

Vivado Design Suite Tcl Command Reference Guide

UG835 (v 2013.1) March 20, 2013

Introduction

Overview of Tcl Capabilities in Vivado

The Tool Command Language (Tcl) is the scripting language integrated in the Vivado™ tool environment. Tcl is a standard language in the semiconductor industry for application programming interfaces, and is used by Synopsys® Design Constraints (SDC).

SDC is the mechanism for communicating timing constraints for FPGA synthesis tools from Synopsys Synplify as well as other vendors, and is a timing constraint industry standard; consequently, the Tcl infrastructure is a “Best Practice” for scripting language.

Tcl lets you perform interactive queries to design tools in addition to executing automated scripts. Tcl offers the ability to “ask” questions interactively of design databases, particularly around tool and design settings and state. Examples are: querying specific timing analysis reporting commands live, applying incremental constraints, and performing queries immediately after to verify expected behavior without re-running any tool steps.

The following sections describe some of the basic capabilities of Tcl with Vivado.

Note This manual is not a comprehensive reference for the Tcl language. It is a reference to the specific capabilities of the Vivado Design Suite Tcl shell, and provides reference to additional Tcl programming resources.

Launching the Vivado Design Suite

You can launch the Vivado Design Suite and run the tools using different methods depending on your preference. For example, you can choose a Tcl script-based compilation style method in which you manage sources and the design process yourself, also known as Non-Project Mode. Alternatively, you can use a project-based method to automatically manage your design process and design data using projects and project states, also known as Project Mode. Either of these methods can be run using a Tcl scripted batch mode or run interactively in the Vivado IDE. For more information on the different design flow modes, see the *Vivado Design Suite User Guide: Design Flows Overview (UG892)*.

Tcl Shell Mode

If you prefer to work directly with Tcl commands, you can interact with your design using Tcl commands with one of the following methods:

- Enter individual Tcl commands in the Vivado Design Suite Tcl shell outside of the Vivado IDE.
- Enter individual Tcl commands in the Tcl Console at the bottom of the Vivado IDE.
- Run Tcl scripts from the Vivado Design Suite Tcl shell.
- Run Tcl scripts from the Vivado IDE.

Use the following command to invoke the Vivado Design Suite Tcl shell either at the Linux command prompt or within a Windows Command Prompt window:

```
vivado -mode tcl
```

Tip On Windows, you can also select **Start > All Programs > Xilinx Design Tools > Vivado 2013.x > Vivado 2013.x Tcl Shell**.

For more information about using Tcl and Tcl scripting, see the *Vivado Design Suite User Guide: Using the Tcl Scripting Capabilities (UG894)*. For a step-by-step tutorial that shows how to use Tcl in the Vivado tool, see the *Vivado Design Suite Tutorial: Design Flows Overview (UG888)*.

Tcl Batch Mode

You can use the Vivado tools in batch mode by supplying a Tcl script when invoking the tool. Use the following command either at the Linux command prompt or within a Windows Command Prompt window:

```
vivado -mode batch -source <your_Tcl_script>
```

The Vivado Design Suite Tcl shell will open, run the specified Tcl script, and exit when the script completes. In batch mode, you can queue up a series of Tcl scripts to process a number of designs overnight through synthesis, simulation, and implementation, and review the results on the following morning.

Vivado IDE Mode

If you prefer to work in a GUI, you can launch the Vivado IDE from Windows or Linux. For more information on the Vivado IDE, see the *Vivado Design Suite User Guide: Using the Vivado IDE (UG893)*.

Launch the Vivado IDE from your working directory. By default the Vivado journal and log files, and any generated report files, are written to the directory from which the Vivado tool is launched. This makes it easier to locate the project file, log files, and journal files, which are written to the launch directory.

In the Windows OS, select **Start > All Programs > Xilinx Design Tools > Vivado 2013.x > Vivado 2013.x**.

Tip You can also double-click the Vivado IDE shortcut icon on your Windows desktop.

In the Linux OS, enter the following command at the command prompt:

```
vivado  
-or-  
vivado -mode gui
```

If you need help, with the Vivado tool command line executable, type:

```
vivado -help
```

If you are running the Vivado tool from the Vivado Design Suite Tcl shell, you can open the Vivado IDE directly from the Tcl shell by using the **start_gui** command.

From the Vivado IDE, you can close the Vivado IDE and return to a Vivado Tcl shell by using the **stop_gui** command.

Tcl Journal Files

When you invoke the Vivado tool, it writes the `vivado.log` file to record the various commands and operations performed during the design session. The Vivado tool also writes a file called `vivado.jou` which is a journal of just the Tcl commands run during the session. The journal file can be used as a source to create new Tcl scripts.

Note Backup versions of the journal file, named `vivado_<id>.backup.jou`, are written to save the details of prior runs whenever the Vivado tool is launched. The `<id>` is a unique identifier that allow the tool to create and store multiple backup versions of the log and journal files.

Tcl Help

The Tcl help command provides information related to the supported Tcl commands.

- `help` — Returns a list of Tcl command categories.

help

Command categories are groups of commands performing a specific function, like File I/O for instance.

- `help -category category` — Returns a list of commands found in the specified category.

help -category object

This example returns the list of Tcl commands for handling objects.

- `help pattern` — Returns a list of commands that match the specified search pattern. This form can be used to quickly locate a specific command from a group of commands.

help get_*

This example returns the list of Tcl commands beginning with `get_`.

- `help command` — Provides detailed information related to the specified command.

help get_cells

This example returns specific information of the `get_cells` command.

- `help -args command` — Provides an abbreviated help text for the specified command, including the command syntax and a brief description of each argument.

help -args get_cells

- `help -syntax command` — Reports the command syntax for the specified command.

help -syntax get_cells

Tcl Initialization Scripts

When you start the Vivado tool, it looks for a Tcl initialization script in two different locations:

1. In the software installation: `installdir/Vivado/version/scripts/init.tcl`
2. In the local user directory:
 - For Windows 7: `%APPDATA%/Xilinx/Vivado/init.tcl`
 - For Linux: `$HOME/.Xilinx/Vivado/init.tcl`

Where:

installdir is the installation directory where the Vivado Design Suite is installed.

If `init.tcl` exists, in one or both of those locations, the Vivado tool sources this file; first from the installation directory and second from your home directory.

- The `init.tcl` file in the installation directory allows a company or design group to support a common initialization script for all users. Anyone starting the Vivado tool from that installation location sources the enterprise `init.tcl` script.
- The `init.tcl` file in the home directory allows each user to specify additional commands, or to override commands from the software installation to meet their specific design requirements.
- No `init.tcl` file is provided with the Vivado Design Suite installation. You must create the `init.tcl` file and place it in either the installation directory, or your home directory, as discussed to meet your specific needs.

The `init.tcl` file is a standard Tcl command file that can contain any valid Tcl command supported by the Vivado tool. You can also source another Tcl script file from within `init.tcl` by adding the following statement:

```
source path_to_file/file_name.tcl
```

Note You can also specify the **-init** option when launching the Vivado Design Suite from the command line. Type **vivado -help** for more information.

Sourcing a Tcl Script

A Tcl script can be sourced from either one of the command-line options or from the GUI. Within the Vivado Integrated Design Environment (IDE) you can source a Tcl script from **Tools > Run Tcl Script**.

You can source a Tcl script from a Tcl command-line option:

```
source file_name
```

When you invoke a Tcl script from the Vivado IDE, a progress bar is displayed and all operations in the IDE are blocked until the scripts completes.

There is no way to interrupt script execution during run time; consequently, standard OS methods of killing a process must be used to force interruption of the tool. If the process is killed, you lose any work done since your last save.

Typing **help source** in the Tcl console will provide additional information regarding the **source** command.

Using Tcl.pre and Tcl.post Hook Scripts

Tcl Hook scripts allow you to run custom Tcl scripts prior to (`tcl.pre`) and after (`tcl.post`) synthesis and implementation design runs, or any of the implementation steps. Whenever you launch a run, the Vivado tool uses a predefined Tcl script which executes a design flow based on the selected strategy. Tcl Hook scripts let you customize the standard flow, with pre-processors or post-processors, such as for generating custom reports. The Tcl Hook script must be a standard Tcl script.

Every step in the design flow has a pre- and post-hook capability. Common examples are:

- Custom reports: timing, power, utilization, or any user-defined tcl report.
- Temporary parameters for workarounds.
- Over-constraining timing constraints for portions of the flow.
- Multiple iterations of stages (e.g. multiple calls to `phys_opt_design`).
- Modifications to netlist, constraint, or device programming.

For more information on defining Tcl Hook scripts, refer to the *Vivado Design Suite User Guide: Using Tcl Scripting (UG894)*.

General Tcl Syntax Guidelines

Tcl uses the Linux file separator (/) convention regardless of which Operating System you are running.

The following subsections describe the general syntax guidelines for using Tcl in the Vivado Design Suite.

Using Tcl Eval

When executing Tcl commands, you can use variable substitution to replace some of the command line arguments accepted or required by the Tcl command. However, you must use the Tcl **eval** command to evaluate the command line with the Tcl variable as part of the command.

For instance, the help command can take the **-category** argument, with one of a number of command categories as options:

```
help -category ipflow
```

You can define a variable to hold the command category:

```
set cat "ipflow"
```

Where:

- **set** is the Tcl keyword that defines the variable.
- **cat** is the name of the variable being defined.
- **"ipflow"** is the value assigned to the variable.

You can then evaluate the variable in the context of the Tcl command:

```
eval help -category $cat
```

or,

```
set cat "category ipflow"  
eval help $cat
```

You can also use braces {} in place of quotation marks "" to achieve the same result:

```
set runblocksOptDesignOpts { -sweep -retarget -propconst -remap }  
eval opt_design $runblocksOptDesignOpts
```

Typing **help eval** in the Tcl console will provide additional information regarding the **eval** command.

Using Special Characters

Some commands take arguments that contain characters that have special meaning to Tcl. Those arguments must be surrounded with curly braces {} to avoid unintended processing by Tcl. The most common cases are as follows.

Bus Indexes - Because square brackets [] have special meaning to Tcl, an indexed (bit- or part-selected) bus using the square bracket notation must be surrounded with curly braces. For example, when adding index 4 of a bus to the Vivado Common Waveform Viewer window using the square bracket notation, you must write the command as:

```
add_wave {bus[4]}
```

Parentheses can also be used for indexing a bus, and because parentheses have no special meaning to Tcl, the command can be written without curly braces. For example:

```
add_wave bus(4)
```

Verilog Escaped Identifiers - Verilog identifiers containing characters or keywords that are reserved by Verilog need to be “escaped” both in the Verilog source code and on the simulator command line by prefixing the identifier with a backslash “\” and appending a space. Additionally, on the Tcl command line the escaped identifier must be surrounded with curly braces.

Note If an identifier already includes a curly brace, then the technique of surrounding the identifier with curly braces does not work, because Tcl interprets curly braces as reserved characters even nested within curly braces. Instead, you must use the technique described below, in VHDL Extended Identifiers.

For example, to add a wire named “**my wire**” to the Vivado Common Waveform Viewer window, you must write the command as:

```
add_wave {\my wire }
```

Note Be sure to append a space after the final character, and before the closing brace.

Verilog allows any identifier to be escaped. However, on the Tcl command line do not escape identifiers that are not required to be escaped. For example, to add a wire named “w” to the Vivado Common Waveform Viewer window, the Vivado simulator would not accept:

```
add_wave {\w }
```

as a valid command, since this identifier (the wire name “w”) does not required to be escaped. The command must be written as:

```
add_wave w
```

VHDL Extended Identifiers - VHDL extended identifiers contain backslashes, “\”, which are reserved characters in Tcl. Because Tcl interprets a backslash next to a close curly brace \} as being a close curly brace character, VHDL extended identifiers cannot be written with curly braces. Instead, the curly braces must be absent and each special character to Tcl must be prefixed with a backslash. For example, to add the signal \my sig\ to the Wave window, you must write the command as:

```
add_wave \\my\\ sig\\
```

Note Both the backslashes that are part of the extended identifier, and the space inside the identifier are prefixed with a backslash.

General Syntax Structure

The general structure of Vivado Design Suite Tcl commands is:

```
command [optional_parameters] required_parameters
```

Command syntax is of the verb-noun and verb-adjective-noun structure separated by the underscore (“_”) character.

Commands are grouped together with common prefixes when they are related.

- Commands that query things are generally prefixed with `get_`.
- Commands that set a value or a parameter are prefixed with `set_`.
- Commands that generate reports are prefixed with `report_`.

The commands are exposed in the global namespace. Commands are “flattened,” meaning there are no “sub-commands” for a command.

Example Syntax

Following is an example of the return format on the `get_cells -help` command:

```
get_cells
```

Description:

Get a list of cells in the current design

Syntax:

```
get_cells [-hsc arg ] [-hierarchical] [-regexp] [-nocase] [-filter arg ]
          [-of_objects args ] [-match_style arg ] [-quiet] [ patterns ]
```

Returns:

list of cell objects

Usage:

Name	Optional	Default	Description
-hsc	yes	/	Hierarchy separator
-hierarchical	yes		Search level-by-level in current instance
-regexp	yes		Patterns are full regular expressions
-nocase	yes		Perform case-insensitive matching (valid only when -regexp specified)
-filter	yes		Filter list with expression
-of_objects	yes		Get cells of these pins or nets
-match_style	yes	sd	Style of pattern matching, valid values are ucf, sd
-quiet	yes		Ignore command errors
patterns	yes	*	Match cell names against patterns

Categories:

SDC, XDC, Object

Unknown Commands

Tcl contains a list of built-in commands that are generally supported by the language, Vivado tool specific commands which are exposed to the Tcl interpreter, and user-defined procedures.

Commands that do not match any of these known commands are sent to the OS for execution in the shell from the `exec` command. This lets users execute shell commands that might be OS-specific. If there is no shell command, then an error message is issued to indicate that no command was found.

Return Codes

Some Tcl commands are expected to provide a return value, such as a list or collection of objects on which to operate. Other commands perform an action but do not necessarily return a value that can be used directly by the user. Some tools that integrate Tcl interfaces return a 0 or a 1 to indicate success or error conditions when the command is run.

To properly handle errors in Tcl commands or scripts, you should use the Tcl built-in command `catch`. Generally, the `catch` command and the presence of numbered info, warning, or error messages should be relied upon to assess issues in Tcl scripted flows.

Vivado tool Tcl commands return either `TCL_OK` or `TCL_ERROR` upon completion. In addition, the Vivado Design Suite sets the global variable `$ERRORINFO` through standard Tcl mechanisms.

To take advantage of the `$ERRORINFO` variable, use the following line to report the variable after an error occurs in the Tcl console:

`puts $ERRORINFO`

This reports specific information to the standard display about the error. For example, the following code example shows a Tcl script (`procs.tcl`) being sourced, and a user-defined procedure (`loads`) being run. There are a few transcript messages, and then an error is encountered at line 5.

```
Line 1: Vivado % source procs.tcl
Line 2: Vivado% loads
Line 3: Found 180 driving FFs
Line 4: Processing pin a_reg_reg[1]/Q...
Line 5: ERROR: [HD-Tcl 53] Cannot specify '-patterns' with '-of_objects'.
Line 6: Vivado% puts $errorInfo
Line 7: ERROR: [HD-Tcl 53] Cannot specify '-patterns' with '-of_objects'. While executing
"get_ports -of objects $pin" (procedure "my_report" line 6) invoked from within procs.tcl
```

You can add **`puts $ERRORINFO`** into catch clauses in your Tcl script files to report the details of an error when it is caught, or use the command interactively in the Tcl console immediately after an error is encountered to get the specific details of the error.

