## Contents

**Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_rename_rule</td>
<td>1-6</td>
</tr>
<tr>
<td>alias</td>
<td>1-10</td>
</tr>
<tr>
<td>analyze</td>
<td>1-12</td>
</tr>
<tr>
<td>apply_rename_rules</td>
<td>1-15</td>
</tr>
<tr>
<td>balance_loads</td>
<td>1-16</td>
</tr>
<tr>
<td>connect</td>
<td>1-18</td>
</tr>
<tr>
<td>connect_path</td>
<td>1-20</td>
</tr>
<tr>
<td>copy</td>
<td>1-22</td>
</tr>
<tr>
<td>create</td>
<td>1-24</td>
</tr>
<tr>
<td>create_rename_ruleset</td>
<td>1-26</td>
</tr>
<tr>
<td>create_wrapper</td>
<td>1-27</td>
</tr>
<tr>
<td>decompose_luts</td>
<td>1-29</td>
</tr>
<tr>
<td>disconnect</td>
<td>1-31</td>
</tr>
<tr>
<td>disconnect_path</td>
<td>1-33</td>
</tr>
<tr>
<td>elaborate</td>
<td>1-35</td>
</tr>
<tr>
<td>generate_timespec</td>
<td>1-38</td>
</tr>
<tr>
<td>group</td>
<td>1-40</td>
</tr>
<tr>
<td>help</td>
<td>1-42</td>
</tr>
<tr>
<td>list_attributes</td>
<td>1-44</td>
</tr>
<tr>
<td>list_connection</td>
<td>1-46</td>
</tr>
<tr>
<td>list_design</td>
<td>1-49</td>
</tr>
<tr>
<td>list_technologies</td>
<td>1-52</td>
</tr>
<tr>
<td>load_library</td>
<td>1-53</td>
</tr>
<tr>
<td>load_modgen</td>
<td>1-55</td>
</tr>
<tr>
<td>move</td>
<td>1-57</td>
</tr>
<tr>
<td>optimize</td>
<td>1-60</td>
</tr>
</tbody>
</table>
optimize_timing ................................................................. 1-66
pack_clbs ............................................................................. 1-69
pre_optimize ......................................................................... 1-71
present_design ...................................................................... 1-73
read ...................................................................................... 1-74
remove ................................................................................ 1-79
remove_attribute ................................................................. 1-81
remove_clock ...................................................................... 1-83
remove_rename_ruleset ....................................................... 1-84
report_area ......................................................................... 1-85
report_constraints .............................................................. 1-87
report_delay ........................................................................ 1-89
report_rename_rules ............................................................ 1-93
report_wire_tables .............................................................. 1-94
set_attribute ........................................................................ 1-95
set_clock ............................................................................. 1-97
set_multicycle_path ........................................................... 1-98
unalias ................................................................................ 1-100
unfold ................................................................................. 1-101
ungroup .............................................................................. 1-102
unmap ................................................................................ 1-104
write .................................................................................. 1-105

Variables ............................................................................. 2-1

Utilities .............................................................................. 3-1

Attributes............................................................................ 4-1
Commands

The commands listed in this chapter and the Tcl script methods for using the commands are available to Levels 1, 2, and 3 with the following restrictions:

- The **group**, **unfold**, **analyze**, and **elaborate** commands are available only to **Level 3**.
- The interactive command line shell (GUI window) is available only to **Level 3**.

This chapter presents the following:

- Methods for Using Commands with Tcl script
- Command Line Description

Methods for Using Commands With Tcl Script

After you create a Tcl script in a standard text editor, you can source your Tcl script from LeonardoSpectrum as follows:

- Interactive Command Line Shell (**Level 3**)
- GUI Menu Bar File -> Run Script
- Command Line with Path to LeonardoSpectrum

**Note:** The Exemplar history file is a Tcl script file that you can use after making the necessary edits.

Interactive Command Line Shell (**Level 3**)

Type the following syntax to source your Tcl script:
```
source <my_tcl_script>
```

GUI Menu Bar File -> Run Script

On the menu bar click on File -> Run Script. Type in your Tcl script name or click on the button and choose a Tcl script file. Your script file runs in the GUI Information window.

Command Line with Path to LeonardoSpectrum

Bring up your PC or UNIX window. In the LeonardoSpectrum install area, locate where $EXEMPLAR points to the location of the software. Type the appropriate argument to source your Tcl script:

- **UNIX:** $EXEMPLAR/bin/spectrum -file <my_tcl_script>
- **PC DOS:** $EXEMPLAR/bin/win32/spectrum -file <my_tcl_script>
Command Line Description

The LeonardoSpectrum command line functions are described as follows:

- Command and Option Abbreviation
- Aliasing
- Redirecting Output
- Command Line Help
- Tcl Scripting Language
- Customizing the Command Line Interface

Command and Option Abbreviation

LeonardoSpectrum allows abbreviated Tcl commands: you only need to spell out a command until the command meaning is unambiguous.

For example, the command `pre_opt` executes the `pre_optimize` command. If the command is still ambiguous, LeonardoSpectrum produces an error message. For example, the command `pre` displays the following:

```
ambiguous command "pre": pre_optimize present_design preserve_signal
```

The LeonardoSpectrum commands also allow abbreviated options: you do not need to type the options in full; only type the part that makes the option unambiguous.

For example, `pre_optimize -c` enables the `-common_logic` option for the `pre_optimize` command.

Aliasing

LeonardoSpectrum offers an `alias` command, which allows you to define your own name for commonly used command strings. For example, if you frequently use the command `list_design -ports`, then you may want to write an alias:

```
alias lp list_design -ports
```

If you now type the command `lp`, LeonardoSpectrum executes `list_design -ports`. 
Redirecting Output

Almost all commands from the LeonardoSpectrum command list print information to standard output (normally the screen). Standard output can be redirected into a file. For example:

```
optimize -target xi4 > optimize.log
```

All messages from the `optimize` command are then redirected into the file `optimize.log`.

```
optimize -target xi4 >> total.log
```

All messages from the command are appended to the file `total.log`.

Error messages are not redirected to a file when you use this method of redirection. They are displayed on your screen. Therefore, you always know whether or not a command completed successfully.

Command Line Help

You can display information about commands by using the `help` command. The `help` command uses a regular expression (a name with or without wildcards), and prints usage for commands that match the regular expression. For example, you can type `help *` to bring up a transcript list of all commands. Typing `help pre*` displays information about all commands that start with the string `pre`.

Also, every command takes `help` as an option.

```
pre_optimize -help
```

is the same as:

```
help pre_optimize
```

Tcl Scripting Language

LeonardoSpectrum accepts all commands of the Tcl language. Tcl supports commands that include: variable assignment, handling of lists and arrays, sorting, string manipulation, arithmetic operations, (if/case/foreach/while) statements, and procedures. Tcl is VERY handy for writing scripts for LeonardoSpectrum.
The Tcl command, `source <my_tcl_script>`, enables you to source (execute) script files from LeonardoSpectrum, or from within other script files. This feature allows you to write customized (portions of) design flows, or any other sequence of commands that you may want to execute.

A feature inherited from Tcl is autoexec: all UNIX (and many DOS) commands available from your path can be run from the LeonardoSpectrum command line. Another helpful Tcl feature is history tracking. Type the command `history` to view your previous commands. Any previous command can be re-executed using `!NUMBER`, or `!!` for re-execution of the last command.

**Customizing the Command Line Interface**

LeonardoSpectrum loads the following script file when starting up:

```
$EXEMPLAR/data/exemplar.ini
```

You can customize the command line interface by modifying the `exemplar.ini` file. Frequently used aliases and several Tcl procedures, such as `view_schematic` and `push_design`, are defined in this file. You can add your own definitions to this file. If you have an `exemplar.ini` file in your local directory, LeonardoSpectrum loads this file during startup instead of `$EXEMPLAR/data/exemplar.ini`. Source `$EXEMPLAR/data/exemplar.ini` from your local file, rather than copying and then modifying this file so that any changes to this file in the next LeonardoSpectrum release will be picked up. For example, to add an alias for `my_command`, create a file in your local directory called `exemplar.ini` and add the following lines:

```
source $env (EXEMPLAR)/data/exemplar.ini
alias MC my_command
```
Alphabetical Command List

This section contains the alphabetical list of commands.

Note: Commands are also listed in the Utilities chapter.

Syntax Definitions

The command list character symbols are defined as follows:

- [ ] optional arguments
- <> fields to be completed with your names
- | “or” symbol indicates mutually exclusive arguments

In the read command the read <file_name(s)> field is replaced with your file name(s). For example: read fsm.vhd data.vhd top.vhd.

In the read command the optional argument of [-format <format_name>] is entered as -format vhdl.

The optional arguments:
[-parameters <parameters_list>] | [-generics <generics_list>]
are mutually exclusive. Only one argument may be used as indicated by the | “or” symbol.

Note     Always use the forward slash character (/) to separate directory names in a path, even on the PC. LeonardoSpectrum interprets the back slash character (\) as a Tcl escape character.
add_rename_rule

Add a renaming rule to a ruleset.

Syntax

```
add_rename_rule
   <ruleset_name>
```

Arguments

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;ruleset_name&gt;, &lt;type_of_objects&gt;, &lt;word&gt;, &lt;substring&gt;, &lt;characters&gt;,</td>
</tr>
<tr>
<td></td>
<td>&lt;replace_string&gt;, &lt;prepend_string&gt;, &lt;append_string&gt;, &lt;escape_string&gt;</td>
</tr>
<tr>
<td>integer</td>
<td>&lt;truncate_word&gt;, &lt;length_exceeds&gt;</td>
</tr>
</tbody>
</table>

Arguments

- `<ruleset_name>`

Name of the ruleset to which this rule should be added. The ruleset should first be created using the `create_rename_ruleset` command. This is a string type of argument.

Options

- `-type <type_of_objects>`

Type of objects to which this renaming rule applies. Valid values are all (default), port, net, instance, view and cell.

- `-find_word <word>`

Matches word exactly. This option is mutually exclusive with the `-find_substring`, `-find_character`, `-find_first_character` and `-find_last_character` options (only one may be specified at a time).
add_rename_rule

-length_exceeds <integer>
The length of the word exceeds the specifications.

-find_substring <substring>
Matches substring anywhere in a word. This option is mutually exclusive with the
-find_word, find_character, find_first_character and
find_last_character options.

-find_character <characters>
Matches any characters anywhere in a word. This option is mutually exclusive with the
-find_word, find_substring, find_character and find_last_character options. Specify any number of characters, enclosed in double quotes. To specify the double quote or back slash character, escape it using a back slash.

-find_first_character <characters>
Matches any characters at the start of a word. This option is mutually exclusive with the
-find_word, find_substring, find_character and find_last_character options. Specify any number of characters, enclosed in double quotes. To specify the double quote or back slash character, escape it using a back slash.

-find_last_character <characters>
Matches any characters at the end of a word. This option is mutually exclusive with the
-find_word, find_substring, find_character and find_first_character options. Specify any number of characters, enclosed in double quotes. To specify the double quote or back slash character, escape it using a back slash.

-replace <replace_string>
Replaces any match found (using the -find_word, find_substring,
-find_character, find_first_character or find_last_character options) with the specified replace_string. This option is mutually exclusive with the
-prepend_word, append_word, and escape_word options.

-prepend_word <prepend_string>
Prepends prepend_string to any word with a match found (using the -find_word,
-find_substring, find_character, find_first_character or
find_last_character options). This option is mutually exclusive with the -replace,
-prepend_word, and escape_word options.

-append_word <append_string>
add_rename_rule

Appends append_string to any word with a match found (using the -find_word, -find_substring, -find_character, -find_first_character or -find_last_character options). This option is mutually exclusive with the -replace, -prepend_word and -escape_word options.

-truncate_word <integer>
Shorten the word to the specified length.

-escape_word <escape_string>
Escapes any word with a match found (using the -find_word, -find_substring, -find_character, -find_first_character or -find_last_character options) with the two characters specified in escape_string. This option is mutually exclusive with the -replace, -prepend_word. This replaces any match with this string.

Description

The add_rename_rule command adds an individual rule to a ruleset, defined with the create_rename_ruleset command. Rename rules may be used to modify any filenames which LeonardoSpectrum produces into filenames acceptable to your third-party tools. Use the -find* options to identify which (illegal) words, substrings or characters to modify. Use the -replace, -prepend_word, -append_word and -escape_word options to specify what to do with each match found using the -find* options. Multiple rename rules may be added to each ruleset.

Examples

add_rename_rule ALTERA -find_substring "\(*\)" -replace "_"
This example replaces all instances of the open and closed parentheses characters) and (with the underscore character _.

add_rename_rule ALTERA -find_character "\"#*+,-:;=|!$%\^`~\" -replace "_"
This example replaces all instances of the (illegal) characters " # * + - , . /:;=|!$%\^` and ~ with the underscore character _.

add_rename_rule XILINX -find_character "\(\<\|\{\}\)" -replace ""
This examples deletes (replaces with an empty string) all instances of the characters ( ) < > [ ] { and }.

add_rename_rule XILINX -find_first_character _0123456789 -prepend "&"
add_rename_rule

This example prepends any word that begins with a number or underscore with the ampersand (&) character.

Related Commands, Variables

Commands  apply_rename_rules, create_rename_ruleset, remove_rename_ruleset, report_rename_rules

Variables    none
alias

Define alternative command for a (set of) command(s).

Syntax

```plaintext
alias
   [<alias_name> [({script_expansion})]]
```

Arguments

- `<alias_name>`
  Name of an alias to define or to display. If you omit this argument, the `alias` command lists all defined aliases. This is a string type of argument.

- `<script>`
  Tcl script (sequence of commands) that is executed in place of the alias. If spaces occur in `script`, put braces ({{}}) around it, as in any Tcl script. If you specify `alias_name` and omit `script`, the existing definition of the alias is displayed.

Description

The `alias` command defines a new command that executes either a built-in LeonardoSpectrum command or a Tcl script.

When you use an alias, any added arguments are appended to the script (for that execution only; the script itself is not modified).

The `alias` command is very suitable for simple redefinition of commands. When you need scripts with multiple commands, or when arguments cannot be simply appended to an alias script, it is easier to write a Tcl procedure (with the Tcl `proc` command) rather than create an alias.

Examples

```plaintext
alias list_ports (list_design -ports)
    This example defines an alias named `list_ports`, which executes the LeonardoSpectrum command `list_design -ports`.

alias dl list_design
    This example defines an alias named `dl`, which executes the LeonardoSpectrum command `list_design`.
```
alias dl

This example causes the alias command to display the script for the alias dl.

Related Commands, Variables

Commands  help, unalias <alias_name>

Variables  none
analyze

Read an hdl file into an hdl library without any elaboration.

Note: analyze is available only to Level 3.

Syntax

```
analyze
   <file_names>
   [-work <library_name>] [-format <format_name>] [-technology <string>]
```

Arguments

<file_names>

Name of the input file. file_names can be local filenames, relative path names, or absolute path names. This is a list type of argument.

If multiple filenames are specified, use the Tcl list format, eg `{name1 name2}`. If filenames with spaces are specified, wrap the filename with spaces in either quotes or curly braces and wrap the list of one or more filename in curly braces, eg `{“filename with spaces”}` or `{”first filename with spaces”} {second filename with spaces} normal filename`.

Example: analyze `{“my files/file1.vhd” “other files/file2.vhd”}`

Note

Always use the forward slash character (/) to separate directory names in a path, even on the PC. LeonardoSpectrum interprets the back slash character (\) as a Tcl escape character.

Options

- `-format <format_name>`

The format of the input file. The table below shows which file name extensions the analyze command automatically interprets. Valid values are as follows: Verilog, vhdl, xnf, edif, sdf, xdb.
analyze

If you omit this argument, the `analyze` command attempts to determine the file format from the extension in the file name as shown in the following table.

<table>
<thead>
<tr>
<th>File Name Extension</th>
<th>File Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>.v, .verilog</td>
<td>verilog</td>
</tr>
<tr>
<td>.vhd, .vhdl</td>
<td>vhdl</td>
</tr>
<tr>
<td>.xdb</td>
<td>xdb</td>
</tr>
<tr>
<td>.xnf</td>
<td>xnf</td>
</tr>
<tr>
<td>.sdf</td>
<td>sdf</td>
</tr>
</tbody>
</table>

If the `analyze` command cannot determine the file type, it prompts you for the information.

- **-work <library_name>**
  
  Store designs from the input file in the work library `library_name` instead of the library `work`.

- **-technology <technology_name>**
  
  Specify a technology target to guide analysis.

**Description**

The `analyze` command performs syntax and semantic checks on a VHDL or Verilog file and stores the result in an intermediate database (an HDL library). Once the designs are stored in an HDL library, they are accessible for other design data through subsequent `analyze`, `elaborate`, or `read` commands.

HDL library data consists of VHDL entities, architectures, packages, configurations and Verilog modules. The command `list_design -hdl .work` displays the design units stored in the HDL library `work`.

The `analyze` command does not create a present design, as the `read` command does. In addition, `analyze` does not perform synthesis checks on the design.
analyze

Related Commands, Variables

Commands  elaborate, list_design, read
Variables   full_case, parallel_case, viewlogic_vhdl, vhdl_87
**apply_rename_rules**

Change names of objects, using renaming rules.

**Syntax**

```bash
apply_rename_rules
[<design>]
 -ruleset <ruleset> [-single_level] [-test]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td><code>&lt;design&gt;</code>, <code>&lt;ruleset&gt;</code></td>
</tr>
</tbody>
</table>

**Arguments**

`<design>`

Name of the design where names of objects are to be changed using renaming rules.

**Options**

- `-ruleset <ruleset>`
  
  The is the rename ruleset string that should be applied to the design.

- `-single_level`
  
  Apply the renaming rules only at the top level of the design hierarchy.

- `-test`
  
  Do not apply the ruleset renaming string. Instead, produce a report of what will be renamed.

**Related Commands, Variables**

**Commands**

`add_rename_rule`, `create_rename_ruleset`, `remove_rename_ruleset`, `report_rename_rules`

**Variables**

`none`
**balance_loads**

Resolve load violations in a design, across the hierarchy.

**Syntax**

```
balance_loads
   [<design_name>] [-single_level]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;design_name&gt;</td>
</tr>
</tbody>
</table>

**Arguments**

- `<design_name>`

  Name of the design for which the `balance_loads` command performs load balancing. If you omit this argument, the command operates on the present design. This is a string type of argument.

**Options**

- `-single_level`

  Perform load balancing only at the top level of hierarchy. If you omit this option, the `balance_loads` command performs load balancing on the entire design hierarchy.

**Description**

The `balance_loads` command resolves load violations throughout the design. Most of the LeonardoSpectrum commands work on individual levels of hierarchy in the design at one time. With this method, load violations on internal nets (not connected to ports) can be resolved on the fly. However, in a hierarchical design, load violations can still be created at the boundaries of the levels of hierarchy.

To assure that gates do not drive more load than they are allowed to anywhere in the design, LeonardoSpectrum traverses the entire hierarchy of the design starting at the bottom. It analyzes all nets for load violations, and it uses buffer (tree) insertion to resolve these violations. The `balance_loads` command is sensitive to the following design constraints (attributes):

- On inputs/inouts: `OUTPUT_LOAD` and `OUTPUT_FANOUT`
- On outputs/inouts: `MAX_LOAD` and `INPUT_MAX_FANOUT`
**balance_loads**

**Examples**

```
balance_loads
```
Resolve load violations on hierarchy boundaries anywhere in the present design.

```
balance_loads .work.second.contents
```
Resolve load violations on hierarchy boundaries anywhere in the view contents of the cell second in the library work.

**Related Commands, Variables**

**Commands**

none

**Variables**

max_cap_load, max_fanout_load, max_transition

**Known Bugs, Limitations**

This command is only effective after the design is mapped to a particular technology with the optimize command.

**Note:** Refer to optimize_drc_resolving variable in the Variables chapter.
connect

Connects net with port or port instance.

