname of the target object, just the basename:

```
rc:/> mv /designs/dpldalgn/subdesigns/mux muxYYY
rc:/> ls /designs/dpldalgn/subdesigns/
muxYYY
```

- The following example attempts to rename instance `aluout_reg_0` to an already existing instance `aluout_reg_1`. Using the `-flexible` option, the instance gets renamed to `aluout_reg585`, which does not cause a conflict:

```
rc:/designs/alu/instances_seq> mv aluout_reg_0 aluout_reg_1 -flexible
/designs/alu/instances_seq/aluout_reg585
rc:/designs/alu/instances_seq> ls
./                aluout_reg_1/   aluout_reg_3/   aluout_reg_5/   aluout_reg_7/
aluout_reg585/  aluout_reg_2/   aluout_reg_4/   aluout_reg_6/   zero_reg/
```

Related Information

Related commands:	change_names on page 783

remove_cdn_loop_breaker

```
remove_cdn_loop_breaker
     -instances instance_list design
```

Removes the specified loop breaker buffers added by the timing engine and restores the loop.

Options and Arguments

design	Specifies the design for which the loop breakers must be removed.
-instances *instance_list*	Specifies the loop breakers instances that you want to remove. If this option is not specified, all instance loop breakers will be removed.

Example

The following example first shows the loop breakers inserted, then removes the loop breakers and lists the report again.

```
rc:/> report cdn_loop_breaker
============================================================
  Generated by:           version
  Generated on:           date
  Module:                 loop
  Technology library:     tutorial 1.1
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

   CDN Loop breaker      Driver   Load
---------------------------------------
inst1/cdn_loop_breaker inst1/i1/Y i0/B

rc:/> remove_cdn_loop_breaker -instance [find / -inst inst1/cdn_loop_breaker]
rc:/> report cdn_loop_breaker
============================================================
  Generated by:           Encounter(R) RTL Compiler 8.1.200
  Generated on:           Oct 20 2008  03:53:12 PM
  Module:                 loop
  Technology library:     tutorial 1.1
  Operating conditions:   typical_case (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

  No loop breakers to report
```

Related Information

Related command:	report cdn_loop_breaker on page 336

reset_design

`reset_design [-verbose] [`*`design`*`]`

Removes all the user specified timing and DFT objects and returns all attributes to their default values for the specified design. Alternatively, this command removes every clock domain, cost group, exception, external delay, scan chain, scan segment, test clock domain, and test signal while returning all attributes on the design to their default values.

Options and Arguments

`design`	Specifies the particular design to reset when there are multiple designs.
`-verbose`	Prints messages indicating that the command was successful.

Example

- The following examples illustrates that the `reset_design` command has eliminated all external delays in the design:

```
rc:/designs/phoenix/timing/external_delays> ls
in_1  out_2

rc:/designs/phoenix/timing/external_delays> reset_design phoenix
rc:/designs/phoenix/timing/external_delays> ls
./
```

rm

```
rm object... [-quiet]
```

Removes an object from the design hierarchy. This command is similar to its UNIX counterpart.

You can remove *timing* objects and any of the following objects:

- actual_scan_chain
- designs
- instances_comb
- instances_hier
- instances_seq
- isolation_rule
- level_shifter_group
- library_domain
- pin_busses_in
- pin_busses_out
- port_busses_out
- port_busses_out
- power_domain
- scan_chain
- scan_segment
- subport_busses_in
- subport_busses_out
- test_clock
- test_clock_domain
- test_signal

If the hierarchical pin or port bus object has a net connection, the net is disconnected first and then the object is removed.

Note: The `rm` command does not work on the `pin` or `port` object.

Options and Arguments

`object`	Specifies the object that you want to remove.
`-quiet`	Suppresses those messages that indicate which objects are being removed. Alternatively, when removing an object, an information message will not be printed.