In the example code above, typing the **`puts $ERRORINFO`** command in line 6, reports detailed information about the command and its failure in line 7.

First Class Tcl Objects and Relationships

The Tcl commands in the Vivado Design Suite provide direct access to the object models for netlist, devices, and projects. These objects are first-class which means they are more than just a string representation, and they can be operated on and queried. There are a few exceptions to this rule, but generally "things" can be queried as objects, and these objects have properties that can be queried and they have relationships that allow you to get to other objects.

Object Types and Definitions

There are many object types in the Vivado Design Suite; this chapter provides definitions and explanations of the basic types. The most basic and important object types are associated with entities in a design netlist, and these types are listed in the following subsections:

Cell

A cell is an instance, either primitive or hierarchical inside a netlist. Examples of cells include flip-flops, LUTs, I/O buffers, RAM and DSPs, as well as hierarchical instances which are wrappers for other groups of cells.

Pin

A pin is a point of logical connectivity on a cell. A pin allows the internals of a cell to be abstracted away and simplified for easier use, and can either be on hierarchical or primitive cells. Examples of pins include clock, data, reset, and output pins of a flop.

Port

A port is a special type of hierarchical pin, a pin on the top level netlist object, module or entity. Ports are normally attached to I/O pads and connect externally to the FPGA device.

Net

A net is a wire or list of wires that eventually be physically connected directly together. Nets can be hierarchical or flat, but always sorts a list of pins together.

Clock

A clock is a periodic signal that propagates to sequential logic within a design. Clocks can be primary clock domains or generated by clock primitives such as a DCM, PLL, or MMCM. A clock is the rough equivalent to a TIMESPEC PERIOD constraint in UCF and forms the basis of static timing analysis algorithms.

Querying Objects

All first class objects can be queried by a `get_ Tcl` command that generally has the following syntax:

```
get_object_type pattern
```

Where `pattern` is a search pattern, which includes if applicable a hierarchy separator to get a fully qualified name. Objects are generally queried by a string pattern match applied at each level of the hierarchy, and the search pattern also supports wildcard style search patterns to make it easier to find objects, for example:

```
get_cells */inst_1
```

This command searches for a cell named `inst_1` within the first level of hierarchy under the top-level of hierarchy. To recursively search for a pattern at every level of hierarchy, use the following syntax:

```
get_cells -hierarchical inst_1
```

This command searches every level of hierarchy for any instances that match `inst_1`.

For complete coverage of syntax, see the specific online help for the individual command:

- `help get_cells`
- `get_cells -help`

Object Properties

Objects have properties that can be queried. Property names are unique for any given object type. To query a specific property for an object, the following command is provided:

```
get_property property_name object
```

An example would be the `lib_cell` property on cell objects, which tells you what UniSim component a given instance is mapped to:

```
get_property lib_cell [get_cell inst_1]
```

To discover all of the available properties for a given object type, use the **report_property** command:

```
report_property [get_cells inst_1]
```

The following table shows the properties returned for a specific object.

Reported Properties for Specified Object

Key	Value	Type
bel	OLOGICE1.OUTFF	string
class	cell	string
iob	TRUE	string
is_blackbox	0	bool
is_fixed	0	bool
is_partition	0	bool
is_primitive	1	bool
is_reconfigurable	0	bool
is_sequential	1	bool
lib_cell	FD	string
loc	OLOGIC_X1Y27	string
name	error	string
primitive_group	FD_LD	string
primitive_subgroup	flop	string
site	OLOGIC_X1Y27	string
type	FD & LD	string
XSTLIB	1	bool

Some properties are read-only and some are user-settable. Properties that map to attributes that can be annotated in UCF or in HDL are generally user-settable through Tcl with the `set_property` command:

```
set_property loc OLOGIC_X1Y27 [get_cell inst_1]
```

Filtering Based on Properties

The object query `get_*` commands have a common option to filter the query based on any property value attached to the object. This is a powerful capability for the object query commands. For example, to query all cells of primitive type FD do the following:

```
get_cells * -hierarchical -filter "lib_cell == FD"
```

To do more elaborate string filtering, utilize the `=~` operator to do string pattern matching. For example, to query all flip-flop types in the design, do the following:

```
get_cells * -hierarchical -filter "lib_cell =~ FD*"
```

Multiple filter properties can be combined with other property filters with logical OR (`||`) and AND (`&&`) operators to make very powerful searches. To query every cell in the design that is of any flop type and has a placed location constraint:

```
get_cells * -hierarchical -filter {lib_cell =~ FD* && loc != ""}
```

Note In the example, the filter option value was wrapped with curly braces `{}` instead of double quotes. This is normal Tcl syntax that prevents command substitution by the interpreter and allows users to pass the empty string (`""`) to the `loc` property.

Large Lists of Objects

Commands that return more than one object generally return a container that looks and behaves like a native Tcl list. This is a feature of the Vivado tool in that it allows dramatic optimization of large lists of Tcl objects handling without the need for special iteration commands like the `foreach_in_collection` command that other tools have implemented. This is handled with the Tcl built-in **foreach** command.

There are a few nuances with respect to large lists, particularly in the log files and the GUI Tcl console. Typically, when you set a Tcl variable to the result of a **get_*** command, the entire list is echoed to the console and to the log file. For large lists, this is truncated when printed to the console and log to prevent memory overloading of the buffers in the tool.

What is echoed is the list printed to the log and console is truncated and the last element appears to be "..." in the log and console, however the actual list in the variable assignment is still correct and the last element is not an error. An example of this is querying a single cell versus every cell in the design, which can be large:

```
get_cells inst_1
inst_1
get_cells * -hierarchical
XST_VCC XST_GND error readIngressFifo wbDataForInputReg fifoSelect_0 fifoSelect_1 fifoSelect_2 fifoSelect_3 ...
%set x [get_cells * -hierarchical]
XST_VCC XST_GND error readIngressFifo wbDataForInputReg fifoSelect_0 fifoSelect_1 fifoSelect_2 fifoSelect_3 ...
%lindex $x end
bftClk_BUFGRP/bufg
%llength $x
4454
```

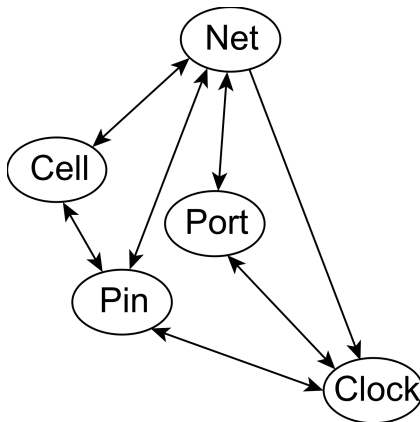
In this example, all four thousand cells were not printed to the console and the list was truncated with a "..." but the actual last element of the list is still correct in the Tcl variable.

Object Relationships

Related objects can be queried using the **-of** option to the relevant **get_*** command. For example, to get a list of pins connected to a cell object, do the following:

```
get_pins -of [get_cells inst_1]
```

The following image shows object types in the Vivado tool and their relationships, where an arrow from one object to another object indicates that you can use the **-of** option to the **get_*** command to traverse logical connectivity and get Tcl references to any connected object.



Errors, Warnings, Critical Warnings, and Info Messages

Messages that result from individual commands appear in the log file as well as in the GUI console if it is active. These messages are generally numbered to identify specific issues and are prefixed in the log file with "INFO", "WARNING", "CRITICAL_Warning", "ERROR" followed by a subsystem identifier and a unique number.

The following example shows an INFO message that appears after reading the timing library.

```
INFO: [HD-LIB 1] Done reading timing library
```

These messages make it easier to search for specific issues in the log file to help to understand the context of operations during command execution.

Generally, when an error occurs in a Tcl command sourced from a Tcl script, further execution of subsequent commands is halted. This is to prevent unrecoverable error conditions. There are Tcl built-ins that allow users to intercept these error conditions, and to choose to continue. Consult any Tcl reference for the catch command for a description of how to handle errors using general Tcl mechanisms.

Tcl Commands Listed by Category

Categories

- [Board](#)
- [ChipScope](#)
- [DRC](#)
- [FileIO](#)
- [Floorplan](#)
- [GUIControl](#)
- [Hardware](#)
- [IPFlow](#)
- [IPIntegrator](#)
- [Netlist](#)
- [Object](#)
- [PinPlanning](#)
- [Power](#)
- [Project](#)
- [PropertyAndParameter](#)
- [Report](#)
- [SDC](#)
- [Simulation](#)
- [SysGen](#)
- [Timing](#)
- [ToolLaunch](#)
- [Tools](#)
- [Waveform](#)
- [XDC](#)
- [XPS](#)

Board:

- [current_board](#)
 - [get_board_interfaces](#)
 - [get_board_pins](#)
-

ChipScope:

- [launch_chipscope_analyzer](#)
 - [launch_impact](#)
 - [write_chipscope_cdc](#)
-

DRC:

- [add_drc_checks](#)
- [create_drc_check](#)
- [create_drc_ruledeck](#)
- [create_drc_violation](#)
- [delete_drc_check](#)
- [delete_drc_ruledeck](#)
- [get_drc_checks](#)
- [get_drc_ruledecks](#)
- [get_drc_vios](#)
- [remove_drc_checks](#)
- [report_drc](#)
- [reset_drc](#)
- [reset_drc_check](#)

FileIO:

- [config_webtalk](#)
- [infer_diff_pairs](#)
- [pr_verify](#)
- [read_checkpoint](#)
- [read_csv](#)
- [read_edif](#)
- [read_ip](#)
- [read_saif](#)
- [read_twx](#)
- [read_vcd](#)
- [read_verilog](#)
- [read_vhdl](#)
- [read_xdc](#)
- [write_bitstream](#)
- [write_bmm](#)
- [write_checkpoint](#)
- [write_chipscope_cdc](#)
- [write_csv](#)
- [write_debug_probes](#)
- [write_edif](#)
- [write_ibis](#)
- [write_sdf](#)
- [write_verilog](#)
- [write_vhdl](#)
- [write_xdc](#)

Floorplan:

- [add_cells_to_pblock](#)
- [create_pblock](#)
- [delete_pblock](#)
- [delete_rpm](#)
- [get_pblocks](#)
- [place_cell](#)
- [place_pblocks](#)
- [remove_cells_from_pblock](#)
- [reset_ucf](#)
- [resize_pblock](#)
- [swap_locs](#)
- [unplace_cell](#)

GUIControl:

- [endgroup](#)
- [get_selected_objects](#)
- [highlight_objects](#)
- [mark_objects](#)
- [redo](#)
- [select_objects](#)
- [show_objects](#)
- [show_schematic](#)
- [start_gui](#)
- [startgroup](#)
- [stop_gui](#)
- [undo](#)
- [unhighlight_objects](#)
- [unmark_objects](#)
- [unselect_objects](#)

Hardware:

- [close_hw](#)
- [close_hw_target](#)

- `commit_hw_sio`
- `commit_hw_vio`
- `connect_hw_server`
- `create_hw_sio_link`
- `create_hw_sio_linkgroup`
- `create_hw_sio_scan`
- `current_hw_device`
- `current_hw_ila`
- `current_hw_ila_data`
- `current_hw_server`
- `current_hw_target`
- `disconnect_hw_server`
- `display_hw_ila_data`
- `display_hw_sio_scan`
- `get_hw_devices`
- `get_hw_ila_datas`
- `get_hw_ilas`
- `get_hw_probes`
- `get_hw_servers`
- `get_hw_sio_commons`
- `get_hw_sio_gtgroups`
- `get_hw_sio_gts`
- `get_hw_sio_iberts`
- `get_hw_sio_linkgroups`
- `get_hw_sio_links`
- `get_hw_sio_plls`
- `get_hw_sio_rxs`
- `get_hw_sio_scans`
- `get_hw_sio_txs`
- `get_hw_targets`
- `get_hw_vios`
- `open_hw`
- `open_hw_target`
- `program_hw_devices`
- `read_hw_ila_data`
- `read_hw_sio_scan`
- `refresh_hw_device`
- `refresh_hw_server`
- `refresh_hw_sio`
- `refresh_hw_target`

- [refresh_hw_vio](#)
 - [remove_hw_sio_link](#)
 - [remove_hw_sio_linkgroup](#)
 - [remove_hw_sio_scan](#)
 - [reset_hw_ila](#)
 - [reset_hw_vio_activity](#)
 - [reset_hw_vio_outputs](#)
 - [run_hw_ila](#)
 - [run_hw_sio_scan](#)
 - [stop_hw_sio_scan](#)
 - [upload_hw_ila_data](#)
 - [wait_on_hw_ila](#)
 - [wait_on_hw_sio_scan](#)
 - [write_hw_ila_data](#)
 - [write_hw_sio_scan](#)
-

IPFlow:

- [copy_ip](#)
 - [create_ip](#)
 - [generate_target](#)
 - [get_ipdefs](#)
 - [get_ips](#)
 - [import_ip](#)
 - [open_example_project](#)
 - [read_ip](#)
 - [report_ip_status](#)
 - [reset_target](#)
 - [update_ip_catalog](#)
 - [upgrade_ip](#)
 - [validate_ip](#)
-

IPIntegrator:

- [apply_bd_automation](#)
- [assign_bd_address](#)
- [close_bd_design](#)
- [connect_bd_intf_net](#)

- [connect_bd_net](#)
- [copy_bd_objs](#)
- [create_bd_addr_seg](#)
- [create_bd_cell](#)
- [create_bd_design](#)
- [create_bd_intf_net](#)
- [create_bd_intf_pin](#)
- [create_bd_intf_port](#)
- [create_bd_net](#)
- [create_bd_pin](#)
- [create_bd_port](#)
- [current_bd_design](#)
- [current_bd_instance](#)
- [delete_bd_objs](#)
- [disconnect_bd_intf_net](#)
- [disconnect_bd_net](#)
- [find_bd_objs](#)
- [generate_target](#)
- [get_bd_addr_segs](#)
- [get_bd_addr_spaces](#)
- [get_bd_cells](#)
- [get_bd_designs](#)
- [get_bd_intf_nets](#)
- [get_bd_intf_pins](#)
- [get_bd_intf_ports](#)
- [get_bd_nets](#)
- [get_bd_pins](#)
- [get_bd_ports](#)
- [group_bd_cells](#)
- [move_bd_cells](#)
- [open_bd_design](#)
- [regenerate_bd_layout](#)
- [save_bd_design](#)
- [ungroup_bd_cells](#)
- [validate_bd_design](#)
- [write_bd_tcl](#)

Netlist:

- [connect_net](#)
- [create_cell](#)
- [create_net](#)
- [create_pin](#)
- [disconnect_net](#)
- [get_net_delays](#)
- [remove_cell](#)
- [remove_net](#)
- [remove_pin](#)
- [rename_ref](#)
- [resize_net_bus](#)
- [resize_pin_bus](#)
- [tie_unused_pins](#)