Syntax

```plaintext
connect
   -port <port_name> -net <net_name>
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td><code>&lt;port_name&gt;</code>, <code>&lt;net_name&gt;</code></td>
</tr>
</tbody>
</table>

Arguments

- `-port <port_name>`

   Formal name of an existing port to connect to the net `net_name`. Only a single port or port-instance is accepted. This is a string type of argument.

- `-net <net_name>`

   Formal name of an existing net to which you are connecting a port. Only a single net is accepted. This is a string type of argument.

Description

The `connect` command attaches a net to a port or port-instance. You can use relative or absolute port and net names. The net and the port or port-instance must be contained in the same view.

If the specified port or port-instance is already connected to a different net, it will be disconnected first, then connected to the new net.

Note

This command operates at a very low level of the design and changes the structure of the design. Simulation differences between pre- and post-synthesized versions of a design and can occur as a result of using this command.
connect

Examples

connect -port clk -net clk

This example connects the port clk in the present design to the net clk in the present design. This example is valid only if present design is a view.

connect -port inst73.reset -net rst

This example connects the port-instance reset of instance inst73 in the present design to net rst in the present design. This example is valid only if present design is a view.

connect -port .work.top.contents.Out -net .work.top.contents.outnet

This example connects the port Out in the view contents in the cell top in the library work to the net outnet in the same view.

Related Commands, Variables

Commands  disconnect, list_connection

Variables  none

Known Bugs, Limitations

There is no way to connect a net to multiple ports or port-instances with a single connect command.
connect_path

Connect a timing path on an instance through ports.

Syntax

```
connect_path
    <port_names>
    [-instance <instance_name>] | [-gate <gate_name>] | [-all]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;port_names&gt;, &lt;instance_name&gt;, &lt;gate_name&gt;</td>
</tr>
</tbody>
</table>

Arguments

- `<port_names>`

  Connect a timing path on an instance through ports.

Options

- `-instance <instance_name>`

  Connect a path on instance_name.

- `-gate <gate_name>`

  Connect a path on the instances of gate_name.

- `-all`

  Connect paths on all instances.

Description

This command marks a timing arc in an instance of a gate as connected or propagating. This command can be used to reconnect paths marked as false paths, or to reconnect paths through which timing analysis is desired.
Examples

# connects set to q of DFF instance, DFF_inst1
connect_path -instance DFF_inst1 SET Q

When gate is specified, all paths for all instances of this gate are disconnected/connected. If all SET to Q paths need to be connected/disconnected, for all gates that have these pins, \(-all\) option can be used.
copy

Copy a single view or the full hierarchy under it and append the suffix to all the copied view names.

Syntax

```
  copy
  <object1> <object2> | <object container> [-hier <suffix_name>]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;object1&gt;, &lt;object2&gt;, &lt;object_container&gt;, &lt;suffix_name&gt;</td>
</tr>
</tbody>
</table>

Arguments

- `<object1>`, `<object2>`
  
  Names of source and target design objects. `<object1>` must exist prior to the copy operation, but `<object2>` must not. This is a string type of argument.

- `<object_container>`
  
  Name of an object in a design that is designed to organize and hold other design objects. Examples are libraries and cells. Both source and target `<object_container>`s must exist prior to the copy operation. This is a string type of argument.

Options

- `-hier <suffix_name>`
  
  This option is only applicable to views. The entire hierarchy under view is copied. The suffix is appended to every copied view.

Description

The `copy` command duplicates the contents of an existing design object.

Examples

There are two examples for the `copy` command: Single View Copy and Full Hierarchy Under View Copy.
**copy**

*Single View Copy*

```plaintext
copy .work.design.contents .work.design.save
```

This example copies the view contents contained in the cell design within the library work to
the view save in the same cell.

*Full Hierarchy Under View Copy*

```plaintext
copy -hier _copy .work.address_decoder.test
```

This example copy command copies the entire hierarchy under view and appends the suffix
“_copy” to every view that is copied as follows:

```plaintext
.work.address_decoder.test_copy
```

**Known Bugs, Limitations**

The `copy` command does not copy ports, nets, or instances. Use the `create` command for that purpose.
create

Creates an object.

Syntax

create
<object_name>

[-port]|[-net]|[-instance][-direction <port_direction>] [-of
<brview_name>]

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;port_direction&gt;, &lt;view_name&gt;</td>
</tr>
</tbody>
</table>

Arguments

<object_name>

Name of a design object, such as a library, cell, view, net, port, or instance for the create command to create. This is a string type of argument.

Options

-port, -net, -instance

Type of the created object, if it is within a view.

-direction <port_direction>

For port objects, the port type. Valid values for port_direction are IN, OUT, and INOUT.

-of <view_name>

Defines the name of the view to instantiate. This argument is required for instance objects only.

Description

The create command makes a new object in the design. If the object already exists, the create command issues an error message.

Examples

create -port X -direction IN
This example creates an input port named \texttt{X} in the present design. This example is valid only if the present design is a view.

\texttt{create .save\_work}

This example creates a new library called \texttt{save\_work}.

\texttt{create -instance vdd -of .PRIMITIVES\_TRUE\_INTERFACE}

This example creates an instance of the view of the primitive cell \texttt{TRUE} (generic power symbol) named \texttt{vdd} within in the present design.

**Related Commands, Variables**

- **Commands**: remove, move, copy, connect, disconnect
- **Variables**: none
create_rename_ruleset

Create a ruleset for object renaming.

Syntax

create_rename_ruleset
    <ruleset_name>
    [-no_collision_between <object_list>] [-case_insensitive]

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;ruleset_name&gt;, &lt;object_list&gt;</td>
</tr>
<tr>
<td>list</td>
<td>&lt;object_list&gt;</td>
</tr>
</tbody>
</table>

Arguments

<ruleset_name>

Create a ruleset for object renaming.

Options

-no_collision_between <object_list>

Valid values in object_list are port, net, or instance. Renaming rule is created to avoid name collisions between objects of these types.

-case_insensitive

Disregard case when checking name collisions.

Related Commands, Variables

Commands

apply_rename_rules, add_rename_rule, remove_rename_ruleset, report_rename_rules

Variables

none
create_wrapper

Creates a wrapper for a design.

**Syntax**

```
create_wrapper
    [<root_entity/module_name>] 
    [-architecture <architecture_name>] 
    [-work <library_name>] 
    [-file <file_name>] 
    [-wrap_name <wrapper_name>] 
    [-parameters <parameters_list>] 
    [-generics <generics_list>]
```

**Arguments**

- `<root_entity/module_name>`
  
  Name of the VHDL entity or configuration or the Verilog module for which you are creating a simulation wrapper. The design unit must exist in the `work` HDL library or in the library indicated with the `-work` option. If you omit this argument, the design unit that was most recently analyzed (with the `analyze` or `read` command) is used to create a simulation wrapper.

**Options**

- `-architecture <architecture_name>`
  
  Name of the VHDL architecture for which you are creating a simulation wrapper. The architecture must exist under the root design unit. If you omit this argument, the `create_wrapper` command elaborates the most recently analyzed architecture of `<root_entity/module_name>`.

- `-work <library_name>`
  
  Defines the HDL library where the `<root_entity/module_name>` resides. If you omit this argument, the `create_wrapper` command uses the HDL library named `work`.

- `-file <file_name>`
  
  Defines the name of the file in which to write the simulation wrapper.

- `-generics <generics_list>`
**create_wrapper**

Specify the setting of a generic in the top-level VHDL entity that will be elaborated. Note that case-insensitive identifiers in VHDL are changed to lower case before elaboration. In that case, define the name of the generic in lower case. This option also sets the value of a parameter of the top-level Verilog module to be elaborated.

The `create_wrapper` command parses the value of each generic and compares the type of the value with the type of the generic.

The syntax of `generics_list` is `{generic=value generic=value...}

The following example is valid for a design unit with an integer generic `size`, an array generic `seed`, and an enumeration-type (boolean) generic `use_this`:

```
-generics { size=9   seed="00100011"   use_this=TRUE }
```

Equivalent to `-generics <generics_list.>`

**Description**

Synthesizing with LeonardoSpectrum expands complex types on ports (like arrays, integers, records, enumeration types) to individual bits. This creates a mismatch between the pre-synthesis and post-synthesis port list for VHDL and Verilog designs. To still be able to simulate a post-synthesis netlist (in Verilog or VHDL) with a pre-synthesis Verilog or VHDL testbench, you can create a simulation wrapper. The wrapper model has the original complex types on the port boundaries, and it instantiates a post-synthesis model (with bits at the port boundaries).

The `create_wrapper` command creates the wrapper from the HDL library models. Therefore, the `create_wrapper` command takes almost the same arguments as the `elaborate` command. To create a wrapper model, you just need to `analyze` the top-level VHDL or Verilog model.

For VHDL source code, a VHDL wrapper will be created. The wrapper model will consist of an architecture that can be linked to the original entity of the design. For Verilog source code, a new Verilog module will be created.

For VHDL, the `create_wrapper` command uses type transformation functions from the `typetran` package (`$EXEMPLAR/data/typetran.vhd`) to translate the complex types to the bits of the synthesized VHDL model. For Verilog, the wrapper does not require additional source code.

**Related Commands, Variables**

**Commands**  
analyze, elaborate, write

**Variables**  
none
The `decompose_luts` command decomposes lookup tables (LUTs) within a view into structural netlists expressed in terms of AND and OR gates. LUTs are created by the `optimize` command when you target Xilinx, ORCA, or Altera FLEX technologies.

Use `decompose_luts`: (1) to write out netlists in formats, such as EDIF, that do not support behavioral descriptions, such as LUTs; and (2) before unmapping designs containing lookup tables. **Note:** Refer to the `unmap` command and to `auto_write` in the Utilities chapter.

You cannot re-assemble structural netlists back into LUTs. Therefore, you should use the `decompose_luts` command just prior to writing the design to a file.
decompose_luts

Related Commands, Variables

Commands none
Variables lut_cell_name

Known Bugs, Limitations

Area reporting may not include LUTs after they have been decomposed.
**disconnect**

Removes connection between net and port or port instance.

**Syntax**

```
disconnect
  -port <port_name> -net <net_name>
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;port_name&gt;, &lt;net_name&gt;</td>
</tr>
</tbody>
</table>

**Arguments**

- **-port <port_name>**
  
  Name of an existing port to disconnect from the net net_name. Only a single port or port-instance is accepted.

- **-net <net_name>**
  
  Name of an existing net from which you are disconnecting a port. Only a single net is accepted.

**Description**

The `disconnect` command detaches a net from a port or port-instance. You can use relative or absolute port and net names. The net and the port or port-instance should be contained in the same view.

**Note**

This command operates at a very low level of the design and changes the structure of the design. Simulation differences between pre- and post-synthesized versions of a design can occur as a result of using this command.

**Examples**

```
disconnect -net clk -port clk
```

This example disconnects the port clk from the net clk, both of which are in the present design. This example is valid only if the present design is a view.
**disconnect**

```
disconnect -net rst -port .work.top.contents.reset
```

This example disconnects the net `rst` in the present design from the port `reset` in the view `contents` of cell `top` within the library `work`. This example is valid only if the present design is set to the view containing the port `reset`.

```
disconnect -port modgen_1.d(0) -net data(0)
```

This example disconnects the port-instance `d(0)` of instance `modgen_1` in the present design from net `data(0)` in the present design. This example is valid only if the present design is a view.

**Related Commands, Variables**

**Commands**  
connect, list_connection

**Variables**  
none

**Known Bugs, Limitations**

There is no way to disconnect a net from multiple ports or port-instances with a single `disconnect` command.
**disconnect_path**

Disconnect a timing path in an instance through ports.

**Syntax**

```plaintext
disconnect_path
  <port_names>
  [-instance <instance_name>] | [-gate <gate_name>] | [-all]
```

**Arguments**

- `<port_names>`
  Disconnect a timing path on an instance through ports.

**Options**

- `-instance <instance_name>`
  Connect a timing path on instance_name.
- `-gate <gate_name>`
  Connect a timing path on the instances of `gate_name`.
- `-all`
  Connect paths on all instances.

**Description**

This command marks a timing arc in an instance of a gate as disconnected. This command can be used to block false paths. These false paths are asynchronous paths or paths through which timing analysis is not desired.
disconnect_path

Examples

# disconnects set to q of DFF instance, DFF_inst1
disconnect_path -instance DFF_inst1 SET Q

When gate is specified, all paths for all instances of this gate are disconnected/connected. If all SET to Q paths need to be connected/disconnected, for all gates that have these pins, -all option can be used.
elaborate

Elaborates analyzed hdl design units creating a design(s).

Note: elaborate is only available to Level 3.

Syntax

elaborate

[<root_entity/module_name>]
[-architecture <string>] [-single_level] [-technology <string>]
[-work <string>] [-parameters <list>] |- [generic list]]

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;architecture_name&gt;, &lt;library_name&gt;, &lt;technology&gt;, &lt;root_entity/module_name&gt;</td>
</tr>
<tr>
<td>list</td>
<td>&lt;parameter_list&gt;, &lt;generics_list&gt;</td>
</tr>
</tbody>
</table>

Arguments

<root_entity/module_name>

Name of the VHDL entity or configuration or the Verilog module to be elaborated. The design unit must exist in the work HDL library (or in the library indicated with the -work argument). If you omit this argument, the elaborate command elaborates the most recently analyzed design unit.

Options

-architecture <architecture_name>

Name of the VHDL architecture to be elaborated. The architecture must exist under the root design unit. If you omit this argument, the elaborate command elaborates the most recently analyzed architecture of root_entity/module_name>.

-single_level

Only elaborate the top-level of the design. Only the specified architecture will be elaborated. All lower level hierarchy is either from the hierarchical data base, or a black box is created for the lower level hierarchy. This option is for bottom up design methodology.
elaborate

-work <library_name>

Defines the HDL library where <root_entity/module_name> resides and the design library where the elaborate command stores the elaborated design. If you omit this argument, the elaborate command uses the HDL and design libraries named work.

Note To make the analyze and elaborate commands operate on the same module, the library named in both commands must be identical.

-generics <generics_list>

Specify the setting of a generic in the top-level VHDL entity that will be elaborated. Note that case-insensitive identifiers in VHDL are changed to lower case before elaboration. In that case, define the name of the generic in lower case. This option also sets the value of a parameter of the top-level Verilog module to be elaborated.

The elaborate command parses the value of each generic and compares the type of the value with the type of the generic.

The syntax of generics_list is {generic=value generic=value...}

The following example is valid for a design unit with an integer generic size, an array generic seed, and an enumeration-type (boolean) generic use_this:

   -generics { size=9   seed="00100011" use_this=TRUE }

-equivalent to -generics <generics_list.>

-technology <technology_name>

The -technology should be the same as the target technology to which the design will be mapped. The technology library does not need to be loaded to use this elaborate command option.

Description

The elaborate command creates a present design from VHDL entities and architectures or Verilog modules stored in an HDL library. The elaborate command synthesizes the design to technology-independent primitives and operators (using design elements from the built-in PRIMITIVES and OPERATORS libraries). During elaboration, Leonardo maintains design
elaborate

Hierarchies expressed in VHDL entity/architecture pairs and Verilog components. The elaborate command performs RTL synthesis-style checks and issues errors and warnings in cases that could produce discrepancies between synthesis and simulation.

If a design with the same name as the design being elaborated already exists in the database, the elaborate command replaces the existing design with the new version and issues a warning message. If there were already references to the existing design from other designs in the database, the elaborate command links the port references to the new design automatically. If ports in the existing design are missing in the new design, the tool disconnects the references to those ports and issues a warning.

Array, record and integer types in VHDL and array types in Verilog will be expanded to bits, according to the hdl_*_name_style Tcl variables. Enumeration types (in VHDL) will be encoded to bits, according to the values of the Tcl encoding variable.

Case-insensitive identifier names in VHDL are changed to lower case to ensure that they are interpreted correctly in the rest of the system, which is case-sensitive. Case-sensitive identifiers in VHDL and all identifiers in Verilog remain unchanged.

Related Commands, Variables

Commands  analyze, read

Variables  encoding, hdl_array_name_style, hdl_integer_name_style, hdl_record_name_style
**generate_timespec**

Create TIMESPEC information from user constraints; Xilinx only.

**Syntax**

```plaintext
generate_timespec
     [<design_name>]
     [-single_level]
```

**Arguments**

<table>
<thead>
<tr>
<th>Type</th>
<th>.Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;design_name&gt;</td>
</tr>
</tbody>
</table>

**Options**

-`-single_level`

Generate timespec data only at the top level of hierarchy. If you omit this option, the `generate_timespec` command traverses the entire design hierarchy.

**Description**

The `generate_timespec` command produces timespec data for designs that use Xilinx LCA-based technologies. The command takes timing constraints that you supply and generates timespec data. This data will appear in the output `xnf` file. Use this command after the `optimize` command.

**Example**

A design has a single clock, some registered paths and some purely combinational paths. This design would have two input TIMEGRPs (PADS and DFF outputs) and two output TIMEGRPs (PADS and DFF inputs). Assuming that all four types of paths exist, this design would have four TIMESPECs written out.
generate_timespec

Related Commands, Variables

- Commands: optimize, write
- Variables: none

Known Bugs, Limitations

This command applies only to Xilinx LCA based architectures.
group

Group a list of instances into one instance of a new view.

Note: group is only available to Level 3.

Syntax

```
group
 <list_of_instances>
 [-cell_name <cell_name>] [-view_name <view_name>] [-inst_name <instance_name>]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;cell_name&gt;, &lt;view_name&gt;, &lt;instance_name&gt;</td>
</tr>
<tr>
<td>list</td>
<td>&lt;list_of_instances&gt;</td>
</tr>
</tbody>
</table>

Arguments

`<list_of_instances>`

Names of design instances that the `group` command uses to form a single instance of a new view. All instance names must be from the same view.

Options

- `-cell_name <cell_name>`

   Name of a new cell to contain the new view. If you omit this option, the `group` command automatically generates a name (using the format `xmplr_cell_number`) for the new cell. The cell name must be a simple name.

- `-view_name <view_name>`

   Name of a new view to contain `list_of_instances`. If you omit this option, the `group` command automatically generates a name (using the format `xmplr_view_number`) for the new view. The view name must be a simple name.

- `-inst_name <instance_name>`
group

Name of the new instance formed from the instances indicated by the value of list_of_instances. If you omit this option, the group command automatically generates a name (using the format xmplr_inst_number) for the new instance. The instance name must be a simple name, not a formalized name.

Description

The group command moves a list of instances, with their connected nets, from one view to a new view in a new cell, thus creating a new level of hierarchy. With this command, you can define the name of the instance, the view, and the cell where the new view resides.

The group command is useful to cluster logic that should be optimized as a single view. For example, if two instances of two views share much of the same logic or interconnect, it makes sense to group them into a new level of hierarchy (and to ungroup the hierarchy inside the new group). This way, subsequent optimization operations can minimize the logic shared between the two original views, resulting in smaller or faster designs.

Related Commands, Variables

Commands  ungroup, unfold
Variables   none
help

Help gives help on commands, aliases, and variables.

Syntax

help

  [<search_string>]
  [-variables]

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;search_string&gt;</td>
</tr>
</tbody>
</table>

Arguments

<search_string>

Regular expression used to search for a full-text informational message about commands or variables that match the search_string pattern. If you omit this argument, the help command displays a one-line description of all commands or variables.

Options

-variables

Displays a one-line description and the current settings for Tcl variables that affect LeonardoSpectrum. If you omit this option, the help command displays help information for commands and aliases instead.

Description

The help command provides descriptions and usages of LeonardoSpectrum commands or Tcl variables, and current settings of those variables.

Examples

help optimize

This example produces a usage message for the optimize command.

help *modgen

This example produces a usage message for all LeonardoSpectrum commands that end with the characters modgen, such as resolve_modgen and load_modgen.
help

help -var *xnf*

This example produces a listing of the current settings for and descriptions of all built-in variables containing the characters xnf in their name, such as write_clb_packing and write_lut_binding.

Related Commands, Variables

Commands alias
Variables none

Known Bugs, Limitations

The help command does not provide usage information for true Tcl commands or Tcl scripts. Tcl commands are documented in *Tcl and the Tk Toolkit*, by John Ousterhout, published by Addison-Wesley, ISBN 0-201-63337-X.
list_attributes

Lists attributes on any object.

Syntax

list_attributes
   [<list_of_objects>]
   [-port]|[-net]|[-instance]

Arguments

<list_of_objects>

Names of attributes applied to any objects (library, cell, view, port, net, instance) you want listed. Object names are case-sensitive, and you can use wildcards. If you omit this argument, the list_attributes command lists the attributes of the present design.

Options

-port, -net, -instance

Indicator that the object name(s) refer to ports, nets, or instances, respectively. If you omit this argument, the list_attributes command assumes that the objects in object_list are instances, unless object_list explicitly refers to libraries, cells or views.

Description

The list_attributes command returns a Tcl list of case-insensitive {name value} pairs for the attributes of indicated objects.

This command is intended to be used in Tcl scripts. You can, however, assign the result to a Tcl variable. You can then use the variable in any other Tcl command, for example, a foreach loop.

Examples

list_attributes

This example lists the attributes of the present design and their values.
list_attributes

list_attributes .work

This example lists the attributes and their values of the library called work.

list_attributes .work.topINTERFACE

This example lists the attributes and their values of the view INTERFACE of the cell top in the library work.

list_attributes -port inport(1)

This example lists the attributes and their values of the port inport(1) in the present design. This command is valid only if the present design points to a view.

list_attributes -inst u*

This example lists the attributes and their values for all instances whose names starts with a u.

**Related Commands, Variables**

<table>
<thead>
<tr>
<th>Commands</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>list_design</td>
<td>none</td>
</tr>
</tbody>
</table>

**Known Bugs, Limitations**

You cannot use file I/O redirection with this command, because it returns a Tcl list and no standard output. There is no command that returns the value of a single attribute on an object.
**list_connection**

Lists connected objects to the specified object(s).

**Syntax**

```
list_connection
   <list_of_objects>
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>&lt;list_of_objects&gt;</td>
</tr>
</tbody>
</table>

**Arguments**

`<list_of_objects>`

Names of the objects for which the list_connection command lists network connections. Object names are case-sensitive, and wildcards are accepted.