Examples

- The following example finds the clock objects in the design:

```
rc:/> find . -clock *
/designs/comp/timing/clock_domains/domain_1/clock1
```

- The following example uses the result of the `find` command to remove the clock. This command also removes all dependent objects. A subsequent `find` cannot find any clock objects.

```
rc:/> rm [find . -clock *]
Info    : Removing a clock object [TIM-102]
        : The clock name is ’clock’
    Removing external delay ’in_del_1’.
    Removing external delay ’ou_del_1’.
```

```
rc:/> find . -clock *
I cannot find any clock named * here
Failed on find . -clock *
```

ungroup

```
ungroup
     [ -all | -flatten instance...
     | -threshold integer | instance...]
     [-simple] [-only_user_hierarchy] [-exclude instance...]
```

Ungroups the specified instances. Ungrouping dissolves the hierarchy and moves the contents of a subdesign into its parent directory. By default, an instance is named by concatenating its name to its parent's name.

Options and Arguments

`-all`	Ungroups all instances at the current level.
`-exclude`	Specifies a list of instances that should not be ungrouped.
`-flatten`	Recursively ungroups all the specified instances.
`instance`	Specifies the instance or instances to be ungrouped.
`-only_user_hierarchy`	Ungroups only those hierarchies created by the user. The hierarchies created by the tool are preserved.
`-simple`	Prevents the use of a complex new instance name during ungrouping. This option has the same effect as the `edit_netlist ungroup` command.
`-threshold`	Ungroups only those hierarchical instances that have a cell count equal to or less than the specified integer.

Example

- The following example ungroups the instance named `CRITICAL_GROUP`:

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

- In the following example, every instance under the `inst` hierarchical instance will be ungrouped except the `inst_sub2` instance:

  ```
  rc:/designs/ksable_hier/instances_hier/inst/instances_hier> ls
  ./      inst_sub1/      inst_sub2/      inst_sub3
  rc:/designs/ksable_hier/instances_hier/> ungroup inst -flatten -exclude \
      [find . -instance inst/inst_sub2]
  ```

Related Information

Related commands:	edit_netlist ungroup on page 809

uniquify

```
edit_netlist uniquify {subdesign|design}
```

Uniquifies the instances under the specified design or subdesign. Uniquification is the process of creating a new subdesign for an instance or a group of instances. The newly created subdesign is merely a copy of the subdesign to which the original instance or group of instances were associated. That is, an instance or a group of instances will now be a part of their own, *unique* subdesign. The newly created subdesign will usually maintain the original design or subdesign name followed by a number suffix.

Note: Preserved modules cannot be uniquified.

Alias for edit_netlist uniquify.

16

Customization

add_command_help

`add_command_help` *`command_name help category`*

Adds a short help message for a new command to RTL Compiler's online help system. Examples of `add_command_help` can be found in the installation `lib/cdn/rc` directory.

Options and Arguments

`category`	Specifies the category to which this command should be added. You can specify an existing category or a new one. To see a list of all existing categories, type help.
`command_name`	Specifies the name of the command. It must be a Tcl procedure that you defined previously.
`help`	Lists the help text to be displayed when the `help` command is used. Use a string.

Examples

- The following example adds help for the `hello_world` command:

```
rc:/> proc hello_world {hello_world} { echo "Hello world" }

rc:/> add_command_help "hello_world" "Says hello to the whole world"
my_category

rc:/> help hello_world
That command is:

        my_category
==========================================================================
hello_world  Says hello to the whole world

Command details:

Hello world
```

define_attribute

```
define_attribute
     -category string -data_type string -obj_type string
     [-hidden] [-help_string string]
     [-check_function string] [-compute_function string]
     [-default_value string]
     [-set_function string] string
```

Creates a new, user-defined attribute with the specified characteristics. These attributes will also be written out if you use the `write_script` command.

Options and Arguments

`-category` *`string`*

Defines the category of the attribute. Categories group attributes that perform similar functions whereas object types describe where in the design an attribute is valid. You can specify any category name: both new and existing category names are valid.

`-check_function` *`string`*

Specifies a previously defined Tcl procedure's name in order to ensure that the newly defined attribute is valid. The Tcl procedure should be of the form:

`proc` {*`object value`*}

The Tcl procedure returns `1` for a valid value, and `0` for an invalid value.