Object:

- [add_drc_checks](#)
- [create_drc_check](#)
- [create_drc_ruledeck](#)
- [current_board](#)
- [delete_drc_check](#)
- [delete_drc_ruledeck](#)
- [filter](#)
- [get_bel_pins](#)
- [get_bels](#)
- [get_board_interfaces](#)
- [get_board_pins](#)
- [get_boards](#)
- [get_cells](#)
- [get_clock_regions](#)
- [get_clocks](#)
- [get_debug_cores](#)
- [get_debug_ports](#)
- [get_delays](#)
- [get_designs](#)
- [get_drc_checks](#)

- [get_drc_ruledecks](#)
- [get_drc_vios](#)
- [get_files](#)
- [get_filesets](#)
- [get_generated_clocks](#)
- [get_hw_devices](#)
- [get_hw_ila_datas](#)
- [get_hw_ilas](#)
- [get_hw_probes](#)
- [get_hw_servers](#)
- [get_hw_sio_commons](#)
- [get_hw_sio_gtgroups](#)
- [get_hw_sio_gts](#)
- [get_hw_sio_iberts](#)
- [get_hw_sio_linkgroups](#)
- [get_hw_sio_links](#)
- [get_hw_sio_plls](#)
- [get_hw_sio_rxs](#)
- [get_hw_sio_scans](#)
- [get_hw_sio_txs](#)
- [get_hw_targets](#)
- [get_hw_vios](#)
- [get_interfaces](#)
- [get_io_standards](#)
- [get_iobanks](#)
- [get_ipdefs](#)
- [get_ips](#)
- [get_lib_cells](#)
- [get_lib_pins](#)
- [get_libs](#)
- [get_macros](#)
- [get_net_delays](#)
- [get_nets](#)
- [get_nodes](#)
- [get_package_pins](#)
- [get_parts](#)
- [get_path_groups](#)
- [get_pblocks](#)
- [get_pins](#)
- [get_pips](#)

- [get_ports](#)
- [get_projects](#)
- [get_property](#)
- [get_runs](#)
- [get_selected_objects](#)
- [get_site_pins](#)
- [get_site_pips](#)
- [get_sites](#)
- [get_slrs](#)
- [get_tiles](#)
- [get_timing_arcs](#)
- [get_timing_paths](#)
- [get_wires](#)
- [list_property](#)
- [list_property_value](#)
- [remove_drc_checks](#)
- [report_property](#)
- [reset_drc_check](#)
- [reset_property](#)
- [set_property](#)

PinPlanning:

- [create_interface](#)
- [create_port](#)
- [delete_interface](#)
- [make_diff_pair_ports](#)
- [place_ports](#)
- [remove_port](#)
- [resize_port_bus](#)
- [set_package_pin_val](#)
- [split_diff_pair_ports](#)

Power:

- [delete_power_results](#)
- [power_opt_design](#)
- [read_saif](#)
- [read_vcd](#)
- [report_power](#)
- [report_power_opt](#)
- [reset_default_switching_activity](#)
- [reset_operating_conditions](#)
- [reset_switching_activity](#)
- [set_default_switching_activity](#)
- [set_operating_conditions](#)
- [set_power_opt](#)
- [set_switching_activity](#)

Project:

- [add_files](#)
- [archive_project](#)
- [close_design](#)
- [close_project](#)
- [copy_ip](#)
- [create_fileset](#)
- [create_project](#)
- [create_run](#)
- [current_board](#)
- [current_fileset](#)
- [current_project](#)
- [current_run](#)
- [delete_fileset](#)
- [delete_run](#)
- [find_top](#)
- [generate_target](#)
- [get_board_interfaces](#)
- [get_board_pins](#)
- [get_boards](#)
- [get_files](#)

- `get_filesets`
- `get_ips`
- `get_projects`
- `get_runs`
- `help`
- `import_files`
- `import_ip`
- `import_synplify`
- `import_xise`
- `import_xst`
- `launch_runs`
- `list_targets`
- `lock_design`
- `make_wrapper`
- `move_files`
- `open_example_project`
- `open_io_design`
- `open_project`
- `open_run`
- `refresh_design`
- `reimport_files`
- `remove_files`
- `reorder_files`
- `report_compile_order`
- `reset_project`
- `reset_run`
- `reset_target`
- `save_constraints`
- `save_constraints_as`
- `save_project_as`
- `set_speed_grade`
- `update_compile_order`
- `update_design`
- `update_files`
- `wait_on_run`

PropertyAndParameter:

- `create_property`
- `filter`
- `get_param`
- `get_property`
- `list_param`
- `list_property`
- `list_property_value`
- `report_param`
- `report_property`
- `reset_param`
- `reset_property`
- `set_param`
- `set_property`

Report:

- `check_timing`
- `create_drc_violation`
- `create_slack_histogram`
- `delete_clock_networks_results`
- `delete_timing_results`
- `delete_utilization_results`
- `get_msg_config`
- `get_msg_count`
- `get_msg_limit`
- `report_carry_chains`
- `report_clock_interaction`
- `report_clock_networks`
- `report_clock_utilization`
- `report_clocks`
- `report_config_timing`
- `report_control_sets`
- `report_datasheet`
- `report_debug_core`
- `report_default_switching_activity`
- `report_disable_timing`

- [report_drc](#)
- [report_environment](#)
- [report_exceptions](#)
- [report_high_fanout_nets](#)
- [report_incremental_reuse](#)
- [report_io](#)
- [report_operating_conditions](#)
- [report_param](#)
- [report_phys_opt](#)
- [report_power](#)
- [report_property](#)
- [report_pulse_width](#)
- [report_route_status](#)
- [report_ssn](#)
- [report_switching_activity](#)
- [report_timing](#)
- [report_timing_summary](#)
- [report_transformed_primitives](#)
- [report_utilization](#)
- [reset_drc](#)
- [reset_msg_config](#)
- [reset_msg_count](#)
- [reset_msg_limit](#)
- [reset_msg_severity](#)
- [reset_ssn](#)
- [reset_timing](#)
- [set_msg_config](#)
- [set_msg_limit](#)
- [set_msg_severity](#)
- [version](#)

SDC:

- [all_clocks](#)
- [all_fanin](#)
- [all_fanout](#)
- [all_inputs](#)
- [all_outputs](#)
- [all_registers](#)

- [create_clock](#)
- [create_generated_clock](#)
- [current_design](#)
- [current_instance](#)
- [get_cells](#)
- [get_clocks](#)
- [get_hierarchy_separator](#)
- [get_nets](#)
- [get_pins](#)
- [get_ports](#)
- [get_timing_arcs](#)
- [get_timing_paths](#)
- [group_path](#)
- [report_operating_conditions](#)
- [reset_operating_conditions](#)
- [set_case_analysis](#)
- [set_clock_groups](#)
- [set_clock_latency](#)
- [set_clock_sense](#)
- [set_clock_uncertainty](#)
- [set_data_check](#)
- [set_disable_timing](#)
- [set_false_path](#)
- [set_hierarchy_separator](#)
- [set_input_delay](#)
- [set_load](#)
- [set_logic_dc](#)
- [set_logic_one](#)
- [set_logic_unconnected](#)
- [set_logic_zero](#)
- [set_max_delay](#)
- [set_max_time_borrow](#)
- [set_min_delay](#)
- [set_multicycle_path](#)
- [set_operating_conditions](#)
- [set_output_delay](#)
- [set_propagated_clock](#)
- [set_units](#)

Simulation:

- [add_bp](#)
- [add_condition](#)
- [add_files](#)
- [add_force](#)
- [checkpoint_vcd](#)
- [close_saif](#)
- [close_sim](#)
- [close_vcd](#)
- [compile_simlib](#)
- [create_fileset](#)
- [current_scope](#)
- [current_sim](#)
- [current_time](#)
- [data2mem](#)
- [delete_fileset](#)
- [describe](#)
- [flush_vcd](#)
- [get_objects](#)
- [get_scopes](#)
- [get_value](#)
- [import_files](#)
- [launch_modelsim](#)
- [launch_xsim](#)
- [limit_vcd](#)
- [log_saif](#)
- [log_vcd](#)
- [log_wave](#)
- [ltrace](#)
- [move_files](#)
- [open_saif](#)
- [open_vcd](#)
- [open_wave_database](#)
- [ptrace](#)
- [read_saif](#)
- [read_vcd](#)
- [remove_bps](#)
- [remove_conditions](#)

- [remove_files](#)
- [remove_forces](#)
- [report_bps](#)
- [report_conditions](#)
- [report_drivers](#)
- [report_objects](#)
- [report_scopes](#)
- [report_simlib_info](#)
- [report_values](#)
- [reset_simulation](#)
- [restart](#)
- [run](#)
- [set_value](#)
- [start_vcd](#)
- [step](#)
- [stop](#)
- [stop_vcd](#)
- [write_sdf](#)
- [write_verilog](#)
- [write_vhdl](#)
- [xsim](#)

SysGen:

- [create_sysgen](#)
- [make_wrapper](#)

Timing:

- [check_timing](#)
- [config_timing_analysis](#)
- [config_timing_corners](#)
- [delete_timing_results](#)
- [get_net_delays](#)
- [get_timing_arcs](#)
- [get_timing_paths](#)
- [report_config_timing](#)
- [report_disable_timing](#)
- [report_exceptions](#)
- [report_timing](#)
- [report_timing_summary](#)
- [reset_timing](#)
- [set_delay_model](#)
- [set_disable_timing](#)
- [update_timing](#)

ToolLaunch:

- [launch_chipscope_analyzer](#)
- [launch_impact](#)
- [launch_modelsim](#)
- [launch_sdk](#)
- [launch_xsim](#)

Tools:

- [link_design](#)
- [list_features](#)
- [load_features](#)
- [opt_design](#)
- [phys_opt_design](#)
- [place_design](#)
- [route_design](#)
- [synth_design](#)

Waveform:

- [add_wave](#)
- [add_wave_divider](#)
- [add_wave_group](#)
- [add_wave_marker](#)
- [add_wave_virtual_bus](#)
- [close_wave_config](#)
- [create_wave_config](#)
- [current_wave_config](#)
- [get_wave_configs](#)
- [open_wave_config](#)
- [save_wave_config](#)

XDC:

- [add_cells_to_pblock](#)
- [all_clocks](#)
- [all_cpus](#)
- [all_dsps](#)
- [all_fanin](#)
- [all_fanout](#)
- [all_ffs](#)
- [all_hsios](#)
- [all_inputs](#)
- [all_latches](#)
- [all_outputs](#)
- [all_rams](#)
- [all_registers](#)
- [create_clock](#)
- [create_generated_clock](#)
- [create_macro](#)
- [create_pblock](#)
- [current_design](#)
- [current_instance](#)
- [delete_macros](#)
- [delete_pblock](#)
- [filter](#)

- [get_cells](#)
- [get_clocks](#)
- [get_generated_clocks](#)
- [get_hierarchy_separator](#)
- [get_iobanks](#)
- [get_macros](#)
- [get_nets](#)
- [get_package_pins](#)
- [get_path_groups](#)
- [get_pblocks](#)
- [get_pins](#)
- [get_ports](#)
- [get_sites](#)
- [get_timing_arcs](#)
- [group_path](#)
- [remove_cells_from_pblock](#)
- [resize_pblock](#)
- [set_case_analysis](#)
- [set_clock_groups](#)
- [set_clock_latency](#)
- [set_clock_sense](#)
- [set_clock_uncertainty](#)
- [set_data_check](#)
- [set_default_switching_activity](#)
- [set_disable_timing](#)
- [set_external_delay](#)
- [set_false_path](#)
- [set_hierarchy_separator](#)
- [set_input_delay](#)
- [set_input_jitter](#)
- [set_load](#)
- [set_logic_dc](#)
- [set_logic_one](#)
- [set_logic_unconnected](#)
- [set_logic_zero](#)
- [set_max_delay](#)
- [set_max_time_borrow](#)
- [set_min_delay](#)
- [set_multicycle_path](#)
- [set_operating_conditions](#)

- `set_output_delay`
 - `set_package_pin_val`
 - `set_power_opt`
 - `set_propagated_clock`
 - `set_property`
 - `set_switching_activity`
 - `set_system_jitter`
 - `set_units`
 - `update_macro`
-

XPS:

- `create_xps`
- `export_hardware`
- `generate_target`
- `get_boards`
- `launch_sdk`
- `list_targets`
- `make_wrapper`
- `reset_target`

Tcl Commands Listed Alphabetically

This chapter contains all SDC and Tcl Commands, arranged alphabetically.

add_bp

Add breakpoint at a line of a HDL source.

Syntax

```
add_bp [-quiet] [-verbose] file_name line_number
```

Returns

Return a new breakpoint object if there is not already a breakpoint set at the specified file line else returns the existing breakpoint object

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>file_name</code>	Filename to add the breakpoint
<code>line_number</code>	Line number of the given file to set the breakpoint

Categories

[Simulation](#)

add_cells_to_pblock

Add cells to a Pblock.

Syntax

```
add_cells_to_pblock [-top] [-add_primitives] [-clear_locs] [-quiet]
[-verbose] pblock [cells...]
```

Returns

Nothing

Usage

Name	Description
[-top]	Add the top level instance; This option can't be used with -cells, or -add_primitives options. You must specify either -cells or -top option.
[-add_primitives]	Assign all the primitives of the specified instances to a pblock
[-clear_locs]	Clear instance location constraints
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>pblock</i>	Pblock to add cells to
<i>[cells]</i>	Cells to add. You can't use this option with -top option. You must specify either -cells or -top option.

Categories

[Floorplan](#), [XDC](#)

Description

Adds specified logic instances to a Pblock. Once cells have been added to a Pblock, you can place the Pblocks onto the fabric of the FPGA using the **resize_pblock** command. The **resize_pblock** command can also be used to manually move and resize pblocks.

You can remove instances from the Pblock using the **remove_cells_from_pblock** command.

Arguments

-top - (Optional) Add the top level instance to create a Pblock for the whole design. You must either specify *cells* or the **-top** option to add objects to the Pblock.

-add_primitives - (Optional) Assign all primitives of the specified instances to a Pblock. This lets you specify a block module and automatically assign all of the instances within that module to the specified Pblock.

Note This option cannot be used with **-top**.

-clear_locs - (Optional) Clear instance location constraints for any cells that are already placed. This allows you to reset the LOC constraint for cells when defining new Pblocks for floorplanning purposes. When this option is not specified, any instances with assigned placement will not be unplaced as they are added to the Pblock.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pblock - The name assigned to the Pblock.

cells - One or more cell objects to add to the specified Pblock.

Note If **-top** is specified, you cannot also specify *cells*

Examples

The following example creates a Pblock called pb_cpuEngine, and then adds all of the primitives found in the cpuEngine module, clearing placement constraints for placed instances:

```
create_pblock pb_cpuEngine
add_cells_to_pblock pb_cpuEngine [get_cells cpuEngine] -add_primitives -clear_locs
```

See Also

- [get_pblocks](#)
- [place_pblocks](#)
- [remove_cells_from_pblock](#)
- [resize_pblock](#)

add_condition

Conditionally execute Tcl commands.