**Options**

- `-port`, `-net`, `-instance`

  Indicator that the object name refers to ports, nets, or instances. If you omit this argument, the list_connection command assumes that the objects in `list_of_objects` are nets.

- `-hierarchical`

  List all objects connected hierarchically to the ones in `<list_of_objects>`. The list_connection command lists objects at each level of hierarchy below the objects in `<list_of_objects>`. If you omit this argument, only the connections to objects in one view are listed.

- `-direction <net_direction>`

  This is for nets only: direction DRIVER DRIVEN.
list_connection

Description

The `list_connection` command returns a Tcl list of formal names of connections for the indicated objects. If the object list denotes a port or port-instance, the net connected to the port or port-instance is returned. If the object list denotes a net, a list of ports and port-instances connected to the net is returned. If the object list denotes an instance, a list of all port-instances associated with the instance is returned.

In a netlist, ports and port-instances can be connected to a net, and nets can be connected to multiple ports and multiple port-instances.

The `list_connection` command enables you to browse through the netlist, finding netlist connections step by step. The `-hierarchical` argument extends the returned list of all connections from the indicated objects downward through the hierarchy within the same view.

Examples

`list_connection -port clk`

This example returns the name of the net to which the port `clk` is connected in the present design. This example is valid only when the present design is a view.

`list_connection -net Net15`

This example returns the list of the ports and port-instances to which the net `Net15` is connected in the present design. This example is valid only when the present design is a view.

`list_connection -port i145.out`

This example returns the name of the net to which the port-instance `i145.out` is connected, that is, the port `out` on the view to which the instance `i145` is pointing.

`list_connection -instance i*`

This example returns the list of the port-instances for all the instances whose names start with an `i` in the present design.

`list_connection -port clk -hier`

This example returns the list of nets to which the port `clk` is connected, all the way down through the hierarchy.
list_connection

Related Commands, Variables

- Commands: connect, disconnect, list_design
- Variables: none

Known Bugs, Limitations

You cannot use file I/O redirection with this command, because it returns a Tcl list and no standard output.
list_design

Returns list of objects in a design (libraries in the root, cells in a library, views in a cell, etc).

**Syntax**

```
list_design
    [<list_of_designs>]
    [-ports][-nets][-clocks][-instances][-references] [-direction <port_direction>] [-hdl] [-short]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;list_of_designs&gt;</td>
</tr>
</tbody>
</table>

**Arguments**

**<list_of_designs>**

Name of the library, cell, or view for which you want to retrieve a listing of the contents. You can use absolute or relative object names, and wildcards are accepted. Object names are case-sensitive. If you omit this argument, the list_design command returns a list of the contents of the design objects in the present design.

If <list_of_designs> indicates a view, and you omit other arguments, the list_design command uses the instances argument.

**Options**

- **-hdl**
  List the design units stored in HDL libraries. If you omit this argument, the list_design command lists objects in the design.

- **-ports**
  List all the ports of <list_of_designs>. This argument is valid only if <list_of_designs> is one or more views.

- **-nets**
  List all the nets of <list_of_designs>. This argument is valid only if <list_of_designs> is one or more views.

- **-instances** (default)

LeonardoSpectrum Command Reference 1-49
list_design

List all the instances in <list_of_designs>. This argument is valid only if <list_of_designs> is one or more views.

If <list_of_designs> indicates a view, and you omit other arguments, the list_design command uses the instances argument.

-references

List all the instances pointing to <list_of_designs>. This argument is valid only if <list_of_designs> is one or more views.

-short

Display only short path names, not full path names, of design objects.

-direction

Valid only with -ports option. The -direction option takes the port direction (IN OUT or INOUT) and prints only the ports of given direction. If this option is not provided with -port, all the ports will be printed.

Description

The list_design command returns a Tcl list of the contents of one or more designs indicated by the <list_of_designs> argument. The returned Tcl list contains formalized names. If the <list_of_designs> is relative, the list returns relative names. If the <list_of_designs> is absolute, the list returns absolute names.

Although an instance does not contain any objects itself, it does point to a view. For that reason, the list_design command returns the formal name of the view if <list_of_designs> itself indicates an instance.

The command does not operate recursively through a design hierarchy. For example, executing list_design where <list_of_designs> is a library returns only the names of cells in library but not the names of views within the cells. Use the report_area command to generate a report.
**Examples**

- `list_design .work` list all cells in the library named `work`
- `list_design -hdl .work` list all entities, packages and modules, analyzed into the HDL library 'work'
- `list_design -port .work.and2.contents` list all ports on the view contents of the cell `and2` in the library `work`
- `list_design .` list all libraries in the data base
- `list_design` list the contents of the present design. If the present design is a view, list the instances.
- `list_design -ports` list the ports of the present design. Valid only if the present design is a view.
- `list_design -ref` list the references (instances of) the present design; valid only if the present design is a view.
- `list_design .*` list all cells in all libraries
- `list_design -net` list all the nets in the present design; valid only if present design is a view
- `list_design xyz` If the present design is a view, then return the view name to which the instance `xyz` is pointing.
- `list_design -inst x` list views contained in the instance `x` in the present design; valid only if present design is a view
- `list_design -nets n*` list the nets in the present design that start with letter `n`; valid only if present design is a view

**Related Commands, Variables**

- **Commands**  list_connection, report_area, list_attributes
- **Variables**  list_design_object_separator

**Known Bugs, Limitations**

You cannot use file I/O redirection with this command, because it returns a Tcl list and no standard output.
**list_technologies**

Returns list of technology libraries used in a design.

**Syntax**

```
list_technologies
    [<list_technology>]
    [-single_level]
```

List technology libraries in this level only.
load_library

Load a technology library.

Syntax

load_library

    [[-s][-t] <library_name>]

Arguments

<library_name>

Name of a technology library file. The library must have been previously compiled with Exemplar’s lGen tool.

technology_library_name may be a simple name, a relative path name, or an absolute path name. For simple names, such as xi4, the load_library command searches the directory $EXEMPLAR/lib for the file, xi4.syn in this example. If two or more vendor libraries have same name, you must use the full pathname for each library.

Description

This load_library command reads a compiled technology library file, then creates a library in the LeonardoSpectrum design database. The name of the created library is derived from technology_library_name. If technology_library_name is not a simple name, the load_library command uses just the base of the name to derive a new name.

Examples

The following command loads the file $EXEMPLAR/lib/max7.syn and creates the library man7 in the design database:

    load_library max7
load_library

Related Commands, Variables

Commands optimize

Variables exclude_gates, include_gates, load_library_file_extension, process, temp, voltage
load_modgen

Read a module generator description from file.

Syntax

load_modgen

<modgen_library_name>

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;modgen_library_name&gt;</td>
</tr>
</tbody>
</table>

Arguments

<modgen_library_name>

Name of a module generator library file. The search path for a module generator library file is the present working directory, then the $EXEMPLAR/data/modgen directory. The load_modgen command can also read user-defined module generator library files coded in VHDL when such modules follow the coding guidelines explained in the HDL Synthesis Guide.

Description

The load_modgen command loads a module generator library description into the LeonardoSpectrum HDL database. The command resolve_modgen uses the descriptions from this generic library to resolve instances of arithmetic and relational operators created in the design during file read operations or inferred when extracting counters and RAMs with the pre_optimize command. There is only one generic module generator library, named OPERATORS, assigned in LeonardoSpectrum. Successive loading of module generator library files overwrite existing ones, operator by operator. Note: To force old behavior use: '_orig_load_modgen

Examples

load_modgen xi4

Load the module generator library that Exemplar provides for the Xilinx 4000 family of devices.

load_modgen my_modgen.vhd

Load a user-defined module generator library coded in VHDL.
load_modgen

Related Commands, Variables

Commands  optimize, pre_optimize, read, resolve_modgen
Variables  none
Move object(s). Moves a single view or the full hierarchy under it and appends the suffix to all the moved view names.

Syntax

```
move
<object1> <object2> | <object1>..<object_n> <object_container>
[-port] [-net] [-instance] [-hier <suffix_name>]
```

Arguments

```
<string> <object1>, <object2>, <object_container> <suffix_name>
```

Options

```
-port, -net, -instance
```

Indicates that the source object is a port, a net or an instance, respectively.

```
-hier <suffix_name>
```

This option is only applicable to views. The entire hierarchy under view is moved. The suffix is appended to every moved view.

Description

The `move` command relocates objects in the design database. In many cases, the `move` command effectively renames an object. This is similar to the UNIX `mv` command.

Examples

There are two examples for the `move` command: Single View move and Full Hierarchy Under View Move.
**move**

**Single View Move**

move .work.design.contents .work.design.save

In this example, the `move` command moves the view contents in cell `design` of the library `work` to the view `save` in the same cell. The result is essentially the same as changing the name of the view.

move .work.top .save_work

In this example, the `move` command moves the cell `top` in the library `work` to the library `save_work`.

move -port clk CLK

In this example, the `move` command moves the port `clk` to a new port `CLK` in the present design. The result is the same as changing the port name. This example is valid only if the present design is a view.

**Full Hierarchy Under View Move**

move -hier _move .work.address_decoder.test

This example move command moves the entire hierarchy under view and appends the suffix "_move" to every view that is moved as follows:

.work.address_decoder.test_move
move

move -port clk CLK

In this example, the move command moves the port clk to a new port CLK in the present design. The result is essentially the same as changing the port name. This example is valid only if the present design is a view.

Related Commands, Variables

Commands: copy, create, remove
Variables: none

Known Bugs, Limitations

You cannot use this command to move ports, nets, or instances across the boundary of a view.
optimize

Optimize and map a design to a target technology.

Syntax

```
optimize

[<design_name>]
[-target <technology_name>] [-io_target <technology_name>]
[-single_level] [-effort <effort_type>] [-chip] [-macro]
[-area] [-delay] [-pass <pass_nums>] [-nopass
<pass_nums>] [-hierarchy <auto|preserve|flatten>]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;design_name&gt;, &lt;technology_name&gt;, &lt;effort_type&gt;, &lt;auto</td>
</tr>
<tr>
<td>list</td>
<td>&lt;pass_nums&gt;</td>
</tr>
</tbody>
</table>

Arguments

<design_name>

Name of the design where the optimize command performs logic optimization. If you omit this argument, the command operates on the present design.

Options

-target <technology_name>

Name of the technology library to use when the optimize command optimizes the design. The technology library named in this argument must be previously loaded by the load_library command.

-single_level

Optimize only the top-level view of the design. If you omit this option, the optimize command optimizes all views in the design hierarchy in a bottom-up manner.
-effort <effort_type> (default)

Level of effort the optimize command should use during the optimization process. Valid values are as follows:

remap
Performs some local optimization and technology mapping to the target technology.

quick
Performs one pass of optimization and technology mapping. This is a fast optimization effort that would usually be used at the beginning of the design cycle.

standard (default)
Performs multiple passes of optimization and technology mapping. Applies different strategies of optimization, and usually achieves better results than the quick effort, but takes more time.

-chip (default)| macro
The -chip option tells the optimize command to add I/O buffers to top-level view in the design. The -macro option prevents the optimize command from adding I/O buffers.

-area (default)| delay
The -area option tells the optimize command to minimize area in the optimized design. The -delay option instructs the optimize command to minimize delay in the optimized design.

-pass <pass_nums>| -nopass <pass_nums>
The -pass option tells the optimize command to perform only the indicated optimization passes. The -nopass option instructs the optimize command not to perform the indicated optimization passes. LeonardoSpectrum performs a maximum of 4 passes.

You obtain information on optimization passes from prior optimization operations. During each optimization of a design, the optimize command displays the results of each optimization pass, together with the ID number of the pass. The Optimize Report Example shows the results of the command optimize -target xi4 on a design with one level of hierarchy.

-hierarchy <auto|preserve|flatten>
The hierarchy manipulation option is available from the interactive command line shell and batch mode (DOS or UNIX environments).

- auto (auto_dissolve) is selected by default. Logic is equated to 2-input NAND gates, and hierarchy is dissolved according to the following rules:

(1) FPGA/CPLD auto_dissolve limit is 3000 gates (default).
optimize

(2) ASIC auto_dissolve limit is 30 gates (default).

(3) There is a maximum system limit of gates that can be dissolved in a module. This system limit cannot be modified by a user switch. If this limit is exceeded then auto_dissolve is not completed. The auto_dissolve dissolves instances in a context sensitive manner. If a module is instantiated more than once, then the instance is dissolved only if total number of gates does not exceed the system limit.

• If preserve is selected, then hierarchy is not dissolved during optimization.
• If flatten is selected, then the entire design hierarchy is flattened (dissolved).

In addition, auto_dissolve is now the default for the optimization command in both the interactive command line shell and in batch mode. Hierarchy can be preserved with the "-hierarchy_preserve" option in batch mode; and with -hierarchy <preserve> option on the interactive command line shell.

Examples for: -hierarchy <auto|preserve|flatten>:

optimize -ta xi4
optimize -ta xi4 -hierarchy auto

Note: Typing the -hierarchy auto is an option, since auto is the default.

Note FPGA/CPLD: You can use a variable to change the number of gates to be dissolved as follows:
set auto_dissolve_limit (600)
optimize -ta xi4 -hierarchy auto

Note ASIC: You can use a variable to change the number of gates to be dissolved as follows:
set asic_auto_dissolve_limit (600)
optimize -ta xi4 -hierarchy auto

Note: You can preserve the entire design hierarchy with:
optimize -ta xi4 -hierarchy preserve

Note: You can flatten (dissolve) the entire design hierarchy with:
optimize -ta xi4 -hierarchy flatten

Note: Refer to Variable Chapter and Attribute Chapter for more auto_dissolve information.
Example Optimize Report:

Start optimization for design .work.glob_dma INTERFACE

<table>
<thead>
<tr>
<th>Pass</th>
<th>Area</th>
<th>Delay</th>
<th>DFFs</th>
<th>--PIs</th>
<th>--POs</th>
<th>--CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>146</td>
<td>33</td>
<td>26</td>
<td>26</td>
<td>16</td>
<td>00:15</td>
</tr>
<tr>
<td>2</td>
<td>123</td>
<td>35</td>
<td>26</td>
<td>26</td>
<td>16</td>
<td>01:02</td>
</tr>
<tr>
<td>3</td>
<td>146</td>
<td>28</td>
<td>26</td>
<td>26</td>
<td>16</td>
<td>01:01</td>
</tr>
<tr>
<td>4</td>
<td>140</td>
<td>37</td>
<td>26</td>
<td>26</td>
<td>16</td>
<td>00:52</td>
</tr>
</tbody>
</table>

Resource Use Estimate
Technology: xi4

Area: 123 Function Generators
Critical Path: 35 ns
DFFs: 26 (in CLBs or IOBs)
IOFFs: 0 (in IOBs)
HM CLBs: 0
Input Pins: 26
Output Pins: 16

Since Pass 2 produces the best results in terms of area, indicate -pass 2 on the next optimization of the .work.glob_dma INTERFACE design.

Description

The optimize command performs technology-specific logic optimization and technology mapping. You control optimization and mapping efforts by setting variables and constraints on the design boundaries, and indicating area and delay options.

Optimization and mapping results also depend on design rules. For example, the maximum capacitance allowed on a single driver as defined in a target technology library effects mapping results.

The default optimization action is to process each level of hierarchy, starting at the bottom of the hierarchy and optimizing one view at a time. The optimize command allows you to dissolve boundaries in some views and to group elements into new levels of hierarchy.

Functional Description of Boundary Optimization

The inputs to the hierarchical module in a hierarchical design may contain constants such as TRUE/FALSE. By propagating the constants across the boundary into the low level hierarchy, the design can be optimized more effectively. Similarly, unused outputs of a hierarchical module can be disconnected and common nets connecting to multiple ports can be merged to single net. This propagation occurs in both upward and downward directions. By default, hierarchical modules are checked for usage at the boundary, then the modules are grouped based on the common usage for boundary optimization. The created views are given the context names based on the higher level cell, instance, and the view name: <cellname_instancename_viewname>
The user may find that some of the lower level hierarchical modules have unused output ports or nets that are merged together. Disable this function with: set no_boundary_optimization TRUE

**Optimization Sequence**

1. The optimize command removes unused logic, and extracts and shares common logic among design elements, and extracts counters, rams and decoders.
2. The optimize command performs a series of optimizations based on algorithms and heuristics specific to the target technology. For example, the optimize command performs fanin-limited decomposition when you target the Xilinx technology.
3. The optimize command maps the design to the target technology.
4. The optimize command performs a design rule check.

**User Control**

You control optimization and mapping efforts by setting LeonardoSpectrum variables, setting constraints on the design boundaries, and indicating area and timing goals for the optimization and mapping. For more information refer to the Variables section in this guide. Boundary constraints include the following:

- timing constraints on input and output signals
- input drive and capacitance
- output load
- definition of clocking schemes

Design boundary information is useful in optimizing the timing of critical sections in the design during technology mapping.

You can also direct the optimize command not to optimize certain views in a hierarchical design by adding the noopt attribute to the views. The optimize command treats such views as black boxes for optimization purposes. Similarly, you can direct the optimize command not to optimize and/or delete certain signals by adding the preserve_signal attribute to the appropriate nets. **Note:** Refer to preserve_driver in the Attributes Section.

**Design Rules**

Design rules, specified in a target technology library by a vendor, also affect optimization efforts. Design rules address design characteristics relevant to physical design tools, such as place-and-route tools.

**Examples**

```
optimize -ta xi4 -effort standard -delay
```
This command optimizes the view pointed to by the present design. The design is targeted for the Xilinx 4000 technology library. The `optimize` command optimizes the view to minimize timing delays and uses a standard optimization effort.

```
optimize -ta flex10 -pass 4
```

This command optimizes the view pointed to by the present design. The design is targeted for the FLEX 10k technology library. The `optimize` command performs only optimization pass number 4.

**Related Commands, Variables**

**Commands**: `load_library, group, ungroup, load_library, pre_optimize`

**Variables**: `area_weight, check_complex_ios, complex_ios, delay_break_loops, delay_weight, dont_lock_lcells, extract_reduction_ops, global_sr, infer_gsr, insert_global_bufs, lut_map, maxarea, maxdly, max_cap_load, max_fanin, max_pt, max_transition, nl_use_approx, nologic_rep, nowire_table, optimize_cpu_limit, package, part, transformations, tristate_map, use_f5map, use_f6_lut, use_qclk_bufs, wire_table, wire_tree, no_boundary_optimization, asic_auto_dissolve_limit, auto_dissolve_limit, optimize_drc_resolving`
**optimize_timing**

Perform extensive timing optimization on a design.