`-compute_function` *`string`*

Specifies a previously defined Tcl procedure's name in order to get the newly defined attribute's value later (with the `get_attribute`) command. The Tcl procedure should be of the form:

`proc` {*`object`*}

When you use this option, the attribute becomes a read-only attribute because its value is always computed.

`-data_type` *`string`*	Defines the data type of the attribute. Possible data types are: ■ boolean ■ "fixed point" ■ "floating point number" ■ integer ■ string
`-default_value` *`string`*	Specifies a default value for the attribute.
`-help_string` *`string`*	Specifies the help text for the attribute.
`-hidden`	Specifies whether the defined attribute is a hidden attribute.
`-more_help_string` *`string`*	Specifies an extended help string.
`-obj_type` *`string`*	Specifies the object type of the attribute. All valid object types can be found by typing `find -help` at the RTL Compiler prompt.
`-obsolete`	Specifies whether the defined attribute is obsolete.
`-set_function`	Specifies a previously defined Tcl procedure's name. This option allows you to override user-defined values provided it conforms to the parameters in the Tcl procedure you created. The Tcl procedure should be of the form: `proc {`*`object new_value current_value`*`}`
`string`	Specifies the name of the attribute.

Examples

- The following example defines a boolean attribute named `new_libpin` for a new category named `my_libpin`:

```
rc:/> define_attribute -data_type boolean -obj_type libpin -category my_libpin
new_libpin
rc:/> get_attribute new_libpin * -help
...
attribute category: my_libpin

    attribute name: new_libpin
          category: my_libpin ()
       object type: libpin
       access type: read-write
         data type: boolean
     default value:
              help:
```

- The following example creates a Tcl procedure, `check_fxn`, and then creates an attribute named `test_check`. The `-check_function` option is specified so that the `test_check` attribute can be tested for validity when it is specified later.

```
rc:/> proc check_fxn {obj val} {
==>   if {$val < 0} {
==>     return 0
==>   }
==>   return 1
==> }

rc:/> define_attribute test_check -obj_type root -data_type integer \
    -category test -help_string "test check function" \
    -check_function check_fxn
/object_types/root/attributes/test_check
```

The following command is a valid use of the newly created `test_check` attribute:

```
rc:/> set_attribute test_check 1 /
Setting attribute of root '/': 'test_check' = 1
```

The following command would be an invalid use:

```
rc:/> set_attribute test_check -1 /
Error  : The data value for this attribute is invalid. [TUI-24] [set_attribute]
       : The value '-1' cannot be set for attribute 'test_check'.
       : To see the usage/description for this attribute, type 'set_attribute
-h <attr_name> *'.
```

- The following example creates a Tcl procedure, `set_fxn`, and then creates an attribute named `test_set`. The `-set_function` option is specified so that you can change the value of the `test_set` attribute (provided it is valid):

```
rc:/> proc set_fxn {obj new_val cur_val} {
==>   if {$new_val > $cur_val} {
==>     return $new_val
==>   }
==>   return $cur_val
==> }
```

```
rc:/> define_attribute test_set -obj_type root -data_type integer \
    -category test -help_string "test set function" -set_function set_fxn

/object_types/root/attributes/test_set
```

The `test_set` attribute will be changed to `1`. It is valid, since there was no previous value:

```
rc:/> set_attribute test_set 1
Setting attribute of root '/': 'test_set' = 1
```

The following command chnages the attribute value to `2`. Again, this is valid because it falls within the definition of the previously defined Tcl procedure, `set_fxn`:

```
rc:/> set_attribute test_set 2 /
Setting attribute of root '/': 'test_set' = 2
```

The attribute's value will not be changed in the following example. The value will remain at `2`:

```
rc:/> set_attribute test_set 0 /
Setting attribute of root '/': 'test_set' = 2
```

- The following example creates a Tcl procedure, `compute_fxn`, which always returns the value of `42`. The `define_attribute` command then creates an attribute named `test_compute`. The `-compute_function` option is specified so that you can obtain the `test_compute` attribute's value later:

```
rc:/> proc compute_fxn {obj} {
==>   return 42
==> }
rc:/> define_attribute test_compute -obj_type root -data_type integer \
    -category test -help_string "test compute function" \
    -compute_function compute_fxn

/object_types/root/attributes/test_compute
```