Syntax

```
add_condition [-name arg] [-radix arg] [-quiet]  
[-verbose] condition_expression commands
```

Returns

The condition object created

Usage

Name	Description
[-name]	Assign a unique name (label) to a condition. Multiple conditions cannot be assigned the same name. If no name is specified, then a default label named as condition is automatically created
[-radix]	Specifies which radix to use. Allowed values are: default, dec, bin, oct, hex, unsigned, ascii.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>condition_expression</i>	The condition expression when true executes the given commands
<i>commands</i>	Commands to execute upon condition

Categories

[Simulation](#)

add_drc_checks

Add drc rule check objects to a rule deck.

Syntax

```
add_drc_checks [-of_objects args] [-regexp] [-nocase] [-filter arg]
-ruledeck arg [-quiet] [-verbose] [patterns]
```

Returns

Drc_check

Usage

Name	Description
[-of_objects]	Get 'drc_rule' objects of these types: 'drc_ruledeck'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
-ruledeck	DRC rule deck to modify
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'drc_rule' objects against patterns. Default: *

Categories

[DRC](#), [Object](#)

Description

Add design rule checks to the specified drc_ruledeck object.

A rule deck is a collection of design rule checks grouped for convenience, to be run with the **report_drc** command at different stages of the FPGA design flow, such as during I/O planning or placement. The tool comes with a set of factory defined rule decks, but you can also create new user-defined rule decks with the **create_drc_ruledeck** command.

Use the **get_drc_ruledecks** command to return a list of the currently defined rule decks available for use in the report_drc command.

You can add standard factory defined rule checks to the rule deck, or add user-defined rule checks that were created using the **create_drc_check** command. Use the **get_drc_checks** command to get a list of checks that can be added to a rule deck.

Checks can also be removed from a rule deck using the **remove_drc_checks** command.

Note To temporarily disable a specific DRC rule, use the **set_property** command to set the **IS_ENABLED** property for the rule to false. This will disable the rule from being run in **report_drc**, without having to remove the rule from the rule deck. Use **reset_drc_check** to restore the rule to its default setting

This command returns the list of design rule checks that were added to the rule deck.

Arguments

-of_objects *arg* - (Optional) Add the rule checks of the specified *drc_ruledeck* object to the specified rule deck. This has the effect of copying the rules from one rule deck into another.

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add **.*** to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by the search pattern, based on specified property values. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

-ruledck *arg* - (Required) The name of the rule deck to add the specified design rule checks to.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Add the design rule checks that match the specified patterns to the rule deck. The default pattern is the wildcard '*' which adds all rule checks to the specified rule deck. More than one pattern can be specified to find multiple rule checks based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example adds the rule checks matching the specified search pattern to the `project_rules` rule deck:

```
add_drc_checks -ruledesk project_rules {*DCI* *BUF*}
```

The following example creates a new rule deck called `placer+`, copies all of the rule checks from the `placer_checks` rule deck into the `placer+` rule deck, then adds some additional checks:

```
create_drc_ruledesk placer+  
add_drc_checks -of_objects [get_drc_ruledecks placer_checks] -ruledesk placer+  
add_drc_checks -ruledesk placer+ *IO*
```

The following example adds only the rule checks with a severity of Warning to the rule deck:

```
add_drc_checks -filter {SEVERITY == Warning} -ruledesk warn_only
```

See Also

- [create_drc_check](#)
- [create_drc_ruledesk](#)
- [get_drc_checks](#)
- [get_drc_ruledecks](#)
- [list_property](#)
- [remove_drc_checks](#)
- [report_drc](#)
- [report_property](#)
- [reset_drc_check](#)
- [set_property](#)

add_files

Add sources to the active fileset.

Syntax

```
add_files [-fileset arg] [-norecurse] [-scan_for_includes] [-quiet]  
[-verbose] [files...]
```

Returns

List of file objects that were added

Usage

Name	Description
<code>[-fileset]</code>	Fileset name
<code>[-norecurse]</code>	Do not recursively search in specified directories
<code>[-scan_for_includes]</code>	Scan and add any included files found in the fileset's RTL sources
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[files]</code>	Name of the files and/or directories to add. Must be specified if <code>-scan_for_includes</code> is not used.

Categories

[Project](#), [Simulation](#)

Description

Adds one or more source files, or the source file contents of one or more directories, to the specified fileset in the current project.

The Vivado tool does not read the contents of a file automatically when the file is added to the project with **add_files**, but rather reads the file contents when they are needed. For instance, a constraints file is not read when added to the design until needed by synthesis, timing, or implementation. To read the file at the time of adding it to the design, use the **read_xxx** command instead.

Note When running the Vivado tool in Non Project mode, in which there is no project file to maintain and manage the various project source files, you should use the **read_xxx** commands to read the contents of source files into the in-memory design. Refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) for more information on Non Project mode.

The **add_files** command is different from the **import_files** command, which copies the file into the local project folders as well as adding them to the specified fileset. This command only adds them by reference to the specified fileset.

Arguments

-fileset *name* - (Optional) The fileset to which the specified source files should be added. An error is returned if the specified fileset does not exist. If no fileset is specified the files are added to the source fileset by default.

-norecurse - (Optional) Do not recurse through subdirectories of any specified directories. Without this argument, the tool searches through any subdirectories for additional source files that can be added to a project.

-scan_for_includes - (Optional) Scan Verilog source files for any **'include** statements and add these referenced files to the specified fileset. By default, **'include** files are not added to the fileset.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Optional) One or more file names or directory names to be added to the fileset. If a directory name is specified, all valid source files found in the directory, and in subdirectories of the directory, are added to the fileset.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example adds a file called `rtl.v` to the current project:

```
add_files rtl.v
```

In the preceding example the tool looks for the `rtl.v` file in the current working directory since no file path is specified, and the file is added to the source fileset as a default since no fileset is specified.

The following example adds a file called **top.ucf** to the **constrs_1** constraint fileset, as well as any appropriate source files found in the **project_1** directory, and its subdirectories:

```
add_files -fileset constrs_1 -quiet c:/Design/top.ucf c:/Design/project_1
```

In addition, the tool ignores any command line errors because the **-quiet** argument was specified.

If the **-norecurse** option had been specified then only constraint files found in the **project_1** directory would have been added, but subdirectories would not be searched.

The following example adds an existing IP core file to the current project:

```
add_files -norecurse C:/Data/ip/c_addsub_v11_0_0.xci
```

Note Use the **import_ip** command to import the IP file into the local project folders

The following example adds an existing Embedded Processor sub-design into the current project:

```
add_files C:/Data/dvi_tpg_demo_ORG/system.xmp
```

Note Use the **create_xps** command to create a new Embedded Processor using Xilinx Platform Studio (XPS)

The following example adds an existing DSP module, created in System Generator, into the current project:

```
add_files C:/Data/model1.mdl
```

Note Use the **create_sysgen** command to use System Generator to create a new DSP module

See Also

- [create_sysgen](#)
- [create_xps](#)
- [import_files](#)
- [import_ip](#)
- [read_ip](#)
- [read_verilog](#)
- [read_vhdl](#)
- [read_xdc](#)

add_force

Force value of signal, wire, or reg to a specified value.

Syntax

```
add_force [-radix arg] [-repeat_every arg] [-cancel_after arg]
[-quiet] [-verbose] hdl_object values...
```

Returns

The force objects added

Usage

Name	Description
<code>[-radix]</code>	Specifies which radix to use. Allowed values are: default, dec, bin, oct, hex, unsigned, ascii
<code>[-repeat_every]</code>	Repeat every time duration
<code>[-cancel_after]</code>	Cancel after time offset
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>hdl_object</code>	Specifies the object upon which to add a force
<code>values</code>	Adds a value and time offset to the force: {value [time_offset] }

Categories

[Simulation](#)

Description

Force the value of a signal, wire, or reg to a certain value during simulation.

The **add_force** command has the same effect as the Verilog **force/release** commands in the test bench or the module definition. It forces an HDL object to hold the specified value for the specified time, or until released by the **-cancel_after** option, or the **remove_forces** command.

Important! If there are Verilog **force/release** statements on an HDL object in the test bench or module, these commands are overridden by the Tcl **add_force** command. When the Tcl force expires or is released, the HDL object resumes normal operation in the simulation, including the application of any Verilog forces

This command returns the name of the force object created, or returns an error if the command failed. The name of the returned force object is important when using the **remove_forces** command, and should be captured in a Tcl variable for later recall, as shown in the examples.

Arguments

-radix *arg* - (Optional) The radix used when specifying the *values*. Supported radix values are: **default**, **dec**, **bin**, **oct**, **hex**, **unsigned**, and **ascii**. The default radix is binary, unless the specified HDL object type has an overriding radix defined.

-repeat_every *arg* - (Optional) Causes the **add_force** to repeat over some specified increment of time. This can be used to create a recurring force on the specified *hdl_object*.

Note The specified time must be greater than the time period defined by any *{value time}* pairs defined by *values*, or an error will be returned

-cancel_after *arg* - (Optional) Cancels the force effect after the specified period of time from the **current_time**. This has the same effect as applying the **remove_forces** command after the specified period of time.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

hdl_object - (Required) Specifies a single HDL object to force the value of. The object can be specified by name, or can be returned as an object by the **get_objects** command. Valid objects are signal, wire, and reg.

values - (Required) The value to force the HDL object to. A single value can be specified, and the value will be held during simulation until the force is removed either through the use of the **-cancel_after** option, or through the use of the **remove_forces** command.

The specified *value* depends on the type of the *hdl_object*. HDL object types include: "logic", floating point, VHDL enumerated, and VHDL integral. For all but "logic" the -radix option is ignored.

- "Logic" does not refer to an actual HDL object type, but means any object whose values are similar to those of VHDL std_logic, such as:
 - the Verilog implicit 4-state bit type,
 - the VHDL bit and std_logic predefined types,
 - any VHDL enumeration type which is a subset of std_logic, including the character literals 0 and 1.
- For logic types the value depends on the radix:
 - If the specified value has fewer bits than the logic type expects, the value is zero extended, but not sign extended, to match the expected length.
 - If the specified value has more bits than the logic type expects, the extra bits on the MSB side should all be zeros, or the Vivado simulator will return a "size mismatch" error.
- Accepted values for floating point objects are floating point values.
- The accepted value for non-logic VHDL enumerated types is a scalar value from the enumerated set of values, without single quotes in the case of characters.
- Accepted values for VHDL integral types is a signed decimal integer in the range accepted by the type.

The *value* can also be specified as *{value time}* pairs, which forces the HDL object to hold a specified *value* for a specified period of *time* from the current time, then hold the next *value* for the next period of *time*, until the end of the *{value-time}* pairs.

Note In *{value time}* pairs, the *time* is optional in the first pair, and will be assumed to be 0 if it is not specified. In all subsequent *{value time}* pairs, the *time* is required

The *time* specified in *{value time}* pairs is defined relative to the current simulation time, and indicates a period of time from the **current_time**. For example, if the current simulation time is 1000 ns, a *time* of 20 ns defines a period from the current time to 1020 ns.

Restriction The *times* must be specified in increasing order on the simulation time line, and may not be repeated, or an error will occur.

The *time* is specified in the default TIME_UNIT of the current simulation, or can be specified with the time unit included, with no white space. Valid units of time are: fs, ps, ns, us, ms, or s. A time of 50 defines a period of 50 ns (the default). A time of 50ps defines a period of 50 picoseconds.

Examples

The following example forces the reset signal high at 300 nanoseconds, using the default radix, and captures the name of the returned force object in a Tcl variable which can be used to later remove the force:

```
set for10 [ add_force reset 1 300 ]
```

The following example shows the use of *{value time}* pairs, repeated periodically, and canceled after a specified time.

```
add_force mySig {0} {1 50 } {0 100} {1 150 } -repeat_every 200 -cancel_after 10000
```

Note In the preceding example, the first *{value time}* pair does not include a time. This indicates that the specified value, 0, is applied at time 0 (the **current_time**)

See Also

- [current_time](#)
- [get_objects](#)
- [remove_forces](#)

add_wave

Add new waves.

Syntax

```
add_wave [-into args] [-at_wave args] [-after_wave args]
[-before_wave args] [-reverse] [-radix arg] [-color arg] [-name arg]
[-recursive] [-r] [-regexp] [-nocase] [-quiet] [-verbose] items...
```

Returns

The new waves

Usage

Name	Description
[-into]	into: the wave configuration, group, or virtual bus into which the new wave object(s) will be inserted. If no <code>-*_row</code> or <code>-*_wave</code> option is specified, the new wave object(s) will be added to the end of the <code>-into</code> object. If such an option is specified in addition to <code>-into</code> , it is an error if the row number is incompatible with the <code>-into</code> object. If <code>wcfgGroupVbusObj</code> is a string instead of an object, it is treated as the name of a group in the current WCFG. If no such group is found, the names of the virtual buses of the current WCFG are searched. If still not found, the names of all WCFG objects are searched. If no <code>-into</code> object is specified, the current wave configuration is assumed.
[-at_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If waveObj is a string, it is treated as the display name of a wave object
[-after_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If waveObj is a string, it is treated as the display name of a wave object
[-before_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If waveObj is a string, it is treated as the display name of a wave object
[-reverse]	reverse: set the <code>is_reversed</code> property of the new wave object(s) to true
[-radix]	radix radix: set the <code>radix</code> property of the new wave object(s) to radixAllowed values are: default, dec, bin, oct, hex, unsigned, ascii
[-color]	color color: set the <code>color</code> property of the new wave object(s) to color
[-name]	name customName: set the <code>display_name</code> property of the new wave object to customName. It is an error for there to be more than one new wave object being created.

Name	Description
[-recursive]	recursive: if the design object is a scope, this option specifies that wave objects for all design objects under that scope should be created
[-r]	recursive: if the design object is a scope, this option specifies that wave objects for all design objects under that scope should be created
[-regex]	regex: using regular expressions, search design objects from which to create wave objects by design object name. The application supplying the design objects determines how the match is to be performed. items must be strings.
[-nocase]	nocase: Only when regex is used, perform a case insensitive match
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>items</i>	Creates one or more new design-based wave objects

Categories

Waveform

Description

The **add_wave** command creates one or more new design-based wave objects.

This command returns the name of the newly-created wave object(s).

Note This command can only be used when running a simulation. At a minimum, you must specify an **item**, which is an HDL object (signal) within the simulation project. In the Vivado interface, the object would display in the Objects Window.

Arguments

-into *wcfgGroupVbusObj* - (Optional) Specifies the wave configuration, group, or virtual bus into which the new wave object(s) are inserted. If *wcfgGroupVbusObj* is a string instead of an object, it is treated as the name of a group in the current WCFG. If no such group is found, the tool searches the names of the virtual buses of the current WCFG. If still not found, the tool searches the names of all WCFG objects. If no **-into** object is specified, the current wave configuration is assumed.

-at_wave *waveObj* - (Optional) Adds a wave object at a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-after_wave *waveObj* - (Optional) Adds a wave object after a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-before_wave *waveObj* - (Optional) Adds a wave object before a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-reverse - (Optional) Sets the **IS_REVERSED** property of the new wave object(s) to **true**.