**Syntax**

```plaintext
optimize_timing
    [<design_name>]
    [-through <node_lists>] [-force] [-single_level]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;design_name&gt;</td>
</tr>
<tr>
<td>list</td>
<td>&lt;node_list&gt;</td>
</tr>
</tbody>
</table>

**Arguments**

`<design_name>`

Name of the view in the design for which to optimize the timing. If you omit this argument, the command operates on the present design.

**Options**

`-through <node_list>`

A list of instances, ports, or nets. Given this list, the `optimize_timing` command tries to improve the timing along signal paths on which the indicated nodes participate.

`-force`

Makes the `optimize_timing` command use the longest paths as the basis of a timing constraint. Useful when no timing constraints are specified, but explicit constraints usually yield better quality of results.

`-single_level`

Performs timing optimization only at the top level of hierarchy. If you omit this option, the `optimize_timing` command traverses the entire design hierarchy.

**Description**

`optimize_timing` examines most critical paths and tries to improve the arrival_time at the end of each. This command is more effective if the user indicates timing constraints, such as the behavior of
clock signals, the arrival time of primary input signals, and the required time on primary output signals. The `optimize_timing` command infers timing constraints from these parameters.

You specify clocks by defining the clock cycle, pulse width, and clock offset. The following figure graphically defines these concepts.

```
| <- width -> |
| ___________ |
| | | |
| _____________| |_________________|
| <-- offset --> |
| <-- cycle ------------------------------> |
```

Arrival times and required times are specified in nanoseconds. These times are relative to a reference point, usually a clock edge.

The `optimize_timing` command enables you to define clocks and arrival times within either a VHDL source file or the LeonardoSpectrum command-line interface. An example VHDL specifications is as follows:

```
ATTRIBUTE CLOCK_CYCLE OF clock:SIGNAL is 20ns;
ATTRIBUTE CLOCK_OFFSET OF clock:SIGNAL is 5ns;
ATTRIBUTE PULSE_WIDTH OF clock:SIGNAL is 10ns;
ATTRIBUTE ARRIVAL_TIME OF inputa:SIGNAL is 3ns;
```

Using the LeonardoSpectrum interface, you would express this same information as follows:

```
CLOCK_CYCLE 20 clock
CLOCK_OFFSET 5 clock
PULSE_WIDTH 10 clock
ARRIVAL_TIME 3 inputa
```

**Examples**

The following command optimizes the timing of the present design and requires the `optimize_timing` command to use the longest path as a timing constraint.
optimize_timing

    optimize_timing -force

    The following command optimizes all critical paths in the present design that end at signal out1.

    optimize_timing -through out1

Related Commands, Variables

    Commands       ungroup, group, set_attribute
    Variables      delay_break_loops, maxarea, maxdly, max_cap_load,
                    max_fanout_load, max_transition, nl_use_approx, nowire_table,
                    optimize_timing_num_paths, wire_table, wire_tree
pack_clbs

Pack Look Up Tables (LUTs) into CLBs; Xilinx 4000 only.

Syntax

pack_clbs
  [<design>][-single_level]

Arguments

<design_name>
Name of the design in which the pack_clbs command is to pack configuration logic blocks (CLBs). The default is the present design.

Options

-single_level
Pack CLBs only at the top level of hierarchy. If you omit this option, the pack_clbs command traverses the entire design hierarchy.

Description

pack_clbs accepts a design that is mapped into FMAP or HMAP lookup tables (LUTs). The command packs FMAPs, HMAPs and DFFs into CLBs. This command applies only the Xilinx 4000 architecture. In XC4000, a CLB can pack up to a maximum of 2 FMAPs, 1 HMAP and 2 DFFs.

The advantage of the CLB packing operation is that you can get an accurate estimate of the design in terms of occupied CLBs.

After this command, area reports (such as those produced by report_area command) include an estimate of the number of CLBs used in the design.
**pack_clbs**

**Related Commands, Variables**

<table>
<thead>
<tr>
<th>Commands</th>
<th>optimize, report_area, write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>none</td>
</tr>
</tbody>
</table>

**Known Bugs, Limitations**

The `pack_clbs` command applies only to Xilinx 4000 architectures. The `pack_clbs` command is not effective if the design does not contain LUTs, such as before the `optimize` or after the `decompose_luts` commands.
**pre_optimize**

Do constant propagation and other pre-optimization on a design.

**Syntax**

```plaintext
pre_optimize
  [<design_name>]  
[-common_logic] [-unused_logic] [-extract] [-single_level] 
[-boundary] [auto_dissolve_hierarchy] [xor_comparator_optimize]
```

**Arguments**

**<design_name>**

Name of the view to perform preliminary logic optimization on. If you omit this argument, the command operates on the present design.

**Options**

- `-common_logic`
  
  Share operators and primitives that have common inputs.

- `-unused_logic`

  Remove logic that does not affect output signals from the design.

- `-extract`

  Recognize counters, decoders, and RAMs from generic logic. New views are created for the recognized logic.

- `-single_level`

  Perform preliminary logic optimization only at the top level of hierarchy. If you omit this option, the pre_optimize command traverses the entire design hierarchy.

- `-auto_dissolve_hierarchy`

  Dissolves a small hierarchy. Refer to `optimize` command and to Variables and Attributes chapter.
pre_optimize

-xor_comparator_optimize

Optimizes wide XORs and comparators by determining common sub-expressions.

Description

pre_optimize performs technology-independent logic optimization, such as resource sharing, removal of unused logic and extraction of data path elements, such counters, decoders, and RAMs.

LeonardoSpectrum always executes the pre_optimize command as part of the optimize command, so you rarely need to use the pre_optimize command independently. One case where you would use it is to investigate the design before going through a lengthy optimization and mapping process with the optimize command.

Examples

pre_optimize -common_logic -unused_logic -extract

This command performs all pre-optimization operations on all levels of the present design’s hierarchy.

Related Commands, Variables

Commands optimize

Variables extract_decoder, extract_ram
present_design

Print or change the present design.

Syntax

```
present_design
    [new_present_design_name]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;new_present_design_name&gt;</td>
</tr>
</tbody>
</table>

Arguments

`<new_present_design_name>`

Name of a design that identifies the top of a design hierarchy and on which LeonardoSpectrum performs actions. If you omit this argument, the `present_design` command returns the setting of the present design.

Description

Present design is a term to determine the design on which LeonardoSpectrum performs design-related operations. The default present design when you start LeonardoSpectrum is set to the top list of libraries in the LeonardoSpectrum design database, called the root. Once you read in a design from a design file, however, the present design is set to the top-level view as described in that file.

You use the `present_design` command to change the present design explicitly or list the present design. `new_present_design` can only be set on the root, a library, a cell or a view.
read

Read a file and create designs from it.

Syntax

read

<file_name(s)>

[-format <format_name>] [-dont_elaborate] [-design <design_name>]
[-architecture <string>] [-work <library_name>] [-parameters
<parameters_list>] | [-generics <generics_list>] [-technology <string>]

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;format_name&gt;, &lt;design_name&gt;, &lt;library_name&gt;, &lt;architecture_name&gt;, &lt;technology_name&gt;</td>
</tr>
<tr>
<td>list</td>
<td>&lt;file_name(s)&gt;, &lt;generics_list&gt;, &lt;parameters_list&gt;</td>
</tr>
</tbody>
</table>

Arguments

<file_name(s)>

Name of the input file or list of files. file_name(s) can be local filenames, relative path names, or absolute path names.

If multiple filenames are specified, use the Tcl list format, eg {name1 name2}. If filenames with spaces are specified, wrap the filename with spaces in either quotes or curly braces and wrap the list of one or more filename in curly braces, eg {“filename with spaces”} or {{filename with spaces}} or {“first filename with spaces” {second filename with spaces} normal filename}.

Example: read {“my files/file1.vhd” “other files/file2.vhd”}

Note

Always use the forward slash character (/) to separate directory names in a path, even on the PC. LeonardoSpectrum interprets the back-slash character (\) as a Tcl escape character.

Options

-format <format_name>
The format of the input file. Valid values are as follows: edif, sdf, verilog, vhdl, xdb, xnf, and preference.

If you omit this argument, the read command attempts to determine the file format from the filename extension, as shown in the following table.

<table>
<thead>
<tr>
<th>File Name Extension</th>
<th>File Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>.edf, .edif, .eds</td>
<td>edif</td>
</tr>
<tr>
<td>.sdf</td>
<td>sdf</td>
</tr>
<tr>
<td>.v, .verilog</td>
<td>verilog</td>
</tr>
<tr>
<td>.vhd, .vhdl</td>
<td>vhdl</td>
</tr>
<tr>
<td>.xdb</td>
<td>xdb</td>
</tr>
<tr>
<td>.xnf</td>
<td>xnf</td>
</tr>
<tr>
<td>Preference</td>
<td></td>
</tr>
</tbody>
</table>

If the read command cannot determine the file format, it prompts you for the information.

-dont_elaborate

Analyze input file only; do not elaborate. Valid only for HDL-format files. Using this argument is equivalent to using the command analyze file_name.

-design <design_name>

Specify top level design name to be read.

-work <library_name>

Store designs in the input file in library_name instead of the work library. This argument is only effective for VHDL, Verilog and XNF formats, where the library of a design is not defined in the file itself.

-technology <technology_name>

Specify a target technology to guide analysis.
**read**

**Description**

`read` copies a design from a file into the LeonardoSpectrum design database.

If no errors occur, the `read` command sets the present design to the top-level design in the file. Which design in the input file is the top-level design depends on the format of the file.

- **EDIF**: View defined by the EDIF 'design' construct
- **VHDL**: Last architecture of the last entity in the file
- **Verilog**: First module in the file
- **XDB**: XDB is a binary dump of the hierarchical design database. The netlist is saved in the original form.
- **XNF**: Design in the file (there is only one design per file possible in xnf).

Since SDF defines only timing information, the `read` command does not set the present design. Instead, the tool annotates the timing information in the file on the present design. Therefore, make sure that `present_design` is set to the appropriate existing design.

For all formats except SDF, if a design with the same name as the design being read already exists in the database, the `read` command replaces the existing design with the new version and issues a warning message. If there were already references to the existing design from other designs in the database, the `read` command links the port references to the new design automatically. If ports in the existing design are missing in the new design, the tool disconnect the references to those ports and issues a warning.

Since the XNF format does not give the design a name, `read` uses the prefix of the source file as the cell name and the view name of the created design. Regardless of the format, if the file contains references to technology cells, make sure to load the appropriate library (using the `load_library` command) first. Otherwise, `read` creates black boxes for all these cells, and area, timing and functional information is not present.

**XDB Designs Only**

The output file is saved in a format that can be read back into LeonardoSpectrum without processing the netlist to remove technology-specific information. XDB writes a binary dump of your hierarchical database to a file. You can read this file back into LeonardoSpectrum. During the write command the library, cell and view objects were saved. The view includes nets, instances, attributes. Separate nets, ports, instances are low level objects and are not saved during write.
read

Note: During read, the necessary libraries for the design must be read in before the design is read in. The read command reads the file containing information in XDB format and reconstructs the design database. The technology libraries are required during the reconstruction.

Limitations

The read command reads VHDL and Verilog designs in two steps: analysis and elaboration. read filename for VHDL or Verilog, therefore, is equivalent to the following two separate commands:

```
analyze filename
elaborate
```

The read command can operate only on VHDL or Verilog designs that meet the following criteria:

- The design is described in a single file.
- The top-level design does not have uninitialized generics or parameters.
- The top-level design is either the last design in the file (VHDL) or the first (Verilog).

In all other cases, you need to run the analyze and elaborate commands separately.

VHDL and Verilog Designs Only

The read command file(s) are analyzed and then the top level design is elaborated. During elaboration only the architectures/modules in the lower level of hierarchy - present in the files(s) - would be elaborated. Any other lower level designs are either picked up from the hierarchical database or a black box is created. This is for bottom up design methodology. If you read and optimize the lower level of hierarchy first and then read a file containing the top level design, the lower level designs are untouched and only the top level design is elaborated. You can do optimization on the top level design only. Read in a design which is distributed over multiple files with 'read {file names}'. You can read the files individually by analyzing all of the files and then doing elaborate on the top level design.

Scripts

The script is available from the new Utilities menu, and includes a description on when and how the script is used. A brief description follows:

`auto_read` automatically does needed processing for technology netlists in addition to the read command, including setting of variable values. This script is invoked automatically from the Read dialog when "Do Automatic Processing" is checked.

Note this script replaces the `read_altera` script available with previous releases of LeonardoSpectrum.
Related Commands, Variables

Commands  analyze, elaborate

Variables  edif_function_property, encoding, full_case,
           hdl_array_name_style, hdl_integer_name_style,
           hdl_record_name_style, parallel_case, preserve_dangling_net,
           sdf_hierarchical_names, sdf_type, use_dffenable, viewlogic_vhdl,
           vhdl_87, xlx_preserve_gsr, xlx_preserve_gts, xlx_preserve_pins
remove

Remove object(s).

Syntax

remove
  <object_name>
  [-port]|[-net]|[-instance] [-hdl]

Arguments

<object_list>

List of names of objects to be removed. Object names are case sensitive and wildcards are accepted

Options

-port, -net, -instance
  Indicator that the object name(s) refer to ports, nets or instances, respectively. If you omit this argument, the remove command assumes that object_list refers to instances unless object_list refers to libraries, cells or views.

-hdl
  Remove objects (libraries, packages, entities, architectures, configurations, modules) from the hdl database, rather than objects from the design database.

Description

Remove objects (libraries, cells, views, ports, nets, instances) from the design database. If an object does not exist, an error is issued. When a port, net or instance is removed, any connections to it are removed as well.

Examples

remove .work

Remove the entire library work from the design database
remove

remove -hdl .ieee.std_logic_1164

Remove the VHDL ieee package std_logic_1164 from the hdl data base.

remove -port P

Remove port P in the present design, and disconnect it from any net it might be connected to. Only valid if the present design is a view.

Related Commands, Variables

Commands  create, move, copy, connect, disconnect
Variables  none

Known Bugs, Limitations

remove will refuse to remove the present design. For this reason, make sure to always specify a full name (starting with .) when a library, cell or view needs to be removed. You can change the present design with the present_design command beforehand.

remove will refuse to remove a view if there are still references to it. A warning message is issued.
remove_attribute

Remove an attribute from object(s).

Syntax

```
remove_attribute
   [<object_list>]
[-port] | [-net] | [-instance] [-type <attribute_name>]
   [<object_list>] [-global]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;attribute_name&gt;</td>
</tr>
<tr>
<td>list</td>
<td>&lt;object_list&gt;</td>
</tr>
</tbody>
</table>

Arguments

- `<object_list>`
  List of names of objects to be removed. Object names are case sensitive and wildcards are accepted

Options

- `-port`, `-net`, `-instance`
  Indicator that the object names refer to ports, nets or instances, respectively. If you omit this argument, the remove_attribute command assumes that `object_list` refers to instances unless `object_list` refers to libraries, cells or views.

- `-name`
  Name of the attribute to be removed. Wildcards are not accepted. Attribute names are case insensitive.

- `-global`
  Remove attributes from nets globally (all levels of hierarchy).

Description

Remove an attribute from an object in the design database.
**remove_attribute**

**Examples**

```sh
class <name>
remove_attribute -port clk -name PIN_NUMBER

Remove attribute PIN_NUMBER from port clk of the present design. Only valid if the present design is a view.

class <name>
remove_attribute -port { in1 bus_a(*) } -name ARRIVAL_TIME

Remove the attribute ARRIVAL_TIME from the ports in1 and all the ports of the bundle bus_a of the present design. Only valid if the present design is a view.

class <name>
remove_attribute -name NOOPT

Remove the attribute NOOPT from the present design.
```

**Related Commands, Variables**

- **Commands**: list_attributes, set_attribute
- **Variables**: none
**remove_clock**

Remove the clock information from object(s).

**Syntax**

remove_clock

<remove_clock>

**Arguments**

<remove_clock>

The object is a port, net, or instance.

[<remove_clock>]

[-port] [-net]

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>-name &lt;string&gt;</td>
</tr>
<tr>
<td></td>
<td>Name of the object.</td>
</tr>
<tr>
<td>string</td>
<td>-type &lt;string&gt;</td>
</tr>
<tr>
<td></td>
<td>Type of attribute (default) - remove even if type not specified.</td>
</tr>
</tbody>
</table>
remove_rename_ruleset

Remove a ruleset for object renaming.

Syntax

remove_rename_ruleset
   <ruleset_name>

Arguments

<ruleset_name>

Remove ruleset ruleset_name for object renaming. Ruleset was previously created with create_rename_ruleset.

Related Commands, Variables

Commands  apply_rename_rules, create_rename_ruleset, add_rename_rule, report_rename_rules

Variables  none
**report_area**

Report the accumulated area of the present design.

**Syntax**

```
report_area
   [<report_file_name>]
   [-cell_usage] [-hierarchy] [-all_leafs]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td><code>&lt;report_file_name&gt;</code></td>
</tr>
</tbody>
</table>

**Arguments**

`<report_file_name>`

Name of the output file in which to write the design area report. If you omit this argument, the report goes to standard output screen. This is a string type of argument.

**Options**

```
-cell_usage
   report cell usage per instance in design.

-hierarchy
   report all levels of hierarchy separately.

-all_leafs
   report on all leaf cells, including black boxes.
```

**Description**

`report_area` is a general-purpose area reporting routine.

For a technology-independent (not optimized) design, the `report_area -all_leafs` command gives an overview of the complexity of the design prior to technology mapping. The report includes the total number of primitives (AND, OR) and operators (add, subtract, multiply), and a count of the black boxes. On a mapped (optimized) design, the same command produces a report that includes technology-specific area information: function generators and flip-flops for Xilinx designs, combinational and sequential modules for Actel designs.
report_area

Examples

LEONARDO: report_area

*******************************************************
Cell: traffic    View: exemplar    Library: work
*******************************************************
Number of ports : 9
Number of nets : 35
Number of instances : 32
Number of references to this view : 0
Total accumulated area :
Number of CLB Flip Flops : 3
Number of H Function Generators : 2
Number of Packed CLBs : 9
Number of FG Function Generators : 17

Related Commands, Variables

Commands  decompose_luts, optimize, pack_clbs, pre_optimize, read, write
Variables  report_area_format_style
report_constraints

List user constraints on any object.

Syntax

```
report_constraints
    [<design_name>]
    [-port] [-net] [-hierarchy]
```

Arguments

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;design_name&gt;</td>
</tr>
</tbody>
</table>

Options

- `-port`

  Report constraints on ports only. If you omit this argument, both port and net constraints are reported.

- `-net`

  Report constraints of nets only. If you omit this argument, both port and net constraints are reported.

- `-hierarchy`

  Report constraints on all levels of hierarchy in the design. If you omit this option, only constraints at the top level of hierarchy are reported.

Description

Report constraints on a design. Design constraints are modeled as attributes on ports and nets. Use these methods to set constraints:

* Use the `set_attribute` command.
**report_constraints**

* Use a design constraint Tcl script from the exemplar.ini file.
* Use the constraint editor on the GUI.
* Set attributes on ports in VHDL or Verilog design source code.

`report_constraints` reports constraints set with any of these methods.

**Examples**

```
LEONARDO: report_constraints [present_design]
-- Printing User constraints for view .work.prep1_2.top_level
  clk CLOCK_CYCLE 20
  clk CLOCK_OFFSET 5
  rst ARRIVAL_TIME 2
  q(7) REQUIRED_TIME 15
  q(6) REQUIRED_TIME 15
  q(5) REQUIRED_TIME 15
  q(4) REQUIRED_TIME 15
  q(3) REQUIRED_TIME 15
  q(2) REQUIRED_TIME 15
  q(1) REQUIRED_TIME 15
  q(0) REQUIRED_TIME 15
```

**Related Commands, Variables**

Commands  list_attributes, set_attribute

Variables  none

**Known Bugs, Limitations**

LeonardoSpectrum does not automatically propagate design constraints across levels of hierarchy. Every level of hierarchy is considered separately. Exemplar Logic has a Tcl script available upon request that propagates design constraints across levels of hierarchy. In absence of this script, specify constraints on all ports in the design separately, regardless of the level of hierarchy.
report_delay

Report timing information about the design.

Syntax

report_delay

[<report_file_name>]
[-num_paths <number>] [-longest_path][-end_points] [-start_points]
[-clock_frequency] [-critical_paths] [-no_io_terminals]
[-no_internal_terminals] [-show_input_pins] [-show_nets] [-through
<node_list>] [-from <start_points>] [-to <end_points>] [-not_through
<node_list>] [-show_schematic <number>]

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;report_file_name&gt;</td>
</tr>
<tr>
<td>list</td>
<td>&lt;node_list&gt;, &lt;start_points&gt;, &lt;end_points&gt;</td>
</tr>
<tr>
<td>integer</td>
<td>&lt;number&gt;</td>
</tr>
</tbody>
</table>

Arguments

<report_file_name>

Name of file in which to write the delay report. report_file_name can be a local file name, a relative path name, or an absolute path name. If you omit this argument, the report appears on the standard output screen.