The `test_compute` value will always be `42`, as defined in the Tcl procedure:

```
rc:/> get_attribute test_compute /
42

rc:/> set_attribute test_compute 23 /
Error   : The attribute is read-only. [TUI-26] [set_attribute]
        : attribute: 'test_compute', object type: 'root'
        : Cannot set or reset read-only attributes.
Failed on set_attribute test_compute 23
```

mesg_make

```
mesg_make -group string [-internal_group] -id number
     -short_desc string -long_desc string
     {-error|-warning|-info_priority number}
```

Creates a custom message that can subsequently be accessed with the `mesg_send` command.

Options and Arguments

`-error`	Creates an error message.
`-group` *`string`*	Specifies the group of messages that the new message belongs to. A group groups messages that apply to a certain engine of the tool. For example, the MAP group groups messages issues by the mapper. **Note:** If you want to create a message for an internal group, you must specify an existing group name.
`-id` *`integer`*	Specifies an identification number for the message. The number must be unique for the specified group. If the specified number already exists, you will overwrite the existing message. *Default*: `1`
`-info_priority` *`integer`*	Creates an info message with the specified priority. You can specify a number between 2 and 8.
`-internal_group`	Specifies whether the message belongs to an internal group.
`-long_desc` *`string`*	Defines the help of the message.
`-short_desc` *`string`*	Specifies the title or the description of the message.
`-warning`	Creates a warning message.

Example

- The following example creates a message in the `test` group and assigns it a unique identification number (501) within that group:

```
rc:/> mesg_make -group test  -id 501 -short_desc note_bene \
==> -long_desc "search for lost time" -warning
/messages/test/test-501
rc:/> man test-501
Entry       : test-501
Severity    : Warning
Verbosity   : Message is visible at any ’information_level’ above ’1’.
Description : note_bene
Help        : search for lost time
```

Related Information

Affects this command: mesg_send on page 833

mesg_send

`mesg_send` *`message string`*

Accesses the various native messages of RTL Compiler and the messages that were created with the `mesg_make` command.

Options and Arguments

`-caller` *`string`*	Specifies the name of the calling procedure.
`-file_info` *`string`*	Specifies to which file the messsage applies.
`message`	Identifies the message to be sent. The identification is in the form of *`group`*-*`id`*. Refer to `mesg_make` for an explanation of *`group`* and *`id.`*
`-newline`	Prints a new (empty) line before the message
`-object_info` *`string`*	Prints the object type and
`string`	Describes the context-spsecific help.

Examples

- The following example accesses the message created in Example on page 832 for `mesg_make`:

```
rc:/messages/test> mesg_send test-501 "reminders" -newline

Warning : note_bene [test-501]
        : reminders
        : search for lost time
```

- The following example sends a message after elaboration:

```
rc:/> mesg_send /messages/VHDLPT/VHDLPT-500 "file not found" -caller read_hdl
Error   : Cannot open file. [VHDLPT-500] [read_hdl]
        : file not found
```

Related Information

Affected by this command:	mesg_make on page 831

parse_options

`parse_options` *`cmd file_var`* `[`*`args`*`] [`*`code var`*`]...`

Interfaces to the RTL Compiler internal command option parser. The RTL Compiler argument parser provides the following features:

- Checking the correctness for arguments and types. Appropriate messages are issued when the input is incorrect.
- No specific order of arguments is required. For example, `ls -long -attribute` behaves just like `ls -attribute -long`.
- Unique abbreviations of arguments is supported. For example, `ls -l` behaves just like ls `-long`.
- Online help is provided if `-help` is specified.
- Optional file redirection is supported. For example, `report gates >> design.rpt` causes output to be appended to the file `design.rpt`.
- RTL Compiler objects can be implicitly searched for based on their type. For example, `fanout SUB/A[0]` performs an implicit find on the string `SUB/A[0]` and locates the object `/designs/TOP/instances_hier/SUB/pins_in/A[0]`.