-radix *arg* - (Optional) Sets the radix property of the new wave object(s) to radix. Allowed values are: **default**, **dec**, **bin**, **oct**, **hex**, **unsigned**, and **ascii**.

-color *arg* - (Optional) Sets the color property of the new wave object(s) to color, which is either a pre-defined color name or a color specified by a six-digit RGB format (RRGGBB).

-name *arg* - (Optional) Sets the **DISPLAY_NAME** property of the new wave object to the specified name. It is an error for there to be more than one new wave object being created.

-recursive | **-r** - (Optional) If *items* specifies a scope, this option specifies that all sub-scopes of that scope should also be added.

-regexp - (Optional) Specifies that *items* are written as regular expressions. Xilinx regular expression commands are always anchored to the start of the search string. You can add "." to the beginning of the search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

items - (Required) Add waves for the specified HDL objects in the current simulation.

Examples

Add a clk to the existing waveform configuration:

```
add_wave clk
clk
```

Add the dout_tvalid signal from the rsb_design_testbench to the existing simulation waveform configuration:

```
add_wave dout_tvalid
/rsb_design_testbench/dout_tvalid
```

See Also

- [add_wave_divider](#)
- [add_wave_group](#)
- [add_wave_marker](#)
- [add_wave_virtual_bus](#)

add_wave_divider

Add a new divider.

Syntax

```
add_wave_divider [-into args] [-at_wave args] [-after_wave args]
[-before_wave args] [-color arg] [-quiet] [-verbose] [name]
```

Returns

The new divider

Usage

Name	Description
[-into]	into: the wave configuration or group into which the new wave object(s) will be inserted. If no <code>-*_row</code> or <code>-*_wave</code> option is specified, the new wave object(s) will be added to the end of the <code>-into</code> object. If such an option is specified in addition to <code>-into</code> , it is an error if the row number is incompatible with the <code>-into</code> object. If <code>wcfgGroupVbusObj</code> is a string instead of an object, it is treated as the name of a group in the current WCFG. If no such group is found, the names of the virtual buses of the current WCFG are searched. If still not found, the names of all WCFG objects are searched. If no <code>-into</code> object is specified, the current wave configuration is assumed.
[-at_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-after_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-before_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-color]	color color: set the color property of the new wave object(s) to color Default: default
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[name]	Creates a divider whose display name is name Default: new_divider

Categories

Waveform

Description

Creates a wave divider in the wave form viewer. The wave divider can be used to separate groups of related objects, for easier viewing.

The wave divider can be added into a specified or current waveform configuration at the specified location. If no location is specified the wave divider is inserted at the end of the waveform configuration.

This command returns the name of the newly-created wave divider.

Note This command can only be used when running a simulation.

Arguments

-into *wcfgGroupVbusObj* - (Optional) Specifies the wave configuration, group, or virtual bus into which the new wave object(s) are inserted. If *wcfgGroupVbusObj* is a string instead of an object, it is treated as the name of a group in the current WCFG. If no such group is found, the tool searches the names of the virtual buses of the current WCFG. If still not found, the tool searches the names of all WCFG objects. If no **-into** object is specified, the current wave configuration is assumed.

-at_wave *waveObj* - (Optional) Adds a wave divider at a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-after_wave *waveObj* - (Optional) Adds a wave divider after a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-before_wave *waveObj* - (Optional) Adds a wave divider before a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-color *arg* - (Optional) Sets the color property of the new wave object(s) to a pre-defined color name or a color specified by a six-digit RGB format (RRGGBB).

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Optional) Creates a divider with the specified display name. The default name is *new_divider*.

Examples

The following example inserts a wave divider named Div1, after the CLK wave object:

```
add_wave_divider -after_wave CLK Div1
```

See Also

- [add_wave](#)
- [add_wave_group](#)
- [add_wave_marker](#)
- [add_wave_virtual_bus](#)

add_wave_group

Add a new group.

Syntax

```
add_wave_group [-into args] [-at_wave args] [-after_wave args]
[-before_wave args] [-quiet] [-verbose] [name]
```

Returns

The new group

Usage

Name	Description
[-into]	into: the wave configuration, or group into which the new wave object(s) will be inserted. If no <code>-*_row</code> or <code>-*_wave</code> option is specified, the new wave object(s) will be added to the end of the <code>-into</code> object. If such an option is specified in addition to <code>-into</code> , it is an error if the row number is incompatible with the <code>-into</code> object. If <code>wcfgGroupVbusObj</code> is a string instead of an object, it is treated as the name of a group in the current WCFG. If no such group is found, the names of the virtual buses of the current WCFG are searched. If still not found, the names of all WCFG objects are searched. If no <code>-into</code> object is specified, the current wave configuration is assumed.
[-at_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-after_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-before_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[name]	Creates a group wave object whose display name is <code>name</code> Default: <code>new_group</code>

Categories

Waveform

Description

Creates a wave group into a specified or current waveform configuration. New wave objects and wave_dividers can be added into the wave group to build up the waveform display.

The wave group can be inserted at a specified location. If no location is specified the group is inserted at the end of the specified waveform configuration.

The command returns the name of the newly created wave group object.

Note This command can only be used when running a simulation.

Arguments

-into *wcfgGroupVbusObj* - (Optional) Specifies the wave configuration, group, or virtual bus into which the new wave object(s) are inserted. If *wcfgGroupVbusObj* is a string instead of an object, it is treated as the name of a group in the current WCFG. If no such group is found, the tool searches the names of the virtual buses of the current WCFG. If still not found, the tool searches the names of all WCFG objects. If no **-into** object is specified, the current wave configuration is assumed.

-at_wave *waveObj* - (Optional) Adds a wave group at a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-after_wave *waveObj* - (Optional) Adds a wave group after a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-before_wave *waveObj* - (Optional) Adds a wave group before a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Optional) Creates a wave group with the specified display name. The default name is *new_group*.

Examples

Add a clk to the existing waveform configuration:

```
add_wave_group clk
group10
```

See Also

- [add_wave](#)
- [add_wave_divider](#)
- [add_wave_marker](#)
- [add_wave_virtual_bus](#)

add_wave_marker

Create a new wave marker.

Syntax

```
add_wave_marker [-into arg] [-name arg] [-quiet] [-verbose] [time]  
[unit]
```

Returns

The new created marker

Usage

Name	Description
[-into]	into: a WCFG object or name of a WCFG object in which to create the marker
[-name]	name: the optional name of the marker
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>time</i>]	Creates a new marker at the specified time Default: 0
[<i>unit</i>]	

Categories

[Waveform](#)

Description

Creates a wave marker at the specified time and of the specified name in the current waveform configuration.

This command returns nothing.

Note This command can only be used when running a simulation.

Arguments

-into *wcfg* - (Optional) Specifies the WCFG object into which the new wave marker is inserted. If **-into** is not specified, the wave marker is added to the current wave configuration.

-name *arg* - (Optional) Creates a marker with the specified name. The default name is `new_marker`.

time - (Optional) Is the simulation runtime within the waveform at which to set the marker. The default is time 0.

unit - (Optional) Is the time unit. Allowable units are **s**, **ms**, **us**, **ns**, and **ps**. The default is the time unit used in the specified waveform configuration.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

Add a marker to the existing waveform configuration at 500ns:

```
add_wave_marker 500 ns
```

See Also

- [add_wave](#)
- [add_wave_divider](#)
- [add_wave_group](#)
- [add_wave_virtual_bus](#)

add_wave_virtual_bus

Add a new virtual bus.

Syntax

```
add_wave_virtual_bus [-into args] [-at_wave args] [-after_wave args]
[-before_wave args] [-reverse] [-radix arg] [-color arg] [-quiet]
[-verbose] [name]
```

Returns

The new virtual bus

Usage

Name	Description
[-into]	into: the wave configuration, group, or virtual bus into which the new wave object(s) will be inserted. If no <code>-*_row</code> or <code>-*_wave</code> option is specified, the new wave object(s) will be added to the end of the <code>-into</code> object. If such an option is specified in addition to <code>-into</code> , it is an error if the row number is incompatible with the <code>-into</code> object. If <code>wcfgGroupVbusObj</code> is a string instead of an object, it is treated as the name of a group in the current WCFG. If no such group is found, the names of the virtual buses of the current WCFG are searched. If still not found, the names of all WCFG objects are searched. If no <code>-into</code> object is specified, the current wave configuration is assumed.
[-at_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-after_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-before_wave]	waveObj: similar to <code>-*_row</code> , except located using a wave object instead of a row number. If <code>waveObj</code> is a string, it is treated as the display name of a wave object
[-reverse]	reverse: set the <code>is_reversed</code> property of the new wave object(s) to true
[-radix]	radix radix: set the <code>radix</code> property of the new wave object(s) to <code>radixAllowed</code> values are: default, dec, bin, oct, hex, unsigned, ascii
[-color]	color color: set the <code>color</code> property of the new wave object(s) to color Default: default
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Name	Description
[name]	Creates a new virtual bus whose display name is name Default: new_divider

Categories

Waveform

Description

The **add_wave_virtual_bus** command creates a new virtual bus. The command inserts the virtual bus by specified name where specified or by default at the end of the existing waveform. It returns a vb### for the newly-created virtual bus.

Note This command can only be used when running a simulation. At a minimum, you must specify an **name**, which is the name of the new virtual bus

Arguments

-into *wcfgGroupVbusObj* - (Optional) Specifies the wave configuration, group, or virtual bus into which the new wave object(s) are inserted. If *wcfgGroupVbusObj* is a string instead of an object, it is treated as the name of a group in the current WCFG. If no such group is found, the tool searches the names of the virtual buses of the current WCFG. If still not found, the tool searches the names of all WCFG objects. If no **-into** object is specified, the current wave configuration is assumed.

-at_wave *waveObj* - (Optional) Adds a wave object at a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-after_wave *waveObj* - (Optional) Adds a wave object after a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-before_wave *waveObj* - (Optional) Adds a wave object before a specified wave object. If *waveObj* is a string, it is treated as the display name of a wave object.

-reverse - (Optional) Sets the **IS_REVERSED** property of the new wave object(s) to **true**.

-radix value - (Optional) Sets the radix property of the new wave object(s) to radix. Allowed values are: **default**, **dec**, **bin**, **oct**, **hex**, **unsigned**, and **ascii**.

-color arg - (Optional) Sets the color property of the new wave object(s) to the specified color, which can be a pre-defined color name or a color specified by a six-digit RGB format (RRGGBB).

-name customName - (Optional) Sets the **display_name** property of the new wave object to *customName*.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

Add a virtual bus of the name **dout_tvalid** to the existing waveform configuration:

```
add_wave_virtual_bus dout_tvalid  
vbus200
```

See Also

- [add_wave_divider](#)
- [add_wave_group](#)
- [add_wave_marker](#)
- [add_wave](#)

all_clocks

Get a list of all clocks in the current design.

Syntax

```
all_clocks [-quiet] [-verbose]
```

Returns

List of clock objects

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

Returns a list of all clocks that have been declared in the current design. To get a list of specific clocks in the design, use the **get_clocks** command.

Clocks can be defined by using the **create_clock** or **create_generated_clock** commands.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example shows all clocks in the sample CPU netlist project:

```
% all_clocks
cpuClk wbClk usbClk phy_clk_pad_0_i phy_clk_pad_1_i fftClk
```

The following example applies the **set_propagated_clock** command to all clocks, and also demonstrates how the returned list (**all_clocks**) can be passed to another command:

```
% set_propagated_clock [all_clocks]
```

See Also

- [create_clock](#)
- [create_generated_clock](#)
- [get_clocks](#)
- [set_propagated_clock](#)

all_cpus

Get a list of cpu cells in the current design.

Syntax

```
all_cpus [-quiet] [-verbose]
```

Returns

List of cpu cell objects

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

XDC

Description

Returns a list of all CPU cell objects in the current design. Creates a list of all the CPU cell objects that have been declared in the current design.

The **all_cpus** command is scoped to return the objects hierarchically, from the top-level of the design or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command.

Note This command returns a list of CPU cell objects

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns all CPU objects in the current design:

```
all_cpus
```

The following example shows how the list returned can be passed to another command:

```
set_false_path -from [all_cpus] -to [all_registers]
```

See Also

- [all_dsps](#)
- [all_hsios](#)
- [all_registers](#)
- [current_instance](#)
- [get_cells](#)
- [set_false_path](#)

all_dsps

Get a list of dsp cells in the current design.

Syntax

```
all_dsps [-quiet] [-verbose]
```

Returns

List of dsp cell objects

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[XDC](#)

Description

Returns a list of all DSP cell objects that have been declared in the current design.

The **all_dsps** command is scoped to return the objects hierarchically, from the top-level of the design or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns a list of all DSPs defined in the current design:

```
all_dsps
```


The following example shows how the list returned can be passed to another command:

```
set_false_path -from [all_dsps] -to [all_registers]
```

See Also

- [all_cpus](#)
- [all_hsios](#)
- [all_registers](#)
- [current_instance](#)
- [get_cells](#)
- [set_false_path](#)

all_fanin

Get a list of pins or cells in fanin of specified sinks.

Syntax

```
all_fanin [-startpoints_only] [-flat] [-only_cells] [-levels arg]
[-pin_levels arg] [-trace_arcs arg] [-quiet] [-verbose] to
```

Returns

List of cell or pin objects

Usage

Name	Description
<code>[-startpoints_only]</code>	Find only the timing startpoints
<code>[-flat]</code>	Hierarchy is ignored
<code>[-only_cells]</code>	Only cells
<code>[-levels]</code>	Maximum number of cell levels to traverse: Value ≥ 0 Default: 0
<code>[-pin_levels]</code>	Maximum number of pin levels to traverse: Value ≥ 0 Default: 0
<code>[-trace_arcs]</code>	Type of network arcs to trace: Values: timing, enabled, all
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>to</code>	List of sink pins, ports, or nets

Categories

[SDC](#), [XDC](#)

Description

Returns a list of port, pin or cell objects in the fan-in of the specified sinks.

The **all_fanin** command is scoped to return objects from current level of the hierarchy of the design, either from the top-level or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command. To return the fan-in across all levels of the hierarchy, use the **-flat** option.

Arguments

-startpoints_only - (Optional) Find only the timing start points. When this option is used, none of the intermediate points in the fan-in network are returned. This option can be used to identify the primary driver(s) of the sinks.

-flat - (Optional) Ignore the hierarchy of the design. By default, only the objects at the same level of hierarchy as the sinks are returned. When using this option, all the objects in the fan-in network of the sinks are considered, regardless of hierarchy.

-only_cells - (Optional) Return only the cell objects which are in the fan-in path of the specified sinks. Do not return pins or ports.

-levels *value* - (Optional) Maximum number of cell levels to traverse. The default value is 0.

-pin_levels *value* - (Optional) Maximum number of pin levels to traverse. The default value is 0.

-trace_arcs *value* - (Optional) Type of network arcs to trace. Valid values are "timing", "enabled", and "all"

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

to - (Required) The pins, ports, or nets from which you want the fan-in objects reported.

Examples

The following example lists the timing fan-in of the led_pins output port:

```
all_fanin [get_ports led_pins[*] ]
```

The following example traces back from the clock pin of the specified flip-flop to the clock source (an MMCM output pin in this example):

```
all_fanin -flat -startpoints_only [get_pins cmd_parse_i0/prescale_reg[7]/C]
```

See Also

- [all_fanout](#)
- [current_instance](#)
- [get_cells](#)
- [get_pins](#)
- [get_ports](#)

all_fanout

Get a list of pins or cells in fanout of specified sources.