Options

-num_paths <number>

Number of paths to report. The report_delay command reports on paths in descending order of criticality. If you omit this argument, the report_delay command reports 10 critical paths.

-longest_path

Sort paths in the design by length, report the longest path first, and ignore criticality.
**report_delay**

- **-end_points**
  Reports slack, arrival and required times at end points only. This option may be used in combination with `-start_points` and/or `-critical_paths`. By default, `report_delay` reports on critical paths only. This default is identical to the `-critical_paths` option.

- **-start_points**
  Reports slack, arrival and required times at start points only.

- **-critical_paths**
  Reports critical paths only.

- **-no_io_terminals**
  Report only those paths that terminate in inputs to registers, ignoring paths that terminate in design outputs or primary outputs.

- **-no_internal_terminals**
  Report only those paths that terminate in design outputs or primary outputs, ignoring paths that terminate in input to registers.

- **-show_input_pins**
  Include the input pins of gates in the report.

- **-show_nets**
  Include in the report the names of nets being driven by the output of gates in addition to the output pin names of gates.

- **-through <node_list>**
  Report only those paths through ports and instances indicated by `node_list`.

- **-not_through <node_list>**
  Do not report paths through ports and instances indicated by `node_list`.

- **-from <start_points>**
  Report only those paths that originate at the indicated input ports or register outputs `<start_points>`.

- **-to <end_points>**
  Report only those paths that terminate at the indicated input ports or register outputs `<end_points>`.
report_delay

-show_schematic <number>

Number of critical path schematics to generate. This is an integer value.

Description

report_delay reports the timing status of a design. The command creates a list of critical paths and, by default, displays them from most critical to least critical. (A critical path is one that has a slack of 0 nanoseconds or less.)

Examples

The following command example reports the most critical path in a given design. The report generated by the command follows the command example.

report_delay -num 1

Critical Path Report

Critical path #1, (unconstrained path)

<table>
<thead>
<tr>
<th>NAME</th>
<th>GATE</th>
<th>ARRIVAL</th>
<th>LOAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay thru clock network</td>
<td>0.0 (ideal)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMPLR_INST_17/I1/Q</td>
<td>FD</td>
<td>3.00</td>
<td>0.0</td>
</tr>
<tr>
<td>modgen_0/XMPLR_NET_8/O</td>
<td>F3_LUT</td>
<td>7.50</td>
<td>0.0</td>
</tr>
<tr>
<td>XMPLR_NET_8/O</td>
<td>F3_LUT</td>
<td>12.00</td>
<td>0.0</td>
</tr>
<tr>
<td>XMPLR_NET_9/O</td>
<td>F4_LUT</td>
<td>16.50</td>
<td>0.0</td>
</tr>
<tr>
<td>XMPLR_NET_155/O</td>
<td>H3_LUT</td>
<td>19.00</td>
<td>0.0</td>
</tr>
<tr>
<td>nxstate(1)/O</td>
<td>F4_LUT</td>
<td>23.50</td>
<td>0.0</td>
</tr>
<tr>
<td>XMPLR_INST_15/I1/D</td>
<td>FD</td>
<td>23.50</td>
<td></td>
</tr>
</tbody>
</table>

data arrival time

23.50

data required time

not specified

unconstrained path
report_delay

Related Commands, Variables

Commands
- optimize, optimize_timing, report_area

Variables
- delay_break_loops, nl_use_approx, nowire_table, propagate_clock_delay, report_delay_analysis_mode, report_delay_arrival_threshold, report_delay_detail, report_delay_format_style, report_delay_slack_threshold, wire_tree, wire_table

Known Bugs, Limitations

report_delay treats all designs as if they are flat. If a design is hierarchical, the command reports the critical paths through the hierarchy, without regard to hierarchy boundaries.
**report_rename_rules**

Report loaded rename ruleset information.

**Syntax**

```
report_rename_rules
[<ruleset_name>]
```

**Arguments**

`<ruleset_name>`

Report the loaded rename ruleset information.

**Related Commands, Variables**

- **Commands**
  - apply_rename_rules, create_rename_ruleset, add_rename_rule, remove_rename_ruleset
- **Variables**
  - none
**report_wire_tables**

Reports all the wire tables in the library.

**Syntax**

```plaintext
report_wire_tables [report_file_name]
```

**Arguments**

```plaintext
report_wire_tables [report_file_name]
```

**Options**

- `[-library <list>]` Report wire tables in library.
- `[-summary]` Report only a summary of the wire table.
set_attribute

Create or set an attribute on an object(s).

Syntax

```
set_attribute
    [<object_name>]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td><code>&lt;object_name&gt;</code>, <code>&lt;attr_name&gt;</code>, <code>&lt;attr_type&gt;</code>, <code>&lt;attr_value&gt;</code></td>
</tr>
<tr>
<td>list</td>
<td><code>&lt;object_name&gt;</code></td>
</tr>
</tbody>
</table>

Arguments

`<object_name>`

Name of the object (library, cell, view, port, net or instance) for which the `set_attribute` command sets an attribute value. Wildcards and lists are accepted. If you omit this argument, the `set_attribute` command operates on the present design.

Options

- `-port`, `-net`, `-instance`

Indicator that `object_name` refers to a port, net, or an instance, respectively. If you omit this argument, the `set_attribute` command assumes that `object_name` refers to an instance unless `object_name` refers to a library, cell or view.

- `-name <attr_name>`

Simple name for the attribute whose value is being set. Attribute names are case-insensitive.

- `-type <attr_type>`

Data type of the attribute whose value is being set. Valid values depend on the attribute indicated with the `-name` option.

- `-value <attr_value>`

Alphanumeric string to assign to named attribute.
**set_attribute**

**Description**

The *set_attribute* command assigns a value to an attribute on an object in the design database.

If the object already has an attribute with the same name as that indicated by the `-name` option, the *set_attribute* command overwrites the existing value with the newly specified value.

**Examples**

```plaintext
set_attribute -port clk -name PIN_NUMBER -value "P14"
```

Set attribute `PIN_NUMBER` on port `clk` of the present design to the string `P14`. Valid only if the present design is a view. This is an alternative to the `PIN_NUMBER` script command.

```plaintext
set_attribute -port {in1 bus_a(*)} -name ARRIVAL_TIME -value 1.3
```

Set the attribute `ARRIVAL_TIME` to 1.3 (ns) for the port `in1` and all the ports of the bundle `bus_a` of the present design. Valid only if the present design is a view. This is an alternative to the `ARRIVAL_TIME` script command.

```plaintext
set_attribute .work -name Version -value "My library version 3.0"
```

Set the attribute `Version` on the library `work` to the string `My library version 3.0`.

```plaintext
set_attribute -name NOOPT -value TRUE
```

Set attribute `NOOPT` on the present design to `TRUE`. The present design could be anything. This is an alternative to the `NOOPT` script command.

**Related Commands, Variables**

**Commands**  
list_attributes, remove_attribute

**Variables**  
none
set_clock

Create or set clock information on an object(s).

Syntax

set_clock

[<set_clock>]
[-port] [-net]

Arguments

<set_clock>

The object is a port, net, or instance.

[<set_clock>]
[-port] [-net]

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>[-clock_cycle &lt;string&gt;]</td>
</tr>
<tr>
<td></td>
<td>clock cycle</td>
</tr>
<tr>
<td>string</td>
<td>[-clock_offset &lt;string&gt;]</td>
</tr>
<tr>
<td></td>
<td>clock offset</td>
</tr>
<tr>
<td>string</td>
<td>[-pulse_width &lt;string&gt;]</td>
</tr>
<tr>
<td></td>
<td>pulse width</td>
</tr>
<tr>
<td>string</td>
<td>-name &lt;string&gt;</td>
</tr>
<tr>
<td></td>
<td>name of object</td>
</tr>
<tr>
<td>string</td>
<td>[-type &lt;string&gt;]</td>
</tr>
<tr>
<td></td>
<td>type of attribute (default)</td>
</tr>
</tbody>
</table>
**set_multicycle_path**

Constrains a path that requires more than one clock cycle.

**Syntax**

```
set_multicycle_path
  <-value number> <-from start_points> <-to end_points>
  [-to <list>] [-from <list>] [value <integer>]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>&lt;start_points&gt;, &lt;end_points&gt; &lt;to&gt; &lt;from&gt;</td>
</tr>
<tr>
<td>integer</td>
<td>&lt;number&gt; &lt;value&gt;</td>
</tr>
</tbody>
</table>

**Arguments**

`<-value number>`

This is the number of clock cycles needed to account for the path delay.

`<-from start_points>`

`start_points` is a list of names of instances and ports at the path start point.

`<-to end_points>`

`end_points` is a list of names of instances and ports at the path end point.

**Options**

`-setup, -hold`

Specifies the use of the `<value number>` of clock cycles. `-setup` is for start point number of clock cycles. `-hold` is for end point number of clock cycles. The default is `-setup`.

`-rise, -fall`

Indicates the effect of `<value number>` on delays. `-rise` specifies that rising path delays are affected at path end point. `-fall` indicates a falling value at the path end point. Default: both rising and falling delays are affected.

`-value`

Specifies number of cycles.
set_multicycle_path

-to
   Specifies target of node list.

-from
   Specifies source of node list.

-src_clk, -dest_clk
   Indicates if the <value number> information is relative to the start clock or to the end clock.
   -src_clk is equivalent to the input of the path. -dest_clk is equivalent to the output of the path.

-unset
   -unset removes multicycle constraints from path.

Note: The register-to-register constraint does not work with set_multicycle_path. You must set the clock to make the set_multicycle_path active.
unalias

Remove an alias.

unalias

<alias_name>

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&lt;alias_name&gt;</td>
</tr>
</tbody>
</table>

Arguments

<alias_name>

Alias to be removed. This is a string type of argument.

Description

The unalias <alias_name> command removes an alias previously created with the alias command.

Examples

unalias dl

Removes the alias definition of dl.

Related Commands, Variables

Commands alias, help

Variables none
Duplicate views that are used more than once (folded) in the design hierarchy.

**Note:** unfold is only available to Level 3.

```
unfold  
[<design>]  
```

**Arguments**

*<design>*

Ensure that all instances in the given view or views refer to unique cell or view.

**Description**

The result of reading a design into LeonardoSpectrum is a netlist in which there may be multiple instances which refer to the same cell/view. This is referred to as the folded state. While the design is in the folded state, you can save it, view reports, set the environment, select instances, and perform optimization. When you select the view of a level of hierarchy and issue the unfold command, LeonardoSpectrum alters the design so that each instance refers to a unique cell/view. This means that LeonardoSpectrum “clones” all hierarchical cells with multiple references so that each cell is referenced by only one instance. This is the unfolded state.

By default, LeonardoSpectrum represents multiple hierarchical instances of the same cell with a single view or folded state. Operators are the most common instances to be folded.

**Examples**

The unfold command operates on a level of hierarchy (view). Unfold ensures that all instances in the view refer to unique cell or views.

```
unfold work.ufold_ex.example  
```
**ungroup**

Flatten out hierarchy.

```
ungroup
  <instance_list>
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td><code>&lt;instance_list&gt;</code>, <code>&lt;exclude_instance_list&gt;</code></td>
</tr>
</tbody>
</table>

**Arguments**

`<instance_list>`

Name or names of instances to decompose into non-hierarchical instances. You can use names of any existing instances, including those created previously with the `group` command. Wildcards are allowed. You may use the `-all` option in place of specifying `instance_list`.

**Options**

- `-hierarchy`
  
  Remove all levels of hierarchy under all instances identified in `instance_list`; then remove hierarchy recursively under each new instance, until all levels of hierarchy have been removed.

- `-simple_names`
  
  Use original, non-hierarchical names for new instances. If you omit this argument, the `ungroup` command generates names automatically. The format for new instance and net names is as follows:

  `<ungrouped instance name> ungroup_hier_separator <original name>`

  The default `ungroup_hier_separator` is the underscore (`_`) character.

For example, suppose a view `TOP` contains an instance called `X`. `X` points to a view `V`. Suppose also that `V` contains a net `N` and an instance `I`. If you execute the command `ungroup X` when the present design is `TOP`, the instance `X` will be removed, and the contents in the view that `X` is pointing to (`V`) will be copied to `TOP` and get new names:

- Net `N` (in `V`) will be copied to a net called `X_N` in `TOP`
- Instance `I` (in `V`) will be copied to instance `X_I` in `TOP`
**ungroup**

- **-all**
  
  Decompose every instance in the current level of hierarchy of the present design. The `-all` option is equivalent to using the `*` character for `instance_list` and may be used in place of specifying `instance_list`.

- **-except <exclude_instance_list>**
  
  Exclude the named instances in `instance_list` from the ungroup operation.

- **-force**
  
  Flatten out cells, even noopt or technology cells.

**Description**

Remove one or more levels of hierarchy from a design by decomposing the instances named in `instance_list`, excluding the instances named in `exclude_instance_list`.

**Examples**

```plaintext
ungroup -all -hierarchy
```

In this example, the `ungroup` command removes all hierarchy under the present design. After this command, the present design will be a flat netlist of primitives or technology cells.

```plaintext
ungroup (x y) -except x -hierarchy
```

In this example, the `ungroup` command ungroups all hierarchy under the instance `y`. Instance `x` is unaffected, since instance `x` is a parameter of the `-except` option.

```plaintext
ungroup x
```

In this example, the `ungroup` command ungroups only instance `x` in the present design. Additional hierarchy under the view pointed to by `x` remains in place.

**Related Commands, Variables**

**Commands**

- `group`
- `present_design`

**Variables**

- `ungroup_hier_separator`
unmap

Flatten out technology cells in the design down to primitives.

Syntax

unmap

[<design_name>] [-single_level]

Arguments

<design_name>

Name of the design in which the unmap command is to flatten out technology cells. The default is the present design.

Options

-signal_level

unmap technology cell instances only at the top level of hierarchy.

Description

The unmap command flattens out technology cells in the design down to primitives. The design can then be written out in VHDL or Verilog and is simulatable without a technology library. Note: After unmap command runs, the design may contain redundant logic and be large. Run pre_optimize -common -unused. This cleans the design, removes constants, shared logic and unused logic. The functionality of the design is not changed. unmap unmaps a design that is mapped to a technology by optimize. unmap unmaps the design back to primitives. The design after unmap probably does not have the same structure as the design before optimize.

Caution for FPGA designs with lookup tables (luts): Before you use unmap, run the decompose_luts command. Refer also to auto_write in the Utilities chapter.

Related Commands, Variables

Commands

write, optimize

Variables

none
write

Write a design to a file.

Syntax

```
write
  <file_name>
  [-format <format_name>] [-downto <library_name>] [-silent]
  [-single_level] [-design <design_name>]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td><code>&lt;format_name&gt;</code>, <code>&lt;design_name&gt;</code>, <code>&lt;library_name&gt;</code></td>
</tr>
<tr>
<td>list</td>
<td><code>&lt;file_name&gt;</code></td>
</tr>
</tbody>
</table>

Arguments

```
<-file_name>
```

Name of the output file. `<file_name>` can be a local file name, a relative path name, or an absolute path name. If you use a dash character (`-`) for `<file_name>`, the output appears on the standard output screen.

---

Note Always use the forward slash character (`/`) to separate directory names in a path, even on the PC. LeonardoSpectrum interprets the back-slash character (`\`) as a Tcl escape character.

---

Options

```
-format <format_name>
```

The format of the output file. Valid values are as follows: edif, sdf, verilog, vhdl, xdb, xnf, and preferences.

If you omit this argument, the write command attempts to determine the file format from the extension in the file name as shown in the following table.
write

<table>
<thead>
<tr>
<th>File Name Extension</th>
<th>File Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>.edf, .edif, .eds</td>
<td>edif</td>
</tr>
<tr>
<td>.sdf</td>
<td>sdf</td>
</tr>
<tr>
<td>.v, .verilog</td>
<td>verilog</td>
</tr>
<tr>
<td>.vhd, .vhdl</td>
<td>vhdl</td>
</tr>
<tr>
<td>.xdb</td>
<td>xdb</td>
</tr>
<tr>
<td>.xnf</td>
<td>xnf</td>
</tr>
</tbody>
</table>

Preference

If the output file has a file name extension that the write command does not recognize, you must indicate the format explicitly. If the write command cannot determine the format, it prompts you for the information.

-downto <library_name>

Do not write the contents of any cells in library_name. Essentially, this argument indicates the leaf level of design hierarchy to be written. If you omit this argument, the write command writes design hierarchy down to primitives (or technology cells).

-silent

Do not write any warnings or informational messages.

-format <format_name>

Specify the format: vhdl|Verilog|edif|xnf|xdb|sdf|preference

-single_level

Write only the top level of (present design) hierarchy. If you omit this option, the write command writes the entire design hierarchy of the present design.

-design <design_name>

By default, the current design is saved during write command.
write

Description

The write command records the present design to a file in the file format you indicate. You can use this command at any point in any design flow, as long as there is a present design.

The write command ensures that all identifiers that are written out comply with syntax restrictions in the indicated format by using built-in renaming rules. If you want to change names of identifiers before writing out a netlist, you need to apply renaming rules before the write command. This could be needed if you want to guarantee that identifiers have the same name in multiple formats, or if you want to comply with stricter syntax rules on names, imposed by tools that will read the file.

The write command never changes anything in the design database.

Format-Specific Considerations

EDIF

Since the Leonardo Spectrum design database uses the same information model as EDIF, the write command always writes EDIF files with a one-to-one reflection correspondence to all objects in the design. Libraries, cells, views, nets, ports, instances and attributes are written out exactly as they exist in the design. Therefore, the EDIF format is useful to save and restore design data to a file at any point in the design flow.

Verilog, VHDL

Since Leonardo Spectrum has no internal representation for busses, the write command always writes all nets and ports as single bits. To simulate a synthesized design against its VHDL or Verilog original (with busses), you can create a wrapper file that will re-install the types of the original ports. For more information, see the create_wrapper command.

If you write a design expressed in terms of technology-independent primitives (before the optimize command), VHDL and Verilog formats contain simulatable dataflow statements. After technology mapping, all formats (except SDF) contain netlist descriptions in terms of technology cells.

SDF

If you indicate SDF for the output format, the write command writes only annotated timing information. This information is in a design only after previously reading in SDF or XNF timing information that was back-annotated from place-and-route tools. Therefore, writing SDF makes sense only for support of (timing) simulation of back-annotated designs.
write

XDB

During the write command, the output file is saved in a format that can be read back into LeonardoSpectrum without processing the netlist to remove technology-specific information. XDB writes a binary dump of your database to a file. The netlist is saved in the original condition during write. The objects related to the hierarchical database are saved: libraries, cells, views. Individual nets, ports, instances are low level objects that are not saved. In addition, constraints, if any, are not saved and must be reapplied.

XNF

The XNF format does not support multiple levels of hierarchy in a single file. Therefore, the write command creates a new file for every level of hierarchy in the design. The new filenames have the prefix of the cell name they write out, and a suffix of .xnf.

Lookup Tables (LUTs)

Lookup Tables (LUTs), used by LUT mapping routines for Xilinx, ORCA and Altera FLEX technologies, are written as dataflow statements for VHDL and Verilog output formats. For the XNF format, LUTs are written as EQN symbols.

Scripts

auto_write automatically does needed processing for your target technology in addition to the write command, including setting of variable values and calling generate_timespec, decompose_luts and apply_rename_rules, where applicable. Refer to the Utilities chapter for a complete description of auto_write.

Note: this script replaces the write_altera script available with previous releases of Leonardo.

Related Commands, Variables

<table>
<thead>
<tr>
<th>Commands</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_rename_rules</td>
<td>edif_function_property, names_collision_extension,</td>
</tr>
<tr>
<td>create_wrapper</td>
<td>sdf_hier_separator, sdf_names_style, vhdl_generic_to_attribute,</td>
</tr>
<tr>
<td>read</td>
<td>vhdl_write_bit, vhdl_write_use_packages, write_clb_packing,</td>
</tr>
<tr>
<td>list_technologies</td>
<td>write_lut_binding</td>
</tr>
</tbody>
</table>
write
Variables

This chapter is divided as follows:

- Tcl Syntax
- GUI: Tools -> Variable Editor
- Alphabetical List of Variables

Tcl Syntax

Since Leonardo Spectrum supports the Tcl language, then variable assignments and evaluation according to Tcl syntax (`set` for setting variables and `$var_name` to evaluate a variable) are also supported.