You can use the command option parser to make a Tcl procedure behave just like a built-in RTL Compiler command by providing on-line help, finding objects automatically, checking for required options, handling unique argument abbreviations, and handling file redirection.

This command can return one of the following values:

- -2: You asked for help and help was provided. The command returns normally.
- 0: Your options were invalid and the command aborts.
- 1: Your options were valid and the command continues normally.

Options and Arguments

`args`

Specifies a list containing the options that the user sends to your command. Usually your procedure will be defined like this:

```
proc your_procedure {args} {
...  (code that implements the procedure)
}
```

You would then pass `$args` as the args parameter to `parse_options` within your procedure.

`cmd`	Specifies the name of the command whose options to parse. This name appears in the help listing if a user calls your procedure with `-h` or if a user does not provide valid arguments.
`code`	Specifies a string that describes the command argument: flag name, whether it is required or optional, what type of data it accepts, and a short help message about it. The string must be in the form: `"(-<name>)?<x><y><z>(<dirtypes>)? <help>"` The question marks in the above string mean that these fields of the string are optional. You cannot specify multiple string values if you are specifying a flag name. That is, if you are specifying a flag, you cannot also specify the `som` (string, optional, multiple values) or `srm` (string, required, multiple values) combinations. `<`*`name`*`>` is the name of the flag. For example, `-number`. `<`*`x`*`>` is a single character indicating the type of the option:

b	Boolean
d	directory object
`f`	float
n	integer number
s	string

`<`*`y`*`>` is a single character indicating whether the option is optional or required

o	optional
`r`	required
x	obsolete

`<`*`z`*`>` is a single character indicating whether lists are accepted

`m`	Accepts multiple values: lists OK
s	Accepts single value only: no lists

	<*dirtypes*> is a string indicating the types of directory objects the option accepts. This string is required for all arguments that have the <*x*> field set to d and cannot be specified otherwise. The list of specified directory types can only be separated by vertical bars (|), that is, no spaces allowed, and must be enclosed in parentheses. <*help*> is a string that indicates to the user of your command what the purpose of the option is.
file_var	Specifies the name of a variable that holds the file handle if the user calls your procedure with > *file* or >> *file*. Your procedure should always check to see if this variable has been set to something other than 'stdout'. If it has, then your command should send its output to that file handle instead of 'stdout' and you should close the file handle once your command is complete.
var	Specifies the name of the variable that will be set with the parsed result for that argument.

Examples

- The following example shows how file indirection works:

```
rc:/> parse_options hello_world file_var {> ./tmp}
1
rc:/> puts $file_var "Hello world!"
rc:/> close $file_var
```

- The following example shows a required argument with the `-design` flag that must be the name of a subdesign (block).

```
rc:/> parse_options hello_world file_var {-design addinc65} \
==> "-design drs(subdesign) A module"   my_design
1
rc:/> puts $my_design
/designs/alu/subdesigns/addinc65
```

- The following example shows a Boolean argument, a numeric argument, and an un-flagged object argument that can be either a subdesign (block) or an instance. It also shows how the parser accepts abbreviations since the argument passed in is only `-t`, it still gets recognized as `-top`.

```
rc:/> set level 300
300
rc:/> parse_options hello_world file_var {-t addinc65} \
==>  "-top bos Do the top level thing"  top \
==>  "-level nos The level to work on"  level \
==>  "dos(subdesign|instance) An object to work on"   object
1
```

```
rc:/> puts $object
/designs/alu/subdesigns/addinc65
```

```
rc:/> puts $top
1
```

```
rc:/> puts $level
300
```

- The following example shows how online help works:

```
rc:/> parse_options hello_world file_var {-h} \
==>  "-top bos Do the top level thing"  top \
==>  "-level nos The level to work on"  level \
==>  "dos(subdesign|instance) An object to work on"   object
 Usage: hello_world [-h]
          -h: this message
        hello_world [-top] [-level number] [instance|subdesign] [> file]
          -top: (Boolean) Do the top level thing
          -level:       (integer) The level to work on
                        (instance|subdesign) An object to work on
-2
```

Index

Note: See Alphabetical List of Commands for an index of the command names.