Syntax

```
all_fanout [-endpoints_only] [-flat] [-only_cells] [-levels arg]
[-pin_levels arg] [-trace_arcs arg] [-quiet] [-verbose] from
```

Returns

List of cell or pin objects

Usage

Name	Description
<code>[-endpoints_only]</code>	Find only the timing endpoints
<code>[-flat]</code>	Hierarchy is ignored
<code>[-only_cells]</code>	Only cells
<code>[-levels]</code>	Maximum number of cell levels to traverse: Value ≥ 0 Default: 0
<code>[-pin_levels]</code>	Maximum number of pin levels to traverse: Value ≥ 0 Default: 0
<code>[-trace_arcs]</code>	Type of network arcs to trace: Values: timing, enabled, all
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>from</code>	List of source pins, ports, or nets

Categories

[SDC](#), [XDC](#)

Description

Returns a list of port, pin, or cell objects in the fanout of the specified sources.

The **all_fanout** command is scoped to return objects from current level of the hierarchy of the design, either from the top-level or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command. To return the fanout across all levels of the hierarchy, use the **-flat** option.

Arguments

-endpoints_only - (Optional) Find only the timing endpoints. When this option is used, none of the intermediate points in the fan-out network are returned. This option can be used to identify the primary loads of the drivers.

-flat - (Optional) Ignore the hierarchy of the design. By default, only the objects at the same level of hierarchy as the sinks are returned. When using this option, all the objects in the fan-out network of the drivers are considered, regardless of hierarchy.

-only_cells - (Optional) Return only the cell objects in the fanout path of the specified sources.

-levels *value* - (Optional) Maximum number of cell levels to traverse. The default value is 0.

-pin_levels *value* - (Optional) Maximum number of pin levels to traverse. The default value is 0.

-trace_arcs *value* - (Optional) Type of network arcs to trace. Valid values are "timing", "enabled", and "all"

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

from - (Required) The source ports, pins, or nets from which to list the objects in the fanout path.

Examples

The following example gets the fanout for all input ports in the design:

```
all_fanout [all_inputs]
```

See Also

- [all_fanin](#)
- [current_instance](#)
- [get_cells](#)
- [get_pins](#)
- [get_ports](#)

all_ffs

Get a list of flip flop cells in the current design.

Syntax

```
all_ffs [-quiet] [-verbose]
```

Returns

List of flip flop cell objects

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[XDC](#)

Description

Returns a list of all flip flop instances in the current design.

The **all_ffs** command is scoped to return the objects hierarchically, from the top-level of the design or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the count of all flops in the design, then returns the count of all flops in the fftEngine module:

```
current_instance
INFO: [Vivado 12-618] Current instance is the top level of design 'netlist_1'.
top
llength [all_ffs]
15741
current_instance fftEngine
fftEngine
llength [all_ffs]
1519
```

The following example reports the currently assigned properties on the specified flop:

```
report_property [lindex [all_ffs] 2 ]
```

See Also

- [all_latches](#)
- [all_registers](#)
- [current_instance](#)
- [get_cells](#)
- [report_property](#)

all_hsios

Get a list of hsio cells in the current design.

Syntax

```
all_hsios [-quiet] [-verbose]
```

Returns

List of hsio cell objects

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[XDC](#)

Description

Returns a list of all High Speed IO (HSIO) cell objects that have been declared in the current design. These HSIO cell objects can be assigned to a variable or passed into another command.

The **all_hsios** command is scoped to return the objects hierarchically, from the top-level of the design or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns all HSIO objects in the current design:

```
all_hsios
```


The following example shows how the list returned can be directly passed to another command:

```
set_false_path -from [all_hsios] -to [all_registers]
```

See Also

- [all_cpus](#)
- [all_dsps](#)
- [all_registers](#)
- [get_cells](#)
- [set_false_path](#)

all_inputs

Get a list of all input ports in the current design.

Syntax

```
all_inputs [-quiet] [-verbose]
```

Returns

List of port objects

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

Returns a list of all input port objects in the current design.

The **all_inputs** command is scoped to return the objects hierarchically, from the top-level of the design or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns all input port objects in the current design:

```
all_inputs
```

The following example shows how the list returned can be passed to another command:

```
set_input_delay 5 -clock REFCLK [all_inputs]
```

See Also

- [all_clocks](#)
- [all_outputs](#)
- [current_instance](#)
- [get_clocks](#)
- [get_ports](#)
- [set_input_delay](#)

all_latches

Get a list of all latch cells in the current design.

Syntax

```
all_latches [-quiet] [-verbose]
```

Returns

List of latch cell objects

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[XDC](#)

Description

Returns a list of all latches that have been declared in the current design.

The **all_latches** command is scoped to return the objects hierarchically, from the top-level of the design or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns a list of all latches in the current design:

```
all_latches
```

The following example shows how the list returned can be passed to another command:

```
set_false_path -from [all_mults] -to [all_latches]
```

See Also

- [all_ffs](#)
- [all_registers](#)
- [current_instance](#)
- [get_cells](#)
- [set_false_path](#)

all_outputs

Get a list of all output ports in the current design.

Syntax

```
all_outputs [-quiet] [-verbose]
```

Returns

List of port objects

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

Returns a list of all output port objects that have been declared in the current design.

The **all_outputs** command is scoped to return the objects hierarchically, from the top-level of the design or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns all the output ports in the current design:

```
all_outputs
```

The following example sets the output delay for all outputs in the design:

```
set_output_delay 5 -clock REFCLK [all_outputs]
```

See Also

- [all_inputs](#)
- [current_instance](#)
- [get_ports](#)
- [set_output_delay](#)

all_rams

Get a list of ram cells in the current design.

Syntax

```
all_rams [-quiet] [-verbose]
```

Returns

List of ram cell objects

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

XDC

Description

Returns a list of all the RAM cell objects present in the current instance, including Block RAMS, Block RAM FIFOs, and Distributed RAMS. These RAM cell objects can be assigned to a variable or passed into another command.

The **all_rams** command is scoped to return the objects hierarchically, from the top-level of the design or from the level of the current instance. By default the current instance is defined as the top level of the design, but can be changed by using the **current_instance** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns all RAM objects in the design:

```
all_rams
```

The following example sets the current instance, and returns all RAM objects hierarchically from the level of the current instance:

```
current_instance usbEngine0  
all_rams
```

See Also

- [all_clocks](#)
- [all_cpus](#)
- [all_dsps](#)
- [all_fanin](#)
- [all_fanout](#)
- [all_ffs](#)
- [all_hsios](#)
- [all_inputs](#)
- [all_latches](#)
- [all_outputs](#)
- [all_registers](#)
- [current_instance](#)
- [get_cells](#)

all_registers

Get a list of register cells or pins in the current design.

Syntax

```
all_registers [-clock args] [-rise_clock args] [-fall_clock args]
[-cells] [-data_pins] [-clock_pins] [-async_pins] [-output_pins]
[-level_sensitive] [-edge_triggered] [-no_hierarchy] [-quiet]
[-verbose]
```

Returns

List of cell or pin objects

Usage

Name	Description
<code>[-clock]</code>	Consider registers of this clock
<code>[-rise_clock]</code>	Consider registers triggered by clock rising edge
<code>[-fall_clock]</code>	Consider registers triggered by clock falling edge
<code>[-cells]</code>	Return list of cells (default)
<code>[-data_pins]</code>	Return list of register data pins
<code>[-clock_pins]</code>	Return list of register clock pins
<code>[-async_pins]</code>	Return list of async preset/clear pins
<code>[-output_pins]</code>	Return list of register output pins
<code>[-level_sensitive]</code>	Only consider level-sensitive latches
<code>[-edge_triggered]</code>	Only consider edge-triggered flip-flops
<code>[-no_hierarchy]</code>	Only search the current instance
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

Returns a list of sequential register cells or register pins in the current design.

The list of returned objects can be limited by the use of the arguments described below. You can limit the list of registers returned to a specific clock or clocks, or to registers triggered by the rising or falling edge of a specified clock.

You can also get a list of the pins of collected registers instead of the register objects by specifying one or more of the pin arguments.

Arguments

-cells - (Optional) Return a list of register cell objects as opposed to a list of pin objects. This is the default behavior of the command.

-clock *args* - (Optional) Return a list of all registers whose clock pins are in the fanout of the specified clock.

-rise_clock *args* - (Optional) Return a list of registers triggered by the rising edge of the specified clocks.

-fall_clock *args* - (Optional) Return a list of registers triggered by the falling edge of the specified clocks.

Note Do not combine **-clock**, **-rise_clock**, and **-fall_clock** in the same command.

-level_sensitive - (Optional) Return a list of the level-sensitive registers or latches.

-edge_triggered - (Optional) Return a list of the edge-triggered registers or flip-flops.

-data_pins - (Optional) Return a list of data pins of all registers in the design, or of the registers that meet the search requirement.

-clock_pins - (Optional) Return a list of clock pins of the registers that meet the search requirement.

-async_pins - (Optional) Limit the search to asynchronous pins of the registers that meet the search requirement.

-output_pins - (Optional) Return a list of output pins of the registers that meet the search requirement.

Note Use the ***_pins** arguments separately. If you specify multiple arguments, only one argument is applied in the following order of precedence: **-data_pins**, **-clock_pins**, **-async_pins**, **-output_pins**.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns a list of registers that are triggered by the falling edge of any clock in the design:

```
all_registers -fall_clock [all_clocks]
```

The following example shows how the list returned can be passed to another command:

```
set_min_delay 2.0 -to [all_registers -clock CCLK -data_pins]
```

See Also

- [all_clocks](#)
- [set_msg_limit](#)
- [set_min_delay](#)

apply_bd_automation

Runs an automation rule on an IPI object.

Syntax

```
apply_bd_automation [-config args] -rule arg [-quiet]  
[-verbose] objects...
```

Returns

Returns success or failure

Usage

Name	Description
[-config]	List of parameter value pairs
-rule	Rule ID string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	The objects to run the automation rule on

Categories

[IPIntegrator](#)

archive_project

Archive the current project.

Syntax

```
archive_project [-force] [-exclude_run_results] [-quiet] [-verbose]  
[file]
```

Returns

True

Usage

Name	Description
[-force]	Overwrite existing archived file
[-exclude_run_results]	Exclude run results from the archive
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[file]</i>	Name of the archive file

Categories

[Project](#)

Description

Archives a project to store as backup, or to encapsulate the design and send it to a remote site. The tool parses the hierarchy of the design, copies the required source files, include files, and remote files from the library directories, copies the constraint files, copies the results of the various synthesis, simulation, and implementation runs, and then creates a ZIP file of the project.

Tip An alternative method of archiving the project is using **write_project_tcl** to create a Tcl script that will recreate the project in its current form

Arguments

-force - (Optional) Overwrite an existing ZIP file of the same name. If the ZIP file exists, the tool returns an error unless the **-force** argument is specified.

-exclude_run_results - (Optional) Exclude the results of any synthesis or implementation runs. This command can greatly reduce the size of a project archive.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Optional) The name of the ZIP file to be created by the **archive_project** command. If *file* is not specified, a ZIP file with the same name as the project is created.

Examples

The following command archives the current project:

```
archive_project
```

Note The project archive is named *project_name.zip* because no file name is specified.

The following example specifies *project_3* as the current project, and then archives that project into a file called *proj3.zip*:

```
current_project project_3
archive_project -force -exclude_run_results proj3.zip
```

Note The use of the **-force** argument causes the tool to overwrite the *proj3.zip* file if one exists. The use of the **-exclude_run_results** argument causes the tool to leave any results from synthesis or implementation runs out of the archive. The various runs defined in the project are included in the archive, but not any of the results.

See Also

- [current_project](#)
- [write_project_tcl](#)

assign_bd_address

Automatically assign addresses to unmapped IP.

Syntax

```
assign_bd_address [-target_address_space arg] [-quiet] [-verbose]  
[objects...]
```

Returns

The newly mapped segments, "" if failed

Usage

Name	Description
<code>[-target_address_space]</code>	Target address space to place segment into
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[<i>objects</i>]</code>	The objects to assign

Categories

[IPIntegrator](#)

check_timing

Check the design for possible timing problems.

Syntax

```
check_timing [-file arg] [-name arg] [-override_defaults args]  
[-include args] [-exclude args] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-name]	Output the results to GUI panel with this name
[-override_defaults]	Overrides the checks in the default timing checks listed below
[-include]	Add this list of checks to be performed along with default timing checks listed below
[-exclude]	Exclude this list of checks to be performed from the default timing checks listed below
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#), [Timing](#)

Description

Checks the design elements of ports, pins, and paths, against the current timing constraints. Use this command to identify possible problems with design data and timing constraints before running the **report_timing** command. The **check_timing** command runs a series of default timing checks, and reports a summary of any violations found. To get detailed information about violations, use the **-verbose** option.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to the GUI, or to a file.

Default Timing Checks:

- **no_clock** - Reports unlocked registers. In this case, no setup or hold checks are performed on data pins related to the register clock pin.
- **unconstrained_internal_endpoints** - This warning identifies timing path endpoints at register data pins that are not constrained. Endpoints at register data pins are constrained by clock assignment using the **create_clock** command. Endpoints at output ports are checked and reported by the **no_output_delay** check.
- **no_input_delay** - Reports the input ports without an input delay constraint. Input delays can be assigned using the **set_input_delay** command. Input ports that are unlocked will not be checked for input delays.
- **no_output_delay** - Reports the output ports without an output delay constraint. Output delays can be assigned using the **set_output_delay** command. Output ports that are unlocked will not be checked for output delays.
- **multiple_clock** - Warns if multiple clocks reach a register clock pin. If more than one clock signal reaches a register clock pin it is unclear which clock will be used for analysis. In this case, use the **set_case_analysis** command so that only one clock will propagate to the register clock pin.
- **generated_clocks** - Checks for loops, or circular definitions within the generated clock network. This check will return an error if a generated clock uses a second generated clock as its source, when the second generated clock uses the first clock as its source.
- **loops** - Checks for and warns of combinational feedback loops in the design.
- **partial_input_delay** - Reports the input ports having partially defined input delay constraints. Assigning **set_input_delay -max** or **set_input_delay -min** to an input port, without assigning the other, creates a partially defined input delay. In such cases, paths starting from the input port may become unconstrained and no timing checks will be done against the port. Assigning **set_input_delay** without specifying either **-min** or **-max** allows the tool to assume both min and max delays, and so does not result in a partial input delay.
Note Unlocked input ports are not checked for partial input delays.
- **partial_output_delay** - Reports the output ports having partially defined output delay constraints. Assigning **set_output_delay -max** or **set_output_delay -min** to an output port, without assigning the other, creates a partially defined output delay. In such cases, paths reaching the port may become unconstrained and no timing checks will be done against the port. Assigning **set_output_delay** without specifying either **-min** or **-max** allows the tool to assume both min and max delays, and so does not result in a partial output delay.
Note Unlocked output ports are not checked for partial output delays.
- **unexpandable_clocks** - Reports clock sets in which the period is not expandable with respect to each other, when there is at least 1 path between the clock sets. A clock is unexpandable if no common multiples are found within 1000 cycles between the source and destination clocks.
- **latch_loops** - Checks for and warns of combinational latch loops in the design.

Arguments

-file *arg* - (Optional) Write the results to the specified file on disk. By default, the output of this command is written to the Tcl console.

Note If the path is not specified as part of the file name, the tool will write the named file into the current working directory, or the directory from which the tool was launched.

-name *arg* - (Optional) Creates the named report in the Timing Results view of the GUI.

-override_defaults *{args}* - (Optional) Override the default timing checks and run the specified checks. Specify the checks to be performed from the list of checks described above.

-include *args* - (Optional) Run the specified checks in addition to the current default checks. Specify the checks to be performed from the list of checks described above.

-exclude *args* - (Optional) Exclude the specified checks from the default checks performed by the **check_timing** command. Specify the checks to be excluded from the list of checks described above.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example runs **check_timing**, but excludes the specified checks from the default timing checks:

```
check_timing -exclude {loops generated_clocks}
```

The following example uses the **-verbose** argument to obtain detailed results running just the **multiple_clocks** check, and then uses **get_clocks** to look further into the issue:

```
check_timing -verbose -override_defaults {multiple_clock}
Checking multiple_clock.
There are 2 register/latch pins with multiple clocks.
procEngine/mode_du/set_reg[0]/C
provEngine/mode_du/set_reg[1]/C
get_clocks -of_objects [get_pin procEngine/mode_du/set_reg[0]/C]
sysClk coreClk
```

See Also

- [create_clock](#)
- [get_clocks](#)
- [report_timing](#)
- [set_case_analysis](#)
- [set_input_delay](#)
- [set_max_delay](#)
- [set_output_delay](#)

checkpoint_vcd

Create a VCD checkpoint (equivalent of Verilog \$dumpall system task).

Syntax

```
checkpoint_vcd [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Simulation](#)

Description

The **checkpoint_vcd** command inserts current HDL object signal values into the Value Change Dump (VCD) file. Nothing is returned. This Tcl command is the equivalent of the Verilog **\$dumpall** system task, providing the initial values of the specified signals.