The Tcl variables in this section have direct links into the Leonardo Spectrum synthesis engine. These variables affect the behavior of the Leonardo Spectrum synthesis commands. A list of Leonardo Spectrum variables can be displayed using `help -variables` (help -v).

For example, a Tcl variable with a link to Leonardo Spectrum is `hdl_array_name_style`. When you type:

```
help -v hdl_array_name_style
```
The following message is displayed:

```
hdl_array_name_style = %s(%d) -- bit name style for array type objects
```

This variable is set to %s(%d) by default.

This variable defines the name style to use when VHDL or Verilog array objects (vectors) are compiled (in the `read` or `elaborate` command) into sets of bits. Using the default setting, a VHDL array `my_bus (0 to 3)` creates four bits in your design, called `my_bus(0)`, `my_bus(1)`, `my_bus(2)`, and `my_bus(3)`. If you change this variable to another value using the Tcl command `set`:

```
set hdl_array_name_style %s%d
```

then the bus names do not contain the () parentheses for the next time you read or elaborate a VHDL or Verilog file. In this example, the bits are called `my_bus0`, `my_bus1`, `my_bus2`, and `my_bus3`.

**GUI Tools -> Variable Editor**

Refer to the User's Guide, Chapter 11, Menu Bar Items.

**Alphabetical List of Variables**

The following list briefly describes the variables that are available for use with LeonardoSpectrum commands.

**Note:** Variables are applied globally in your design, while attributes are applied to ports. Attributes take precedence over variables. Some variables are also attributes.
allow_black_box_modgens

Produce a black box instead of an error for modgens for which there is no implementation. If a modgen does not have an implementation, then a black box is produced instead of an error.

Default value: FALSE

alt_auto_fast_io

set alt_auto_fast_io is for Altera place and route.

Default value: FALSE

alt_auto_register_packing

set alt_auto_register_packing is for Altera place and route.

Default value: FALSE

altera_allow_cascade_fanout

Set for Altera FLEX 6/8/10 and APEX 20K/20KE.

Default value: TRUE

Map to cascades with fanouts. Applies if altera_use_cascades is TRUE.

altera_cascade_chain_length

Set for Altera FLEX 6/8/10 and APEX 20K/20KE.

Chain length = 8 for FLEX 8 and FLEX 10; chain length = 10 for FLEX 6

Chain length = 10 for APEX 20K/20KE.

altera_use_cascades

Map logic to CASCADEs during LUT mapping for Altera FLEX 6/8/10 and APEX 20K/20KE.

Default value: TRUE

Related commands: optimize
area_weight

Indication of how much effort LeonardoSpectrum should make to minimize design area when mapping a design; higher values indicate more effort.

Default value: -1.000000

Valid Values: Numbers between 0 and 1.0

Related commands: optimize

asic_auto_dissolve_limit (30 gates)

The asic_auto_dissolve_limit variable dissolves (flattens by default) blocks of hierarchy that contain counts of 30 or fewer gates. Blocks or modules are dissolved in a context sensitive manner. If a module is instantiated two or more times in the same design and the gate count is 30 or less, then the module instantiation is dissolved. You can set asic_auto_dissolve_limit to 0 to disable. Refer to Attribute chapter for information on the auto_dissolve attribute, and refer also to optimize in the Command chapter.

Example: set asic_auto_dissolve_limit 20

Default value: 30 gates

auto_dissolve_limit (FPGA/CPLD 3000 gates)

The auto_dissolve_limit variable dissolves (flattens by default) blocks of hierarchy that contain counts of 3000 or fewer gates. Blocks or modules are dissolved in a context sensitive manner. If a module is instantiated two or more times in the same design and the gate count is 3000 or less, then the module instantiation is dissolved. You can set auto_dissolve_limit to 0 to disable. Refer to Attribute chapter for information on the auto_dissolve attribute, and refer also to optimize in the Command chapter.

Example: set auto_dissolve_limit 2000

Default value: 3000 gates

balance_adders

Balance adders/subtractors cascaded in series.

Default value: true
big_mux_percentage_for_lut <integer>

This is the upper cutoff of the percentage of unique data signals for creating a LUT node.

Default value: 50

Related commands: optimize

bubble_tristates

Choose (1) or (2) for setting your variable:

(1) Set bubble_tristates true if tristates are not in common levels. The tristates bubble up to the common top level. (2) Set bubble_tristates true if tristates are feeding an output port. The tristates bubble up to the top primary output port. This occurs if tristates are either in or not in a common level. Note: During optimization tristates automatically bubble up to level of hierarchy where all drivers become visible, or bubble up to top level if boundary. Note: bubble_tristates only bubbles boundary tristates up to a level where all drivers of a net become known. bubble_tristates does not bubble internal tristates and does not convert tristates to muxes. The conversion of tristates to muxes is controlled by tristate_map.

Note: Refer also to Utilities chapter.

Note: [-noclean] Before pushing out tristates, bubble tristates does some pre_optimization to avoid redundant ports.

Default value: TRUE

buffer_for_timing

Buffer for timing optimization.

Default value: TRUE

Related commands: optimize, optimize_timing

check_complex_ios

Use Design Rules Checker for complex ios for Actel (Act3) technology.

Default value: TRUE

Related commands: optimize
**complex_ios**

Map generic logic to complex I/O cells.

Default value: TRUE

Related commands: optimize

**constraints_save_only_multicycle**

Save multicycle constraints only, or save all advanced constraints.

Default value: TRUE

**default_bdbuf**

This is default for bidirectional buffer.

Default value: not set

**default_input_arrival**

Default arrival time at all input ports.

Default value: 0.000000

**default_input_buffer**

This is a default input buffer.

Default value: not set

**default_output_buffer**

This is a default output buffer.

Default value: not set

**default_output_required**

Default required time at all output ports.

Default value: 1073741824.000000
default_register_arrival
Default arrival time at all registers.
Default value: 0.000000

default_register_required
Default required time at all registers.
Default value: 1073741824.000000

default_tribuf
Default tristate buffer.
Default value: not set

delay_break_loops
Break combinational loops statically for timing analysis.
Default value: FALSE
Related commands: optimize, optimize_timing, report_delay

delay_weight
Indication of how much effort LeonardoSpectrum should make to minimize design delay when mapping a design; higher values indicate more effort.
Default value: -1.000000
Valid Values: 0 to 1.0, inclusive
Related commands: optimize

dont_lock_lcells
Dont lock LCELLS for Altera FLEX/MAX and Xilinx 9500 technologies.
Default value: FALSE
Related commands: optimize
**drc_const_nets**

Perform design rule checking (drc) resolving on power and ground nets.

Default value: FALSE

**edif_array_range_extraction_style**

Format of arrays in EDIF (read/write) to identify range information.

Default value: %s[%d:%d]

Related commands: read, write, auto_read

**edif_eqn_and**

Symbol to represent AND in EDIF equations.

Default value: *

Related commands: write, set_altera_eqn, set_xilinx_eqn, set_orca_eqn

**edif_eqn_not**

Symbol to represent NOT in EDIF equations.

Default value: !

Related commands: write, set_altera_eqn, set_xilinx_eqn, set_orca_eqn

**edif_eqn_not_is_prefix**

Symbol to represent NOT in EDIF equations is prefixed

Default value: FALSE

Related commands: write, set_altera_eqn, set_xilinx_eqn, set_orca_eqn

**edif_eqn_or**

Symbol to represent OR in EDIF equations

Default value: +

Related commands: write, set_altera_eqn, set_xilinx_eqn, set_orca_eqn
**edif_function_property**

Attribute name to read or write functions on lookup-table instances in EDIF.

Default value: EQN(lut_function).

Related commands: read, write, auto_read, auto_write, set_altera_eqn, set_xilinx_eqn, set_orca_eqn

**edif_write_arrays**

Allows writing arrays (busses) in EDIF output.

Default value: TRUE

Related commands: auto_write, write

**edif_write_internal_properties**

Write out all properties in EDIF, including internal ones.

Default value: FALSE

Related commands: auto_write, write

**edifin_ground_net_names**

Specify that net(s) with given name(s) are ground nets.

Default value: not set

Related commands: read

**edifin_ground_port_names**

Specify that port(s) with given name(s) are ground ports.

Default value: not set

Related commands: read, auto_read
edifin_ignore_port_names

Specify that port(s) with given name(s) are ignore port(s).

Default value: not set

Related commands: read

edifin_power_net_names

Specify that net(s) with given name(s) are power nets.

Default value: not set

Related commands: read

edifin_power_port_names

Specify that port(s) with given name(s) are power ports.

Default value: not set

Related commands: read, auto_read

edifout_ground_net_name

Special name for ground nets when edifout_power_ground_style_is_net is TRUE

Default value: GND

Related commands: write

edifout_no_prims_in_noopt

Do not write primitives in noopts.

Default value: False

Related commands: write
edifout_power_ground_style_is_net

Writes out power and ground as undriven nets with special name.
Default value: FALSE
Related commands: write

edifout_power_net_name

Special name for power nets when edifout_power_ground_style_is_net is TRUE.
Default value: VCC
Related commands: write

edifout_write_noopted_contents

Write contents for noopted views.
Default value: TRUE
Related commands: write

Note: Refer to auto_write command in the Utilities chapter for information on using this variable after running the decompose_luts command.

enable_dff_map_optimize

Infer clock-enables from random logic.
Default value: FALSE
Related commands: optimize

encoding

Specify default enumeration type encoding style.
Default value: auto
Valid Values: binary, onehot, twohot, gray, random, auto
Related commands: analyze, read
**exclude_gates**

Specify gates to be excluded from target technology library.

Default value: not set

Related commands: `load_library`

**extract_cin_cout**

Enable automatic detection of carry-in/carry-out.

Default value: TRUE

Related commands: `pre_optimize`

**extract_counter**

Enable automatic extraction of counters from generic logic.

Default value: TRUE

Related commands: `pre_optimize`

**extract_decoder**

Enable automatic extraction of decoders from generic logic.

Default value: TRUE

Related commands: `pre_optimize`

**extract_ram**

Enable automatic extraction of RAMs from generic logic.

Default value: TRUE

Related commands: `pre_optimize`
**extract_reduction_ops**

Enable automatic detection of reduction operators.

Default value: TRUE

Related commands: pre_optimize

**extract_rom**

Enable automatic extraction of ROMs from generic logic.

Default value: TRUE

Related commands: pre_optimize

**flex_auto_implement_in_eab**

Set for Altera FLEX place and route.

Default value: FALSE

**force_user_load_values**

Use the user values for max capacitance and fanout loads, ignore library values.

Default value: FALSE

Related commands: balance_loads, optimize

**fsm_do-collapse**

Collapse the next state logic before optimization and mapping.

Default value: FALSE
fsm_flow

By specifying the FSM flow we take:

- **best_optd**: optimized/mapped/unmapped network with best encoding
- **best_orig**: original primary network with best encoding
- **bind_early**: optimized/mapped/noopted network with best encoding
- **bind_late**: undefined currently
- **onehot_optd**: optimized/mapped onehot encode network (for debug)
- **onehot_orig**: original primary network with onehot encoding
- **binary_optd**: optimized/mapped binary encoded network (for debug)
- **binary_orig**: original primary network with binary encoding
- **nootp**: extract only dont optimize

full_case

Interpret Verilog case statements as fully specified.

Default value: FALSE

Related commands: analyze, read

gate_sizing

Perform gate sizing for ASIC technologies.

Default value: TRUE

Related commands: load_library
**generate_timespec_from_inputs**

Generate timespec from input pins to registers or output pins.

Default value: FALSE

Related commands: generate_timespec

**global_sr**

Define an active high signal name as global set or reset for Xilinx or ORCA technologies. Only works for designs without hierarchy.

Default value: not set

Related commands: optimize

**hdl_array_name_style**

Bit name style for array type objects.

Default value: %s(%d)

Related commands: elaborate, read

**hdl_array_separator_style**

Dimension separator in bit name style for multi-dimensional array type objects.

Default value: ) ( 

Related commands: elaborate, read

**hdl_integer_name_style**

Bit name style for integer type objects (index 0 is LSB).

Default value: %s(%d)

Related commands: elaborate, read
**hd1_record_name_style**

Bit name style for record type objects.
Default value: %s_%s
Related commands: elaborate, read

**include_gates**

Do not exclude gates that are predefined in target library (ORCA technologies only).
Default value: not set
Related commands: load_library

**infer_gsr**

Detect the global set or reset signal for Xilinx and ORCA architectures. Only works for designs without hierarchy.
Default value: TRUE
Related commands: optimize

**input2output**

Constrain paths between input ports and output ports.
Default: 1073741824.000000
Example: set input2output 10

**input2register**

Constraint paths between input ports and register. **Note:** Use input2register to constrain sub-block boundary logic to one-half of the clock period (as defined by register2register variable).
Default: 1073741824.000000
Example: set input2register 10
**insert_global_bufs**

Use global buffers for clocks and other global signals (Xilinx, Actel technologies only).

Default value: TRUE

Related commands: optimize

**inversion_prefix**

Prefix for inverted nets.

Default value: NOT

**keep_flattened_views**

Keep flattened views even if the views are no longer referenced.

Default value: FALSE

Related commands: write

**lgen_array_name_style**

Naming style for array type objects in Lgen library cells.

Default value: %s(%d)

Related commands: load_library

**list_design_object_separator**

Separator string used to find objects in a design name (library, cell, view, etc).

Default value: .

Related commands: list_design

**load_library_file_extension**

Technology library filename extension.

Default value: syn

Related commands: load_library
lpm_remove_unused_ports

Remove unused ports from instantiated modules when reading VHDL or Verilog.

Default: TRUE

lut_buffering

Insert buffers (LUTs) on high fanout nets for better routeability.

Default value: FALSE
Related commands: write

lut_cell_name

Prefix for LUT cells created by LUT decomposition.

Default value: lut_cell
Related commands: decompose_luts

lut_map

Do lookup table mapping for LUT-based FPGA architectures (Altera Flex, ORCA, Xilinx technologies only).

Default value: TRUE
Related commands: optimize

lut_max_fanout

Specify net fanout for LUT technologies (Xilinx, ORCA, Altera FLEX). LeonardoSpectrum attempts to maintain reasonable fanouts by replicating the driver which results in net splitting. If replication is not possible, then the signal is buffered. This may make the wire slower by adding intrinsic delays. A Max Fanout window is on the Advanced Technology FlowTab. Refer also to Attributes chapter for syntax for the lut_max_fanout attribute.

Default value: 0

set lut_max_fanout <integer>
**map_fanin_limit**

If limit is not 0 then specify upper boundary for fanin to a function.

Default value: 4

**map_muxcy**

Map to MUXCY cells from random logic for Xilinx Virtex. LeonardoSpectrum maps to MUXCY when possible from random logic to implement priority encoders in carry chains. This variable controls mapping. Set this variable true for Xilinx M1 2.1 and higher versions.

```plaintext
set map_muxcy false
set map_muxcy true (Xilinx M1 2.1 and higher)
```

Default value: FALSE

**map_muxf5**

Map to MUXf5 for Xilinx XCV and Virtex technologies. For odd sized muxes (non 2-power input size muxes), more MUXf5s and MUXf6s are utilized.

Default value: TRUE

Related commands: optimize

**map_muxf6**

Map to MUXf6 for Xilinx XCV and Virtex technologies. For odd sized muxes (non 2-power input size muxes), more MUXf5s and MUXf6s are utilized.

Default value: TRUE

Related commands: optimize

**map_pfu_gates**

Map pfu gates for ORCA 2TA/2CA/3C/3T.

Default value: TRUE
**map_sop_muxcy**

Map to muxcy for sum-of-products for Xilinx Virtex technology.

Default value: FALSE

**map_sync_reg**

Map to synchronize S/R registers.

Default value: TRUE

**max_cap_load**

Override default max_cap_load if specified in the library. Refer to library for values.

Default value: 0.000000

Related commands: balance_loads, optimize, optimize_timing

Example: set max_cap_load 4

**max_fanin**

Define upper bound for number of fanins to a function (0 is no limit).

Default value: 0

Related commands: optimize

**max_fanout_load**

Override default max_fanout_load if specified in the library. Refer to library for values.

Default value: 0.000000

Related commands: balance_loads, optimize, optimize_timing

Example: set max_fanout_load 16
**max_pt**

Define maximum number of product terms in a function (0 is no limit).

Default value: 0

Related commands: optimize

**max_transition**

Override default max_transition if specified in the library. Refer to library for values.

Default value: 0.000000

Related commands: balance_loads, optimize, optimize_timing

Example: set max_transition 1.2

**maxarea**

Maximum area allowed in one design.

Default value: 1073741824.000000

Related commands: optimize, optimize_timing

**maxdly**

Maximum delay allowed in one design.

Default value: 1073741824.000000

Related commands: optimize -delay, optimize_timing

**mem_minimum_size**

Minimum memory size for mapping to lpm_rom for Altera FLEX 10.

Default value: 256

By default the minimum size of the ROMs detected in FLEX 10 is 256. Set this variable to detect ROMs smaller in size:

set mem_minimum_size 64

set mem_minimum_size 0 (Detect ROMs of all sizes.)
**modgen_select**

Default mode for modgen resolving.

Default value: `auto`

Valid Values: `auto`, `smallest`, `small`, `fast`, `fastest`

**move_files_on_cwd_change**

Move list and session file on new current working directory (cwd) when changing cwd.

Default: `FALSE`

**multi_driver_drc_resolving**

Perform DRC resolving on multidriver net.

Default value: `TRUE`

**names_collision_extension**

Name extension to be used when names collide in renaming process.

Default: `_rename`

**nl_use_approx**

Use approximation for nonlinear delay computation during mapping only (ASICs only).

Default value: `TRUE`

Related commands: `optimize`, `optimize_timing`, `report_delay`

**no_boundary_optimization**

Disable `-boundary optimize` which is executed during optimization. Skip boundary optimization.

Default value: `FALSE`

Related command: `optimize`
**no_sequential_cell_replication**
Do not replicate sequential cell.
Default value: FALSE

**nowire_table**
Do not use a wire table during delay calculations.
Default value: FALSE
Related commands: optimize, optimize_timing, report_delay

**old_style_session_file**
Create old 4.x style history file.
Default value: FALSE

**operating_condition**
Specify operating conditions for timing computations.
Default: not set

**optimize_clock_enable**
Enable automatic optimization of clock enable.
Default value: FALSE

**optimize_clock_enable_support_limit**
Fanin limit for optimization of clock enable.
Default value: 40

**optimize_cpu_limit**
CPU limit for optimize command, in seconds (0 is no limit).
Default value: 0 (no limit)
**optimize_drc_resolving**

Enables DRC (design rule checking) resolving during optimization by default.

Default value: TRUE

For example, if you are using script for an ASIC design, then you can

```
set optimize_drc_resolving false
```

to disable this variable.

**Note:** You must run the `balance_loads` command at the end of your design run to ensure that the final design meets the design rule checking (DRC).

**Note:** DRC resolving may require more runtime. However, DRC resolving can improve the initial timing estimation, and can prevent heavily loaded nets.

**optimize_sequential_cell**

This is a global ASIC variable. You can set this variable to FALSE to keep DFF for equivalent checking. Improves verifying state machine.

```
set optimize_sequential_cell true
```

Default value: TRUE

**optimize_timing_cpu_limit**

CPU limit for `optimize_timing` command, in seconds (0 is no limit).

Default value: 0 (no limit)

**package**

Indicate a specific package type for the output design.

Default value: not set

Related commands: optimize

**parallel_case**

Interpret Verilog `CASE` statements as parallel.

Default value: FALSE

Related commands: analyze, read
part

Indicate a specific part for the output design.
Default value: not set
Related commands: optimize

pref_only_primary_ios

Only write ORCA preference file timing constraints for primary I/Os.
Default value: FALSE

pref_type_of_port_object

Type of object on which to write ORCA preference file port timing constraints.
Default value: net
Valid values: instance, port, net
Related commands: write

prepend_dff_inst_name

Prepend the string to a dff name driving user defined net.
Default value: reg_
Related commands: write

prepend_io_inst_name

Prepend the string to IO buffer instance names.
Default value: IO

prepend_latch_inst_name

Prepend the string to a latch name driving user defined net.
Default value: lat_
Related commands: write
prepend_tri_inst_name

Prepend the string to a tri name driving user defined net.
Default value: tri_
Related commands: write

preserve_dangling_net

Create ports to unconnected nets when reading XNF file.
Default value: FALSE

process

Use specified process or speed-grade from target technology to compute delay derating factors. Refer to library for values.
Default value: not set (2)
Related commands: load_library
Example: set process typical

process_pragma

Process VHDL and Verilog pragmas.
Default value: TRUE

propagate_clock_delay

If FALSE, then use ideal clock. If TRUE then use clock delay in delay calculation.
Default value: FALSE

propagate_clock_delay

If FALSE, then use ideal clock. If TRUE then use clock delay in delay calculation.
Default value: FALSE
pterm_max_fanin

If not 0 then specify upper boundary for fanin to a function for pterm based technology - Altera.

Default value: 0

register2output

Constraint paths between registers and output ports. **Note:** set the register2output and input2register variables to one-half of the clock period to ensure that the boundary logic of subblocks meets timing when combined in a top-level design.

Default value: 1073741824.000000

Example: `set register2output 10`

register2register

Constraint paths between registers.

Default value: 1073741824.000000

Example: `set register2register 20`

replicate_for_complex_io_mapping

Replicate latches/flip flops for complex IO mapping.

Default value: FALSE

replicate_logic_for_drc

Enable the use of logic replication for design rule checker (drc).

Default value: TRUE

replicate_logic_for_timing

Replicate logic for timing optimization.

Default value: TRUE

Related commands: `optimize`, `optimize_timing`
**report_area_format_style**

Style (precision) of reporting area numbers.

Default value: %6.0f

Related commands: report_area

**report_delay_analysis_mode**

Delay analysis mode. Note: Perform timing optimization and timing analysis with this variable.

Default value: maximum

Valid Values: maximum, minimum, both

Related commands: report_delay

Example: set report_delay_analysis_mode minimum

Note: You can perform timing optimization and timing analysis when the variable is set to the default minimum.

Example: set report_delay_analysis_mode maximum

Note: You can perform worst-case hold time analysis when the variable is set to maximum.

**report_delay_arrival_threshold**

Arrival time threshold for delay report, in nanoseconds.

Default value: 0.000000

Related commands: report_delay

**report_delay_detail**

Amount of detail in delay report.

Default value: full

Valid Values: full, short

Related commands: report_delay
**report_delay_format_style**

Style (precision) of reporting delay numbers.

Default value: %4.2f

Related commands: report_delay

**report_delay_slack_threshold**

Slack threshold for delay report, in nanoseconds.

Default value: 0.000000

Related commands: report_delay

**resolve_mux_stat**

Decision table for modgen resolving vs default resolving of muxes.

Default value: not set

Related commands: resolve_modgen

**resource_sharing**

Enable resource sharing. Allows you to reduce number of certain devices.

Default value: TRUE

**resource_sharing_through_pattern**

Enables resource sharing through pattern matching.

Default value: TRUE

Related commands: write

**restructure_for_timing**

Restructure combinatorial logic for timing optimization.

Default value: TRUE
sdf_hier_separator

Separator for hierarchical names in the SDF writer.
Default value: /
Related commands: write

sdf_hierarchical_names

Treat all SDF names with divider character as hierarchical.
Default value: TRUE
Related commands: read

sdf_names_style

Rename rules for SDF writer
Default value: vhdl
Valid Values: verilog, vhdl, none
Related commands: write

sdf_read_suppress_warnings

Suppress cell names mismatch warnings.
Default value: TRUE

sdf_type

Define delay derating for SDF reading and writing.
Default value: maximum
Valid Values: minimum, typical, maximum
Related commands: read
**sdf_write_flat_netlist**

Write SDF for a flat netlist. This variable constrains the SDF writer when set to TRUE.

Default value: FALSE

**Example:** The SDF writer is controlled by the `sdf_write_flat_netlist` Tcl variable. Set this variable to TRUE. An example set of **Level 3** commands may be:

```tcl
set sdf_write_flat_netlist TRUE
ungroup -all /*need to flatten the netlist*/
write design.vhd /*write out flat VHDL or Verilog*/
write design.sdf /*write out SDF*/
```

**state_table_threshold**

Do not print a state assignment table for enumerated types with more values than this (40).

Default value: 40

**sweep_in_fe**

Remove unused logic, propagate constants, merge common logic, etc. during elaboration.

Default value: TRUE

**sweep_unused_user_cells**

Remove unused instantiated cells.

Default value: TRUE

**temp**

Indicate temperature (in celsius, centigrade) to compute delay derating factors. Refer to the library for values.

Default value: not set

**Example:** `set temp 80`
transformations

Allow transforming set/reset on dffs and latches to match target technology.
Default value: TRUE

tristate_map

Allow conversion of internal tri-states to combinational logic to match target technology. **Note:** The conversion of tristates to muxes is controlled by tristate_map. Refer also to bubble_tristates.
Default value: FALSE

ungroup_hier_separator

Separator for hierarchical names.
Default value: TRUE
Related commands: ungroup

use_assign_for_vcc_gnd

Use assign statement or supply [01] nets for power ground nets.
Default value: TRUE

use_dffenable

Infer clock-enable from HDLs. When set to TRUE LeonardoSpectrum maps to any existing flip-flops with clock enable. When set to FALSE, LeonardoSpectrum is prevented from doing this automatic optimization.
Default value: TRUE
Related commands: read

use_f5map

Enable mapping to F5MAP (for Xilinx XC5200 technology only).
Default value: FALSE
Related commands: optimize
use_f6_lut

Map to 6 input LUTs during ORCA LUT mapping.
Default value: FALSE
Related commands: optimize

use_qclk_bufs

Use quadrant clocks for Actel 3200dx architecture.
Default value: FALSE
Related commands: optimize

userVerbose

Use this boolean global variable to print out optimization messages.
Default value: FALSE

verilog_max_line_length=80

Define maximum line length when writing Verilog.
Default value: 80

verilog_parameter_to_attribute

Move the parameter values to the view (useful for LPM instantiation).
Default value: TRUE
Related commands: read

verilog_read_ignore_input_assign_errors

Ignore assignment of input to bidir.
Default value: FALSE
**verilog_write_arrays**

Allows writing arrays (busses) in Verilog output.

Default value: TRUE

Related commands: write, auto_write

**verilog_write_pwr_gnd_cells**

Write technology power/ground cells instead of assign/supply statements.

Default value: TRUE

**vhdl_87**

Use VHDL'87 style syntax/semantics instead of VHDL'93 for reading VHDL.

Default value: FALSE

Related commands: analyze, read

**vhdl_generic_to_attribute**

Move the generic values to the view (useful for LPM instantiation).

Default value: TRUE

Related commands: read

**vhdl_write_87**

Use VHDL'87 style syntax/semantics instead of VHDL'93 for writing VHDL.

Default value: FALSE

Related commands: write

**vhdl_write_arrays**

Allows writing arrays (busses) in VHDL output.

Default value: TRUE

Related commands: write, auto_write
vhdl_write_bit

  type for bit used in VHDL writer.
  Default value: std_logic
  Valid Values: any user-defined string
  Related commands: write

vhdl_write_bit_vector

  This is the type for bit vectors used in VHDL writer.
  Default value: std_logic_vector
  Related commands: write

vhdl_write_component_package

  Write components in package instead of in line with architecture.
  Default: TRUE

vhdl_write_component_package_name

  Name of package containing component for VHDL writing.
  Default: components

vhdl_write_configuration

  Write VHDL configuration.
  Default: TRUE

vhdl_write_inst_uppercase

  Write VHDL instance names in uppercase.
  Default: FALSE
vhdl_write_port_uppercase

Write VHDL port names in uppercase.
Default: FALSE

vhdl_write_pwr_gnd_cells

When this variable is set to default TRUE, the power and ground cells in the logic network are written in the netlist. When the variable vhdl_write_pwr_gnd_cells is set to FALSE, then the power and ground cells are written as VHDL assignment statements instead.

set vhdl_write_pwr_gnd_cells TRUE
Default: TRUE

vhdl_write_signal_uppercase

Write VHDL signal names in uppercase.
Default: FALSE

vhdl_write_use_packages

Define which packages to include for each entity.
Default value: library IEEE, EXEMPLAR; use IEEE.STD_LOGIC_1164.all; use EXEMPLAR.EXEMPLAR_1164.all;

viewlogic_vhdl

Read ViewLogic's pack1076 built-in package as standard.
Default value: FALSE
Related commands: analyze, read
virtex_infer_gsr

Xilinx for Virtex: GSR processing is not recommended. The variable is FALSE by default since GSR lines can be slower than local asynchronous S/R signals. This variable allows LeonardoSpectrum to do “GSR” processing for Virtex. Exemplar recommends using GSR processing sparingly.

set virtex_infer_gsr false

Default: FALSE

virtex_map_iob_registers

Map to IOB registers for Xilinx Virtex.

Default value: FALSE

virtex_map_srl

Map to shift register luts (SRL). An array of flip flops is mapped to SRLs. This SRL mapping provides an improvement in area for a design with similar structures. SRL is used in static addressing mode. This means that the address lines of SRL are tied to a constant. LeonardoSpectrum maps to two flavors of SRL: SRL16 and SRL16E (with clock enable). Refer to Synthesis & Technology guide, Xilinx Virtex chapter, for more information.

Default value: TRUE

set virtex_map_srl false

virtex_map_srl_pack

Pack shift register luts (SRL). This variable packs SRL into a single slice. Refer to Synthesis & Technology guide, Xilinx Virtex chapter, for more information.

Default value: TRUE

set virtex_map_srl_pack false

virtex_mapWide_clusters

Map to wide clusters for Virtex.

Default value: TRUE

Related commands: write
**voltage**

Specify voltage (in V) to compute delay derating factors. Refer to the library for values.

Default value: not set

Related commands: load_library

Example: set voltage 5.0

---

**wire_load_library**

Name of library the present design is mapped to.

Default NIL (char)

---

**wire_load_mode**

Mode to compute wire loads: top (default), segmented (enclosed).

Default: top

---

**wire_table**

Use named wire table from target library for delay calculations. **Note:** Set the wire load model to reflect the gate count of the sub-block. The wire load model determines the capacitance value applied to all nets.

Default value: not set

Related commands: optimize, optimize_timing, report_delay

Example: set wire_table cg61_50000area

Example: set wire_table worst

---

**wire_tree**

Specify interconnect wire tree model to use for delay calculations. Default value: not set

Valid Values: best, balanced, worst

Related commands: optimize, optimize_timing, report_delay
**write_clb_packing**

Print CLB packing (HBLKNM) information, if available, in XNF/EDIF files.

Default value: FALSE

Related commands: write

**write_lut_binding**

Print LUT binding (HMAP/FMAP) information, if available, in XNF/EDIF files.

Default value: TRUE

Related commands: write

**write_xrf_file**

Write cross reference file.

Default value: TRUE

**x_probe**

Enable cross probe for schematic viewer.

Default value: TRUE

**x_probe_autocopy**

During optimization, auto save a copy of RTL view for x_probe.

Default value: TRUE

**xdb_write_version**

For LeonardoSpectrum v1998.x to read xdb written by the v1999.1 version, set this variable to 1998.x while in v1999.1 before writing.

Default value: 1999.1

**Note**: This variable is available only in v1999.x. This variable allows you to write a project in v1999.x and then read the project in v1998.x.

```plaintext
set xdb_write_version v1998.x
```
**xi_write_init_on_luts**

Xilinx Virtex and Virtex-like architecture: VirtexE and Spartan2: LeonardoSpectrum writes out post synthesis VHDL and Verilog netlist with LUT and INIT property for the LUT. In the default flow, LeonardoSpectrum still writes out equations instead of LUT.

```bash
set xi_write_init_on_luts TRUE
```

By using this feature, you take advantage of the LUT model in Xilinx “unisim” library which will speed up the post synthesis simulation considerably.

Default value: FALSE

**xlx_fast_slew**

Specifies setting outputs to FAST, if slew rate attributes do not already exist. All Xilinx XC4000, 4000E, and 5200 technologies except XC4000H. TRUE sets outputs to FAST if no slew rate attributes already exist.

Default value: TRUE

Related commands: optimize

**xlx_preserve_gsr**

Preserve the global set signal for Xilinx when reading XNF files.

Default value: FALSE

Related commands: read

**xlx_preserve_gts**

Preserve the global tri-state signal for Xilinx when reading XNF files.

Default value: FALSE

Related commands: read

**xlx_preserve_pins**

Preserve Xilinx pin locations when doing Xilinx optimization.

Default value: TRUE

Related commands: read
**xor_decomp**

Do XOR decomposition for Altera MAX and Xilinx 9500 technologies.

Default value: TRUE

Related commands: optimize
Utilities

This chapter contains:

- Utility Scripts
- Aliases

Utility Scripts

The following Tcl scripts are available:

**all_clocks**

List all clocks. Usage: `all_clocks [design][short]
Print only short names not full path to objects.

**all_inputs**

List all input ports. Usage: `all_inputs [design][short]
Print only short names not full path to objects.

**all_outputs**

List all output ports. Usage: `all_outputs [design][short]
Print only short names not full path to objects.

**all_registers**

List all registers. Usage: `all_registers [design][short]
Print only short names not full path to objects.
all_selected

List all selected objects. Usage: all_selected [<port>] [<net>] [<instance>]
[ -direction <string>] List port, net, object for ports only INOUT or IN OUT.

auto_read

Automatically does needed processing for technology-specific netlists in addition to the read command, including setting of variable values.

Alterna Note: auto_read replaces the previously available read_altera script.

auto_write

Automatically does needed processing for your target technology in addition to the write command, including setting of variable values and calling generate_timespec, decompose_luts and apply_rename_rules, where applicable. Note: Refer also to decompose_luts and unmap in the Commands chapter.

Alterna Note: auto_write replaces the previously available write_altera script.

AMD Vantis Note: When targeting Vantis devices use auto_write to generate EDIF.

Attribute Removal Note: the following applies to auto_write, decompose_luts commands and to noopt/dont_touch attributes:

CAUTION: If you want to use the auto_write command for a bottom-up hierarchical design, then consider the following:

- After you synthesize the lower modules of your design the first time, you can prevent a second optimization of these modules by applying noopt and/or dont_touch attribute to each module.
- However, before using auto_write command, you must manually remove the dont_touch and noopt attributes. Otherwise, the output FPGA netlist is not valid in most cases.
- After you remove the attributes, then run the decompose_luts command. This is important to technologies with look up tables. The decompose_luts command skips over modules with dont_touch/noopt attributes attached to modules. The FPGA netlist is not valid when the applied noopt/dont_touch attributes prevent preprocess of the decompose_luts command from running.
- After running decompose_luts you can set the edifout_write_noopted_contents variable TRUE to write the contents of your noopt/dont_touch modules into an EDIF file. If you do not run this variable, then you may have some black boxes in your netlist that the P&R tool cannot read.
blackbox

The usage for the blackbox script is:

    blackbox <instance_name | view_name>

You must ensure that view_name is hierarchical. The specified view_name is written as a blackbox in the output netlist.

Note: view_name may also be instance_name.

bubble_tristates

Moves tristates up in the design hierarchy, allowing you to optimize designs with buried tristate I/Os without flattening. Refer also to Variables chapter.

Choose (1) or (2) for setting your variable:

(1) Set bubble_tristates true if tristates are not in common levels. The tristates bubble up to the common top level. (2) Set bubble_tristates true if tristates are feeding an output port. The tristates bubble up to the top primary output port. This occurs if tristates are either in or not in a common level. (3) Set bubble_tristates false to disable. Note: During optimization tristates automatically bubble up to level of hierarchy where all drivers become visible, or bubble up to top level of boundary. Note: bubble_tristates only bubbles boundary tristates up to a level where all drivers of a net becomes known. bubble_tristates does not bubble internal tristates and does not convert tristates to muxes. The conversion of tristates to muxes is controlled by tristate_map variable.

Note: Refer also to tristate_map in Variables chapter.

Default value: TRUE

clean_all

Removes all objects in database
usage: clean_all

dfs

Performs a depth-first search and returns a list of views.

extract_best_passes

Finds the best pass number for each view that was optimized in the last call to the optimize command.
**fix_backanno**
Modifies VHDL files produced by FPGA Place & Route tools by commenting out simulation constructs which LeonardoSpectrum cannot parse.

**getlist**
Displays the contents of a technology library.

**global_set_attribute**
Sets attributes on nets throughout the design hierarchy. This is useful for the clock_cycle constraint.

**global_remove_attribute**
Removes attributes on nets throughout the design hierarchy.

**lo2up**
Renames all lower case objects with upper case.

**move_nodelay**
Moves the attribute NODELAY from the input port to the input flip-flop.
Place and Route

usage: place_and_route <edif input file>

-target <string> target technology
[-gui] GUI instead of interactive shell mode
[-exe_path <string>] path to place and route tools
[-part <string>] [-speed_grade <string>]
[-ba_format <string>] Back annotated netlist format
VHDL|Verilog|EDIF
[-n] Show, but do not execute commands

The M1 commands apply to all M1 supported Xilinx technologies except xi95

[-ml_pr_standard] M1, standard place and route effort level
[-ml_pr_high] M1, high place and route effort level
[-ml_functional_sim] M1, only produce functional sim netlist
[-ml_preroute_timing] M1, only produce pre-route timing estimates
[-ml_pack_iobs] M1, pack FFs & latches into IOBs
[-ml_no_bits] M1, don't produce bit file after place and route
[-ml_no_backanno] M1, don't produce timing sim files after place and route
[-ml_no_ngm] M1, don't use ngm file with ngdanno
[-ml_bitgen_cmd_file <file>] M1, use -f <command file> option with bitgen
[-max_acf_only] Only generate acf, don't run maxplus2 compiler
[-max_no_acf] Suppress generation of acf-file
[-max_ta_delay] MAX+PLUS II shows input to output delays
[-max_ta_setup] MAX+PLUS II shows setup hold matrix
[-max_ta_reg] MAX+PLUS II shows register performance
[-max_area] [-max_delay] MAX+PLUS II optimizes for minimum delay or area (default)
[-max_auto_fast_io] MAX+PLUS II place and route option
[-max_auto_register_packing] MAX+PLUS II place and route option
[-max_auto_implement_in_eab] MAX+PLUS II place and route option
**pop_design**

Pops the specified number of levels out of the design stack.

**print_design_stack**

Prints out the design stack for informational purposes. This is useful with the push_design and pop_design scripts.

**push_design**

Pushes into the design stack while preserving your ability to pop back to your current position in the stack.

**puts_log**

Puts a string to a stdout and log file. Usage: `puts_log <string> [-nonewline]`  
Do not add a new line: `[-nonewline]`.

**read_constraints**


**recompose_flex**

Recompose LUTs from post place and route.  
Usage: `recompose_luts`  
- `source <string>`, (source technology)

**restore_project_script**

Usage: `restore_project_script [<filename>].`  
[-no_design]  
Do not restore design: `[-no_design]`.