VCD is an ASCII file containing header information, variable definitions, and value change details of a set of HDL signals. The VCD file can be used to view simulation result in a VCD viewer or to estimate the power consumption of the design. See the *IEEE Standard for Verilog Hardware Description Language* (IEEE Std 1364-2005) for a description of the VCD file format.

You must execute the **open_vcd** and **log_vcd** commands before using the **checkpoint_vcd** command. After you execute the **checkpoint_vcd** command, run or rerun the simulation to capture the signal values.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following is an example of the **checkpoint_vcd** command where the command dumps signal values of specified HDL objects into the open VCD file:

```
checkpoint_vcd
```

See Also

- [flush_vcd](#)
- [log_vcd](#)
- [open_vcd](#)

close_bd_design

Close a design.

Syntax

```
close_bd_design [-quiet] [-verbose] name
```

Returns

The design object, "" if failed

Usage

Name	Description
-quiet	Ignore command errors
-verbose	Suspend message limits during command execution
<i>name</i>	Name of design to close

Categories

[IPIntegrator](#)

Description

Closes the specified IP subsystem design in the IP Integrator feature of the Vivado Design Suite.

If the design has been modified, you will not be prompted to save the design prior to closing. You will need to run **save_bd_design** to save any changes made to the design before using the **close_bd_design** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - The name of the IP subsystem design object to close.

Example

The following example closes the current IP subsystem design in the current project:

```
close_bd_design [current_bd_design]
```

See Also

- [create_bd_design](#)
- [current_bd_design](#)
- [get_bd_designs](#)
- [open_bd_design](#)
- [save_bd_design](#)

close_design

Close the current design.

Syntax

```
close_design [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Project](#)

Description

Closes the currently active design. If the design has been modified, you will not be prompted to save the design prior to closing. You will need to run **save_design** or **save_design_as** to save any changes made to the design before using the **close_design** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example closes the current design:

```
close_design
```

Note If multiple designs are open, you can specify the current design with the **current_design** command prior to using **close_design**.

The following example sets the current design, then closes it:

```
current_design rtl_1  
close_design
```

current_design sets **rtl_1** as the active design, then the **close_design** command closes it.

See Also

[current_design](#)

close_hw

Close the hardware tool.

Syntax

```
close_hw [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Hardware](#)

close_hw_target

Close a hardware target.

Syntax

```
close_hw_target [-quiet] [-verbose] [hw_target]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_target]	hardware target Default: current hardware target

Categories

[Hardware](#)

close_project

Close current opened project.

Syntax

```
close_project [-delete] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-delete]	Delete the project from disk also
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Project](#)

Description

Closes the current open project.

Arguments

-delete - (Optional) Delete the project data from the hard disk after closing the project.

Note Use this argument with caution. You will not be prompted to confirm the deletion of project data.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following command closes the active project:

```
close_project
```

This example closes the current project. If you have multiple projects open, the **close_project** command applies to the current project which can be defined with the **current_project** command.

The following example sets `project_1` as the current project, and then closes the project and deletes it from the computer hard disk:

```
current_project project_1  
close_project -delete
```

Note Use the **-delete** argument with caution. You will not be prompted to confirm the deletion of project data.

See Also

[current_project](#)

close_saif

Flush SAIF toggle information to the SAIF output file and close the file.

Syntax

```
close_saif [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Simulation](#)

Description

Closes the open SAIF file.

Only one SAIF file can be open in the Vivado simulator at one time, using **open_saif**. You must close the currently opened SAIF file before opening another file.

This command returns nothing if it is successful, or an error if it fails.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following is an example:

```
close_saif
```

See Also

- [log_saif](#)
- [open_saif](#)

close_sim

Unload the current simulation without exiting Vivado.

Syntax

```
close_sim [-force] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-force]	Forces the closing of the simulation, even if changes would be lost. Default behavior is to reject the closing with an error if changes would be lost.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Simulation](#)

Description

Close the current Vivado simulation.

Note This command does not support third party simulators.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example closes the current simulation:

```
close_sim
```


See Also

- [current_sim](#)
- [launch_xsim](#)

close_vcd

Flush VCD information to the VCD output file and close the file.

Syntax

```
close_vcd [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Simulation](#)

Description

Closes the open Value Change Dump (VCD) file.

Only one VCD file can be open in the Vivado simulator at one time. You must close the currently opened VCD file before opening another file.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example closes the current VCD object:

```
close_vcd
```

See Also

[open_vcd](#)

close_wave_config

Closes the wave config.

Syntax

```
close_wave_config [-force] [-quiet] [-verbose] [wcfgobj]
```

Returns

Nothing

Usage

Name	Description
[-force]	Forces the closing of the wave configuration, even if changes would be lost. Default behavior is to reject the closing with an error if changes would be lost.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[wcfgobj]</i>	Closes and destroys the specified wave configuration object, or the current wave configuration if none specified Default: NULL

Categories

[Waveform](#)

commit_hw_sio

Commit the property changes of the current hardware object. Inputs can be any hardware object. At least one object is required.

Syntax

```
commit_hw_sio [-quiet] [-verbose] hw_objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_objects</i>	hardware objects

Categories

[Hardware](#)

commit_hw_vio

Write hardware VIO probe OUTPUT_VALUE properties values to VIO core(s).

Syntax

```
commit_hw_vio [-quiet] [-verbose] hw_objects...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_objects</i>	List of hardware VIO and hardware probe objects.

Categories

[Hardware](#)

compile_simlib

Compile simulation libraries.

Syntax

```
compile_simlib [-cfg_file] [-directory arg] [-exclude_sublib]
[-exclude_superseded] [-family arg] [-force] [-language arg]
[-library arg] [-precompiled_directory arg] [-simulator arg]
[-simulator_exec_path arg] [-source_library_path arg] [-32bit] [-quiet]
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[-cfg_file]	Create new configuration file with default settings Default: compile_simlib.cfg
[-directory]	Directory path for saving the compiled results Default: .
[-exclude_sublib]	Exclude the sub-lib(s) defined in the edk .pao file for compilation (For edk library only)
[-exclude_superseded]	Exclude the superseded edk lib(s) for compilation (For edk library only)
[-family]	Select device architecture Default: all
[-force]	Overwrite the pre-compiled libraries
[-language]	Compile libraries for this language Default: all
[-library]	Select library to compile Default: all
[-precompiled_directory]	Specify the existing directory where previously compile_simlib-compiled libraries are located
[-simulator]	Compile libraries for this simulator
[-simulator_exec_path]	Use simulator executables from this directory
[-source_library_path]	If specified, this directory will be searched for the library source files before searching the default path(s) found in environment variable XILINX_PLANAHEAD (for Vivado) or XILINX_EDK (for EDK)
[-32bit]	Perform the 32-bit compilation
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Simulation

Description

Compile Xilinx simulation libraries.

Xilinx HDL-based simulation libraries come pre-compiled for use with the Vivado simulator and Isim. The **compile_simlib** command compiles the simulation libraries for use by other simulators. Libraries must generally be compiled or recompiled with a new software release to update simulation models and to support a new version of a simulator.

When this command is run with a project open, the tool will use the device family, target language, and library settings specified by the project as the default values, rather than the default settings of the command defined below. The default settings can be overridden by specifying the necessary options when the command is run.

The command returns information related to the compiled libraries.

Arguments

-cfg_file - (Optional) Create a configuration file called `comp_xlib.cfg` with the default settings if it is not present in the current directory.

-directory arg - (Optional) Directory path for saving the compiled results.

Note By default the libraries are saved to the current working directory, or the directory the tool was launched from.

-exclude_sublib - (Optional) Exclude the sub-library defined in the EDK `.pao` file for compilation. This option is only for use with for EDK libraries. See the *Embedded System Tools Reference Guide* (UG111) for information on sub-libraries.

-exclude_superseded - (Optional) Exclude the superseded EDK library from compilation. This option is only for use with for EDK libraries. See the *Embedded System Tools Reference Manual* (UG111) for more information on superseded libraries.

-family arg - (Optional) Compile selected libraries to the specified device family. All device families will be generated by default. The following are the device families that can be specified:

- `virtex7` (for Virtex-7)
- `kintex7` (for Kintex-7)
- `kintex7l` (for Kintex-7 Lower Power)
- `artix7` (for Artix-7)
- `artix7l` (for Artix-7 Lower Power)
- `zynq` (for Zynq-7000 EPP)

-force - (Optional) Overwrite the current pre-compiled libraries.

-language [verilog | vhdl | all] - (Optional) Compile libraries for the specified language. If this option is not specified then the language will be set according to the simulator selected with **-simulator**. For multi-language simulators both Verilog and VHDL libraries will be compiled.

-library *arg* - (Optional) Specify the library to compile. Valid values for library are:

- all
- unisim
- simprim
- xilinxcorelib
- edk

To specify multiple libraries, repeat the -lib options for each library. For example:

```
.. -library unisim -library simprim ..
```

Note If you select EDK libraries (-lib edk), all ISE libraries will be compiled because EDK libraries are dependent on UNISIM and SIMPRIM.

-precompiled_directory *arg* - (Optional) Specify the directory where pre-compiled libraries are currently located.

-simulator *arg* - (Optional) Compile libraries for the specified simulator. Valid simulator values are:

- modelsim
- questasim
- ies (Linux only)
- vcs_mx (Linux only)
- riviera
- Active-HDL

-simulator_exec_path *arg* - (Optional) Specify the directory to locate the simulator executable. This option is required if the target simulator is not specified in the \$PATH or %PATH% environment variable; or to override the path from the \$PATH or %PATH% environment variable.

-source_library_path *arg* - (Optional) If specified, this directory will be searched for the library source files before searching the default path(s) defined by the environment variables (\$XILINX, \$XILINX_PLANAHEAD, or \$XILINX_EDK).

Note Do not use this option unless explicitly instructed to by Xilinx Technical Support.

-32bit - (Optional) Perform 32-bit compilation instead of the default 64-bit compilation.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example shows how to compile UNISIM and SIMPRIM libraries for ModelSim (VHDL) for a design using a Virtex-7 device:

```
compile_simlib -simulator modelsim -family virtex7 -library unisim \  
-library simprim -language vhdl
```

See Also

[launch_modelsim](#)

config_timing_analysis

Configure timing analysis general settings.

Syntax

```
config_timing_analysis [-enable_input_delay_default_clock arg]
[-enable_preset_clear_arcs arg] [-disable_flight_delays arg] [-quiet]
[-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-enable_input_delay_default_clock]</code>	Launch SDC unlocked input delays from an internally defined clock: Values: true, false; This option is not supported for UCF constraints
<code>[-enable_preset_clear_arcs]</code>	Time paths through asynchronous preset or clear timing arcs: true, false;
<code>[-disable_flight_delays]</code>	Disable adding package times to IO Calculations : Values: true, false;
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Timing](#)

Description

This command configures general features of timing analysis.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-enable_input_delay_default_clock [true | false] - (Optional) Launch unlocked input delays from an internally defined clock for timing analysis. The valid values are true or false, with a default setting of false.

-enable_preset_clear_arcs [true | false] - (Optional) Time paths through asynchronous preset or clear timing arcs. The valid values are true or false, with a default setting of false.

-disable_flight_delays [true | false] - (Optional) Do not add package delays to I/O calculations when this option is true.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example ignores the package delays during timing analysis:

```
config_timing_analysis -disable_flight_delays true
```

See Also

- [config_timing_corners](#)
- [report_timing](#)

config_timing_corners

Configure single / multi corner timing analysis settings.

Syntax

```
config_timing_corners [-corner arg] [-delay_type arg] [-setup] [-hold]  
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-corner]	Name of the timing corner to be modified : Values: Slow, Fast
[-delay_type]	Type of path delays to be analyzed for specified timing corner: Values: none, max, min, min_max
[-setup]	Enable timing corner for setup analysis (equivalent to -delay_type max)
[-hold]	Enable timing corner for hold analysis (equivalent to -delay_type min)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Timing

Description

This command configures the Slow and Fast timing corners for single or multi-corner timing analysis.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-corner [Slow | Fast] - (Optional) Specifies the name of the timing corner to be configured. Valid values are "Slow" and "Fast".

Note The names of the corners are case sensitive.

-delay_type *value* - (Optional) Specify the type of path delays to be analyzed for the specified timing corner. Valid values are "max", "min" and "min_max".

-setup - (Optional) Specifies setup analysis for the specified timing corner. This is the same as **-delay_type max**.

-hold - (Optional) Specifies hold analysis for the timing corner. This is the same as **-delay_type min**.

Note You can specify both **-setup** and **-hold** which is the same as **-delay_type min_max**.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example configures the Slow timing corner for both setup and hold analysis:

```
config_timing_corners -corner Slow -setup -hold
config_timing_corners -corner Slow -delay_type min_max
```

Note The two preceding examples have the same effect.

The following example configures the Fast corner for min delay analysis:

```
config_timing_corners -corner Fast -delay_type min
```

See Also

- [config_timing_analysis](#)
- [report_timing](#)

config_webtalk

Enable/disable WebTalk to send software, IP and device usage statistics to Xilinx.