**same_tech_noopt**

Usage: `set same_tech_noopt`  
Example: `nomap;`
function ();
area = 5000
set same_tech_noopt

Note:
nomap: Set the same_tech_noopt variable to allow component instantiation of the RAM cell.
function (): The function parameter allows you to define a particular function for a gate.
area: Specify the RAM cell area in gates.

same_tech_dont_touch

Usage: set same_tech_dont_touch

save_project_script

Usage: save_project_script [<filename>].
[-no_design]
Do not restore design: [-no_design].

set_altera_eqn

This script replaces the following Altera variables:
edif_eqn_and, edif_eqn_not, edif_eqn_not_is_prefix, edif_eqn_or, edif_function_property.

set_xilinx_eqn

This script replaces the following Xilinx variables:
edif_eqn_and, edif_eqn_not, edif_eqn_not_is_prefix, edif_eqn_or, edif_function_property.

show_var_settings

Display all variable settings made in a session.

up_design

Traverses the hierarchy of your design by moving up one or more levels of hierarchy.
**view_schematic**
Displays a schematic view of the current design (default) or of the specified design.

usage: view_schematic [design]

Specify symbol library to use: [-symlib <string>]

Display the RTL version of the design, if any: [rtl]

**warp_vhdl**
Aliased to "uplevel #0 set vhdl_write_use_packages "library ieee, work;
use ieee.std_logic_1164.all;\nuse WORK.EXEMPLAR_GATES.ALL;"

**xmplr_exec**
A version of executable supporting nonblocking read.

usage: xmplr_exec command

**xmplr_socket_client**
Command to open a client socket.

[host <string>]
**Aliases**

This alias list is also available in the interactive command line shell.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Aliased to</th>
</tr>
</thead>
<tbody>
<tr>
<td>al</td>
<td>list_attributes</td>
</tr>
<tr>
<td>arrival_time</td>
<td>ARRIVAL_TIME</td>
</tr>
<tr>
<td>buffer_sig</td>
<td>BUFFER_SIG</td>
</tr>
<tr>
<td>clock_cycle</td>
<td>CLOCK_CYCLE</td>
</tr>
<tr>
<td>clock_offset</td>
<td>CLOCK_OFFSET</td>
</tr>
<tr>
<td>clock_cycle</td>
<td>CLOCK_CYCLE</td>
</tr>
<tr>
<td>connect_timing_arc</td>
<td>CONNECT_TIMING_ARC</td>
</tr>
<tr>
<td>dc</td>
<td>present_design</td>
</tr>
<tr>
<td>dl</td>
<td>list_design</td>
</tr>
<tr>
<td>disconnect_timing_arc</td>
<td>DISCONNECT_TIMING_ARC</td>
</tr>
<tr>
<td>h</td>
<td>history</td>
</tr>
<tr>
<td>implement_in_pterm</td>
<td>IMPLEMENT_IN_PTERM</td>
</tr>
<tr>
<td>input_drive</td>
<td>INPUT_DRIVE</td>
</tr>
<tr>
<td>input_max_fanout</td>
<td>INPUT_MAX_FANOUT</td>
</tr>
<tr>
<td>input_max_load</td>
<td>INPUT_MAX_LOAD</td>
</tr>
<tr>
<td>max_trans_fall</td>
<td>MAX_TRANS_FALL</td>
</tr>
<tr>
<td>max_trans_rise</td>
<td>MAX_TRANS_RISE</td>
</tr>
<tr>
<td>modgen_read</td>
<td>MODGEN_READ</td>
</tr>
<tr>
<td>modgen_resolve</td>
<td>MODGEN_RESOLVE</td>
</tr>
<tr>
<td>nobuff</td>
<td>NOBUFF</td>
</tr>
<tr>
<td>noopt</td>
<td>NOOPT</td>
</tr>
<tr>
<td>output_fanout</td>
<td>OUTPUT_FANOUT</td>
</tr>
<tr>
<td>output_load</td>
<td>OUTPUT_LOAD</td>
</tr>
<tr>
<td>pad</td>
<td>PAD</td>
</tr>
<tr>
<td>pd</td>
<td>present_design</td>
</tr>
<tr>
<td>Command</td>
<td>Aliased To</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>pin_number</td>
<td>PIN_NUMBER</td>
</tr>
<tr>
<td>preserve_signal</td>
<td>PRESERVE_SIGNAL</td>
</tr>
<tr>
<td>pulldown</td>
<td>PULLDOWN</td>
</tr>
<tr>
<td>pullup</td>
<td>PULLUP</td>
</tr>
<tr>
<td>pulse_width</td>
<td>PULSE_WIDTH</td>
</tr>
<tr>
<td>quit</td>
<td>EXIT</td>
</tr>
<tr>
<td>required_time</td>
<td>REQUIRED_TIME</td>
</tr>
<tr>
<td>run_opt</td>
<td>optimize</td>
</tr>
<tr>
<td>sweep</td>
<td>pre_optimize</td>
</tr>
<tr>
<td>synthesis</td>
<td>elaborate</td>
</tr>
<tr>
<td>unimplement_in_pterm</td>
<td>UNIMPLEMENT_IN_PTERM</td>
</tr>
<tr>
<td>uniquify</td>
<td>UNFOLD</td>
</tr>
<tr>
<td>unnoopt</td>
<td>UNNOOPT</td>
</tr>
<tr>
<td>undont_touch</td>
<td>UNDONT_TOUCH</td>
</tr>
</tbody>
</table>
Attributes

An attribute is attached to HDL objects or predefined data about HDL objects. The syntax and methods for applying attributes to your design are described in this chapter. This chapter is divided as follows:

- How to Set Attributes
- Alphabetical List of Attributes

How to Set Attributes

This section provides examples of four ways to set attributes.

- (1) VHDL Syntax
- (2) Verilog Syntax
- (3) Interactive Command Line Shell Tcl Syntax
- (4) Attributes for the GUI Constraint File Syntax
(1) **VHDL Syntax**

Use the following syntax for VHDL attributes:

```vhdl
attribute <attribute_name> : <attribute_type> ;

Note: This is a declaration of the attribute.
```

```vhdl
attribute <attribute_name> of <object_name> : component is <attribute_value>

Note: Set on component which is corresponding to view.
```

```vhdl
attribute <attribute_name> of <object_name> : label is <attribute_value>

Note: Set on a label which is corresponding to instance.
```

**VHDL Example:**

HDL Synthesis guide, Chapter 3, shows the assignment of the `buffer_sig` attribute:

```vhdl
library exemplar;
use exemplar.exemplar.all; -- Include the 'exemplar' package
entity example is
    port ( inp, clk : in std_logic;
            outp : out std_logic;
            inoutp : inout std_logic;
    );
attribute buffer_sig : string;
attribute buffer_sig of clk:signal is "CLOCK BUFFER";
end example;
```
(2) Verilog Syntax

Use the following directive for Verilog attributes.

```verilog
//exemplar attribute <object_name> <attribute_name> <attribute_value>
```

Verilog Example:

This example is for assigning the attribute `modgen_sel`. Refer to the HDL Synthesis guide, Chapter 7.

```verilog
//example
module expr (a, b, c, out1, out2);
input [15:0] a, b, c;
output [15:0] out1, out2;

    assign out1 = a + b;
    assign out2 = b + c;

// exemplar attribute out1 modgen_sel fastest
endmodule
```

(3) Interactive Command Line Shell Tcl Syntax

Use the following Tcl syntax, for example, if you do not want to modify your Verilog or VHDL code:

```tcl
set_attribute -<obj_type> <obj_name> -name <attribute_name> -value <attribute_value>
```

Interactive Command Line Shell Example:

```tcl
set_attribute -instance abc -name noopt -type boolean -value TRUE
remove_attribute -instance abc -name noopt

where: -type is [instance, net, port]
```
(4) Constraint File Syntax for the GUI

Use the following syntax for your constraint file (constraint_filename.ctr) editor report on the GUI:

<constraint_name> <value> <port or signal_name>

Constraint File Syntax:

```
buffer_sig clkbuf clk1
Note: Connects signal clk1 to the input of the external clock buffer (clkbuf).

buffer_sig clkint rstn
Note: Connects signal rstn (reset signal) to input of internal clock buffer (clkint).
```

Alphabetical List of Attributes

The following list of attributes are written with an abbreviated syntax that is accepted by the LeonardoSpectrum constraint file on the constraint editor report. In addition, the Verilog, VHDL, and interactive command line shell syntax can be used as shown for some attributes. Note: These attributes are also listed in the User’s Guide, Constraint chapter. NOT all attributes listed in this chapter are available for the constraint file. Note: Attributes apply to ports, while variables apply globally. Attributes take precedence over variables.

**arrival_time**

```
arrival_time <delay_value> <input_port_list>
```

**Interactive Command Line Shell Syntax:**

```
set_attribute -port inp(1) -name arrival_time -value 10
```

**Constraint File Syntax:**

```
arrival_time <value> <input port 1....input port n>
```
Specifies the latest arrival time (nanoseconds) of a signal at an input port. Specifies the maximum delay to the input port through external logic. This is a timing related command. Refer also to required_time.

**VHDL Syntax:**

```
attribute arrival_time : real;

attribute arrival_time of inputA:signal is 3 ns;
```

**IMPORTANT NOTE:** All input arrival times start at time zero and cannot be specified relative to a particular clock edge. You can adjust for a particular clock edge by adding the clock offset to the arrival time.

**auto_dissolve**

The `auto_dissolve` attribute allows you to dissolve an instance or view on a port. If you set `auto_dissolve` on a view, then all instantiations in the view are affected. If you set `auto_dissolve` on an instance, then only that instance is affected. The function of the `auto_dissolve` attribute is in contrast to the global functions of the `asic_auto_dissolve_limit` (ASIC 30) or `auto_dissolve_limit` (CPLD/FPGA 3000) variables. Refer also to the Variables chapter in this guide.

**Apply Attribute Examples:**

```
set_attribute -instance <view_name> -name auto_dissolve -value true
set_attribute -instance <instance_name> -name auto_dissolve -value true
```

**Remove Attribute Example:**

```
remove_attribute -instance <view_name> -name auto_dissolve
remove_attribute -instance <instance_name> -name auto_dissolve
```

**HDL Notes:** attribute name: `auto_dissolve` (case insensitive); attribute value: `[true|false]`

**blockRam**

Xilinx Virtex: By default RAMs mappable to block RAMs are mapped to block RAMs. Disable mapping to block RAMs by setting the following attribute to false.

```
set_attribute -instance <instance_name> -name block_ram -value false
```
buffer_sig

buffer_sig <buffer_type> <signal_name>

Specifies signals to be buffered. This is a signal buffering command.

Interactive Command Line Shell Syntax:

set_attribute -net <signal_name> -name buffer_sig -value <buffer_name>

Constraint File Syntax:

buffer_sig clkbuf clk1

clock_cycle

clock_cycle <clock period> <signal name>

Specifies the length (nanoseconds, real numbers) of the clock. This is a clock control command.

Note: clock_cycle is one of the three basic clock commands. The other two are: clock_offset and pulse_width.

Note: For flip flops the trailing edge occurs at time clock_offset + clock_cycle.

Interactive Command Line Shell Syntax:

set_attribute -port foo -name clock_cycle -value 30.0

VHDL Syntax:

attribute clock_cycle : real;

attribute clock_cycle of in_clock:signal is 30.0;

clock_offset

clock_offset <time> <signal name>

Specifies the time (nanoseconds, real numbers) of the leading edge offset from zero. This is a clock control command.
Note: clock_offset is one of the three basic clock commands. The other two are: clock_cycle and pulse_width.

Note: For both flip flops and latches, the leading edge occurs at time clock_offset. For flip flops the trailing edge occurs at time clock_offset + clock_cycle. For latches, the trailing edge occurs at time clock_offset + pulse_width.

Interactive Command Line Shell Syntax:

set_attribute -port foo -name clock_offset -value 5.0

VHDL Syntax:

attribute clock_offset : real;
attribute clock_offset of clock:signal is 5 ns;

dont_touch
dont_touch <instance name> <true or false>

dont_touch is used to mark desired instances to prevent unmapping and optimization. In contrast to noopt, dont_touch prevents optimization of the lower levels of hierarchy and leaf instances. This attribute is also available on the design browser. RMB over an object in design browser to popup a menu. Note: Refer to CAUTION for auto_write in Utilities chapter.

Interactive Command Line Shell Syntax:

set_attribute -instance foo -name dont_touch -value TRUE

input_drive

input_drive <value> <input signal>

Specifies the sensitivity to loading of the gate driving an input to the design. This is a load related command.

input_max_fanout

input_max_fanout <load>

Specifies the maximum fanout load the synthesized circuit presents at design input.
**input_max_load**

`input_max_load <load>`

Specifies the maximum load that the synthesized circuit may create on an input to the design.

**lut_max_fanout**

`lut_max_fanout <value integer>`

LeonardoSpectrum attempts to maintain reasonable fanouts by replicating the driver which results in net splitting. If replication is not possible, then the signal is buffered. This may make the wire slower by adding intrinsic delays. A Max Fanout window is on the Advance Technology FlowTab. Also refer to the Variables chapter.

**Interactive Command Line Shell Syntax:**

`set_attribute -net <net_name> -name lut_max_fanout -value <integer>`

**modgen_select**

`modgen_select <auto|fast|fastest|small|smallest>`

Operators in the modgen library use `auto|fast|fastest|small|smallest` to define the operation of a counter, adder, or multiplier, for example. Also see Commands.

**no_buff**

`no_buff <signal name> <true or false>`

Specifies signals that are not buffered internally. Works for input ports only. This is a signal buffering command.

**Note:** Input pins with `no_buff` applied are not buffered.

**Verilog:**

```verilog
//exemplar attribute <module_name> nobuff TRUE;
```
VHDL:

attribute no_buff : boolean
attribute no_buff inport: signal is TRUE;

Interactive Command Line Shell:

set_attribute -port abcde -name no_buff -type boolean -value TRUE

Note: You can prevent LeonardoSpectrum from buffering or resolving DRC violations on a net by setting the no_buff attribute:

set_attribute -net <net_name> -name no_buff -value TRUE

noopt

noopt <instance name> <true or false>

Specifies that an instance should not be optimized or changed. However, in contrast to dont_touch, lower level hierarchy and leaf instances are not protected from optimization or change. For example:

Verilog:

//exemplar attribute <module_name> noopt TRUE

VHDL:

attribute noopt : boolean;
attribute noopt of <component_name> : component is TRUE;

Interactive Command Line Shell:

set_attribute -instance abcde -name noopt -type boolean -value TRUE

Note: Refer to CAUTION for auto_write in the Utilities chapter.

nopad

Interactive Command Line Shell Syntax:

set_attribute -port <port_name> -name nopad -value true

VHDL Syntax:

attribute noopad : boolean
attribute nopad of <port_name>:signal is true

Verilog Syntax:

//exemplar attribute <port_name> nopad true

Note: Refer also to ORCA FPSC section in the ORCA chapter of the Synthesis and Technology guide.

output_fanout

output_fanout <load> <port>

Specifies the amount of external fanout loads on an output port of the design.

output_load

output_load <load> <port>

Specifies the number of external unit loads on an output port of the design. This is a load related command.

pad

pad <IO pad type> <signal name>

Specifies I/O gates to be used for specific signals. This is a signal buffering command.

Interactive Command Line Shell Syntax:

set_attribute -port <name> -name pad -value <pad_name>

Constraint File Syntax:

pad HCLKBUF hclk

VHDL Syntax:

attribute PAD of <signal_name>: signal is <pad_name>

pin_number

pin_number <pin number> <port name>

Assigns a device pin number to a certain port.

Note: Pin location corresponds to pin_number attribute.
Verilog Syntax:
//exemplar attribute clock pin_number 10;

VHDL Syntax:

```vhdl
attribute pin_number : string;
attribute pin_number of i : signal is “P10”;
```

Interactive Command Line Shell:

```shell
set_attribute pin_number abcde “P10”;
set_attribute -port <name> -name pin_number -value <pin_name>
```

**preserve_driver**

The attribute, `preserve_driver` is similar to `preserve_signal`. When you apply `preserve_driver`, LeonardoSpectrum preserves the specified signal and the driver in the design.

**preserve_driver** \(<signal\ name>\)

Specifies that both a signal and the signal name must survive optimization. **Note**: This is an attribute that is allowed in the constraint file.

**Verilog Example**:

```verilog
//exemplar attribute <signal_name> preserve_driver TRUE
```

**VHDL Example**:

Any parallel logic, such as a parallel inverters (gates), are optimized to a single instance. The attribute `preserve_driver` can be applied on the parallel signals to tell LeonardoSpectrum to maintain the parallel structure. Refer to the following:

```vhdl
library ieee;
use ieee_std_logic_1164.all
entity test is port ( 
a1 :in bit ;
z1, z2 :out std_logic 
);
attribute preserve_driver :boolean;
end test ;
```
architecture exemplar of test is
signal nz1, nz2 :bit;
attribute preserve_driver of nz1:signal is true ;
attribute preserve_driver of nz2:signal is true ;
begin
nz1 <= not(a1);
z2 <= not(a1);
z1 <= nz1;
z2 <= nz2;
end exemplar ;

Interactive Command Line Shell Example:

set_attribute -instance abc -name preserve_driver -type boolean -value TRUE

Constraint File Example:
<preserve_driver> <value> <signal_name>

preserve_signal

preserve_signal <signal name>

Specifies that both a signal and the signal name must survive optimization. This attribute is a preserve signal command. Note: Refer also to Users guide, Constraints chapter.

pull (up/dn)

Assign pullup or pulldown (resistors) to your ports. Note: The pull attribute is not available for the constraint file. Refer to the Synthesis & Technology guide, Xilinx chapter, for complete VHDL and Verilog examples.

Interactive Command Line Shell:

set attribute -port <port_name> -name pull -value "pullup"

VHDL:

... 

attribute pull:string;
attribute pull of inbus_a:signal is "pullup";
attribute pull of inbus_b:signal is "pulldn";

Verilog:

... 

pulse_width

pulse_width <clock_width>

Specifies the width (nanoseconds) of the clock pulse. This is a clock control command.

Note: pulse_width is one of the three basic clock commands. The other two are: clock_offset and clock_cycle.

Note: For latches, the trailing edge occurs at time clock_offset + pulse_width.

VHDL Syntax:

attribute pulse_width : real;

attribute pulse_width of clock:signal is 10 ns;

required_time

required_time <signal> <value>

Interactive Command Line Shell Syntax:

set_attribute outp -name required_time -value 25 -port

Constraint File Syntax:

required_time <value> <output port 1...output port n>
VHDL Syntax:

```vhdl
attribute required_time : real;

attribute required_time of out_port:signal is 10.0;
```

Specifies the latest time (nanoseconds, real numbers) a signal is allowed to arrive at an output port. This is a timing related command. Refer also to `arrival_time`.

**simple_register**

The ASIC mapper supports the attribute `simple_register` on signals. This attribute specifies that only simple registers are mapped to the signal. With this attribute, you can prevent certain signals from being mapped to complex registers. For example, prevent mapping to registers with synch clear. A simple register is a register that has only data, enable, and asynchronous inputs.

**unnoopt**

```vhdl
unnoopt <instance name>
```

Specifies that the `noopt` symbol is removed from the specified instance.
Index

A
abbreviated options 1-2
abbreviated Tcl commands 1-2
alias command 1-2
alias utility 3-9
alphabetical list 1-5, 2-2
attribute
  HDL object 4-1
attributes
  alphabetical list 4-4
auto_read
  read command 1-77, 3-2
auto_write
  Altera 3-2
  AMD Vantis 3-2
  utility 3-2
  write command 1-108, 3-2

C
character symbols 1-5
component
  corresponding view 4-2
connect
  connect 1-18
  connect path 1-20
create
  create 1-24
  create_rename_ruleset 1-26
  create_wrapper 1-27
customize
  command line interface 1-4

D
disconnect
disconnect 1-31
disconnect_path 1-33

E
exemplar.ini 1-4

H
HDL objects
  predefined data 4-1
help command 1-3
history tracking 1-4

L
label
  corresponding to instance 4-2
Level 3 command
  analyze 1-12
  elaborate 1-35
  group 1-40
  unfold 1-101
Level 3 commands
  group, unfold, analyze, elaborate 1-1
list
  list_attributes 1-44
  list_connection 1-46
  list_design 1-49
  list_technologies 1-52
load
  load_library 1-53
  load_modgen 1-55

M
messages
optimize.log 1-3
total.log 1-3

N
name style 2-2

O
optimize
  optimize 1-60
  optimize_timing 1-66
  preoptimize 1-71

R
read
  auto_read 1-77
  read command 1-74
redirecting output 1-3
remove
  remove 1-79
  remove_attribute 1-81
  remove_clock 1-83
  remove_rename_ruleset 1-84
report
  report_area 1-85
  report_constraints 1-87
  report_delay 1-89
  report_rename_rules 1-93
  report_wire_tables 1-94

S
set
  set_attribute 1-95
  set_clock 1-97
  set_multicycle_path 1-98
slash character 1-5
syntax
  constraint file 4-4
  declaration 4-2
  interactive command line shell 4-3

Verilog 4-3
VHDL 4-2

T
Tcl language 1-3
Tcl script
  command line with path 1-1
  Exemplar history file 1-1
  interactive command line shell 1-1
  run script 1-1
Tcl variables 2-1

U
utility aliases 3-9
utility scripts 3-1

V
variable editor 2-2
Verilog array objects 2-2
VHDL array objects 2-2

W
write
  auto_write 1-108
  write command 1-105