Syntax

```
config_webtalk [-info] [-user arg] [-install arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-info]</code>	Show whether WebTalk is currently enabled or disabled
<code>[-user]</code>	Enable/disable WebTalk for the current user. Specify either 'on' to enable or 'off' to disable. Default: empty
<code>[-install]</code>	Enable/disable WebTalk for all users of the current installation. Specify either 'on' to enable or 'off' to disable. If you specify 'off', individual users will not be able to enable WebTalk using the -user option. You may need administrator rights to use this option. Default: empty
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[FileIO](#)

Description

WebTalk is a secure design data collection feature of Xilinx software that helps Xilinx understand how you are using Xilinx FPGA devices, software, and Intellectual Property (IP).

This command returns the current state of the WebTalk feature for the current user and software installation. You can also enable or disable WebTalk to send software, IP and device usage statistics to Xilinx. No data is sent if you disable WebTalk, except for the use of the WebPACK license to generate a bitstream.

Participation in WebTalk is voluntary, except for the use of the WebPACK license. WebTalk data transmission is mandatory, and is always enabled for WebPACK users. WebTalk ignores user and install preference when a bitstream is generated using the WebPACK license.

Note If a design is using a device contained in WebPACK and a WebPACK license is available, the WebPACK license will be used. To change this, please see answer record 34746.

Arguments

-info - (Optional) Returns information about the current Webtalk configuration. The state of WebTalk is dependant on both the user and install setting. If either of these settings is disabled, then WebTalk is disabled.

-user arg - (Optional) Enables or disables WebTalk for the current user.

-install - (Optional) Enables or disables WebTalk for the current software installation.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the current state of the WebTalk configuration:

```
config_webtalk -info
INFO: [Coretc1-120] Webtalk has been disabled by the current user.
INFO: [Coretc1-123] Webtalk has been enabled for the current installation.
INFO: [Coretc1-110] This combination of user/install settings means that WebTalk is currently disabled.
```

The following example enables WebTalk for the current user:

```
config_webtalk -user on
```

connect_bd_intf_net

Connect intf_port and intf_pin list.

Syntax

```
connect_bd_intf_net [-intf_net arg] [-quiet]  
[-verbose] object1 object2
```

Returns

0 if successful, error otherwise

Usage

Name	Description
[-intf_net]	The single intf_net that all objects connect to
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>object1</i>	Name of intf_port or intf_pin to connect
<i>object2</i>	Name of intf_port or intf_pin to connect

Categories

[IPIntegrator](#)

connect_bd_net

Connect port and pin object list.

Syntax

```
connect_bd_net [-net arg] [-quiet] [-verbose] objects...
```

Returns

0 if successful, error otherwise

Usage

Name	Description
[-net]	The single net that all objects connect to
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	The objects connect to the net

Categories

IPIntegrator

Description

Creates a block diagram (bd) net in the current design connecting the specified list of block diagram port and pin objects.

Use the **get_bd_ports** and **get_bd_pins** commands to specify the port and pin objects.

The command returns the connected block diagram net object.

Arguments

-net *arg* - (Optional) Create a single block diagram net that all the specified block diagram pins and ports connect to.

Note The **-net** argument is optional. When the objects being connected are not in the same level of hierarchy, the net argument should not be specified.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - Connect the specified list of block diagram port and pin objects.

Examples

The following example connects a bd_port to a bd_pin on different level of hierarchy:

```
connect_bd_net [get_bd_ports /clk] [get_bd_pins /mycell/mysubcell/clk]
```

Note Because /clk and /mycell/mysubcell/clk are in different levels of the hierarchy, the **-net** option is not specified. In this case, multiple nets are created for connection across the hierarchy.

See Also

- [get_bd_pins](#)
- [get_bd_ports](#)

connect_debug_port

Connect nets and pins to debug port channels.

Syntax

```
connect_debug_port [-channel_start_index arg] [-quiet]  
[-verbose] port nets...
```

Returns

Nothing

Usage

Name	Description
<code>[-channel_start_index]</code>	Connect nets starting at channel index
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>port</code>	Debug port name
<code>nets</code>	List of nets or pins

Categories

Description

Connects a signal from the Netlist design to a port on a ChipScope debug core. The signal can either be connected to a specific channel index on the port, or simply connected to an available channel on the port.

If you try to connect too many signals to a port, or there are not enough channels to support the connection, the tool will return an error.

Additional ports can be added to a debug core through the use of the `create_debug_port` command, and you can increase the available channels on an existing port with the **set_property port_width** command. See the examples below.

You can disconnect signals from ports using the **disconnect_debug_port** command.

When the ChipScope debug core has been defined and connected, you can implement the debug core as a block for inclusion in the Netlist Design. Use the **implement_debug_core** command to use CoreGen to implement the core.

Arguments

-channel_start_index *arg* - (Optional) The channel index to use for the connection. If more than one signal has been specified, this is the channel index where connections will start to be added. Channel indexes are numbered starting at 0.

Note If this argument is not specified, the tool will place connections on the first available channel index.

port - (Required) The name of the port to connect signals to. The port must be referenced by the *core_name/port_name*.

nets - (Required) A list of one or more net names from the Netlist Design to connect to the specified debug port.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns **TCL_OK** regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example creates a new TRIG port on the myCore debug core, increases the *port_width* of the port in order to prepare it to receive the number of signals to be connected, then connects the signals to the port starting at the third channel position (index 2).

```
create_debug_port myCore TRIG
set_property port_width 8 [get_debug_ports myCore/TRIG0]
connect_debug_port myCore/TRIG0 [get_nets [list m0_ack_o m0_cyc_i m0_err_o \
    m0_rty_o m0_stb_i m0_we_i ]] -channel_start_index 2
```

Note If you specify too many nets to connect to the available channels on the port, the tool will return an error and will not connect the ports.

See Also

- [create_debug_port](#)
- [disconnect_debug_port](#)
- [get_debug_ports](#)
- [get_nets](#)
- [implement_debug_core](#)
- [set_property](#)

connect_hw_server

Open a connection to a hardware server.

Syntax

```
connect_hw_server [-host arg] [-port arg] [-password arg]  
[-launch arg] [-quiet] [-verbose]
```

Returns

Hardware server

Usage

Name	Description
[-host]	server host name Default: localhost
[-port]	server port number Default: 60001
[-password]	server password Default: none
[-launch]	launch server process Default: No
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Hardware](#)

connect_net

Connect a net to pins or ports.

Syntax

```
connect_net [-hier] [-basename arg] -net arg -objects args [-quiet]
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[-hier]	Allow hierarchical connection, creating nets and pins as needed (see -basename).
[-basename]	base name to use for net / pin names needed when doing hierarchical connection (see -hier). Default value is inferred from the name of the net being connected (see -net).
-net	Net to connect
-objects	List of pins or ports to connect
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Netlist](#)

Description

This command allows the user to connect a specified net to one or more pins or ports in the netlist of an open Synthesized or Implemented Design.

The **connect_net** command will also connect nets across levels of hierarchy in the design, by adding pins and hierarchical nets as needed to complete the connection. Added nets and pins can be assigned a custom basename to make them easy to identify, or will be assigned a basename by the Vivado tool.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

Arguments

-hier - (Optional) Connect the net across different levels of the hierarchy.

Note If **-hier** is not specified, attempting to connect to hierarchical pins will fail with a warning

-basename *arg* - (Optional) Specifies a custom name to use for any hierarchical nets or pins that are needed to connect the specified net across levels of the hierarchy. If this option is not used, the basename is automatically derived from the net being connected.

-net *arg* - (Required) Specifies the net to connect.

Note Although you can create a bus using the **-from** and **-to** arguments of the **create_net** command, you must connect each bit of the bus separately using the **connect_net** command

-objects *args* - (Required) Specified the list of pins or ports to connect the net to. You can connect a net to one or more pin or port objects.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Example

The following example creates a port; creates a pin on the myDMA instance; creates a net called myEnable; and connects the net to the newly created port and pin:

```
create_port -direction IN enableIn
create_pin -direction IN myDMA/en
create_net myEnable
connect_net -net myEnable -objects {enableIn myDMA/en}
```

The following example creates 32-bit bus ports, pins, and nets, then connects them together one bit at a time using a for loop to connect each bit individually:

```
create_port -from 0 -to 31 -direction IN dataIN
create_pin -from 0 -to 31 -direction IN myDMA/data
create_net -from 0 -to 31 dataBus
for {set x 0} {$x<32} {incr x} { \
    connect_net -net dataBus[$x] -objects {dataIN[$x] myDMA/data[$x] } }
```

Note Attempting to connect the dataBus will result in a "Net not found error." Each bit of the bus must be separately connected.

See Also

- [create_net](#)
- [create_pin](#)
- [create_port](#)
- [disconnect_net](#)
- [remove_net](#)
- [resize_net_bus](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

copy_bd_objs

Make copies of the objects and add the copies to the given hierarchical cell.

Syntax

```
copy_bd_objs [-prefix arg] [-from_design arg] [-quiet]  
[-verbose] parent_cell objects...
```

Returns

0, "" if failed

Usage

Name	Description
[-prefix]	Prefix name to add to cells
[-from_design]	The design to own the original objects
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>parent_cell</i>	Parent cell
<i>objects</i>	The objects to copy

Categories

[IPIntegrator](#)

copy_ip

Copy an existing IP.

Syntax

```
copy_ip -name arg [-dir arg] [-quiet] [-verbose] objects...
```

Returns

IP file object that was added to the project

Usage

Name	Description
-name	Name of copied IP
[-dir]	Directory path for remote IP to be created and managed outside the project
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	IP to be copied

Categories

[Project](#), [IPFlow](#)

create_bd_addr_seg

Create a new segment.

Syntax

```
create_bd_addr_seg -range arg -offset arg [-quiet] [-verbose]  
[parent_addr_space] [slave_segment] name
```

Returns

The newly created segment object, "" if failed

Usage

Name	Description
-range	Range of segment. e.g. 4096, 4K, 16M, 1G
-offset	Offset of segment. e.g. 0x00000000
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[parent_addr_space]	Parent address space of segment
[slave_segment]	Slave segment of the created segment
name	Name of segment to create

Categories

[IPIntegrator](#)

create_bd_cell

Add an IP cell from the IP catalog, or add a new hierarchical block.

Syntax

```
create_bd_cell [-vlnv arg] [-type arg] [-quiet] [-verbose] name
```

Returns

The newly created cell object. Returns nothing if the command fails

Usage

Name	Description
[-vlnv]	Vendor:Library:Name:Version of the IP cell to add from the IP catalog.
[-type]	Type of cell to create. Valid values are IP and hier. Default: IP
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of cell to create

Categories

[IPIntegrator](#)

create_bd_design

Create a new design and its top level hierarchy cell with the same name.

Syntax

```
create_bd_design [-quiet] [-verbose] name
```

Returns

The newly created design object, "" if failed

Usage

Name	Description
-quiet	Ignore command errors
-verbose	Suspend message limits during command execution
<i>name</i>	Name of design to create

Categories

[IPIntegrator](#)

Description

Create a new IP subsystem design module to add to the current project, and for use with the IP Integrator feature of the Vivado Design Suite.

An empty IP subsystem module is created and added to the source files of the current project. The subsystem module and file are created with the specified *name* in the current project at:

project_name/project_name.srcs/sources_1/bd/name/name.bd

This command returns the file path and name of the IP subsystem design created if the command is successful. An error is returned if the command fails.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - The name of the IP subsystem design module to create.

Example

The following example creates a new empty IP subsystem module called `design_1`, adds the module to the current project, and creates a file called `design_1.bd` in the sources directory of the project:

```
create_bd_design design_1
```

See Also

- [close_bd_design](#)
- [current_bd_design](#)
- [open_bd_design](#)
- [save_bd_design](#)

create_bd_intf_net

Create a new intf_net.

Syntax

```
create_bd_intf_net [-quiet] [-verbose] name
```

Returns

The newly created intf_net object, "" if failed

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of intf_net to create

Categories

[IPIntegrator](#)

create_bd_intf_pin

Create a new intf_pin.

Syntax

```
create_bd_intf_pin -vlnv arg -mode arg [-quiet] [-verbose] name
```

Returns

The newly created intf_pin object, "" if failed

Usage

Name	Description
-vlnv	Bus vlnv
-mode	Bus interface mode
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of intf_pin to create

Categories

[IPIntegrator](#)

create_bd_intf_port

Create a new interface port.

Syntax

```
create_bd_intf_port -vlnv arg -mode arg [-quiet] [-verbose] name
```

Returns

The newly created interface port object, "" if failed

Usage

Name	Description
-vlnv	Bus vlnv
-mode	Bus interface mode
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of port to create

Categories

[IPIntegrator](#)

create_bd_net

Create a new net.

Syntax

```
create_bd_net [-quiet] [-verbose] name
```

Returns

The newly created net object, "" if failed

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of net to create

Categories

[IPIntegrator](#)

create_bd_pin

Create a new pin.

Syntax

```
create_bd_pin [-from arg] [-to arg] -dir arg [-type arg] [-quiet]  
[-verbose] name
```

Returns

The newly created pin object, "" if failed

Usage

Name	Description
[-from]	Begin index Default: Unspecified
[-to]	End index Default: Unspecified
-dir	Pin direction
[-type]	Pin type
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of pin to create

Categories

[IPIntegrator](#)

create_bd_port

Create a new port for an IP subsystem design.

Syntax

```
create_bd_port [-from arg] [-to arg] -dir arg [-type arg] [-quiet]  
[-verbose] name
```

Returns

The newly created port object. Returns nothing if the command fails

Usage

Name	Description
[-from]	Beginning index Default: Unspecified
[-to]	Ending index Default: Unspecified
-dir	Port direction. Valid values are I, O, or IO.
[-type]	Port type. Valid values are clk, ce, rst, intr, data.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of port to create

Categories

[IPIntegrator](#)

create_cell

Create cells in the current design.

Syntax

```
create_cell -reference arg [-black_box] [-quiet] [-verbose] cells...
```

Returns

Nothing

Usage

Name	Description
-reference	Library cell or design which cells reference
[-black_box]	Create black box instance
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>cells</i>	Names of cells to create

Categories

[Netlist](#)

Description

Add cells to the netlist of the current Synthesized or Implemented design.

Note You cannot add cells to library macros, or macro-primitives.

New cell instances can be added to the top-level of the design, or hierarchically within any module of the design. Instances can reference an existing cell from the library or design source files, or a black box instance can be added that reference cells that have not yet been created.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

This command returns the name of the created cell instance or instances.

Arguments

-reference *arg* - (Required) The library cell or source file module referenced by the new cell instances.

-black_box - (Optional) Define a black box instance of the specified reference cell. Use this argument when the reference cell does not exist yet, but you would like to create a black box instance of the cell for a top-down design approach.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

cells - (Required) Instance names of the cells to create. The instance name can be specified as a hierarchical name, from the top-level of the design. In this case, you must use the hierarchy separator character in the hierarchical instance name. You can determine the current hierarchy separator with the **get_hierarchy_separator** command.

Examples

The following example creates three new cell instances of the or1200_cpu module with the specified instance names:

```
create_cell -reference or1200_cpu myCell11 myCell12 myCell13
```

The following example sets the hierarchy separator character, then creates a black box instance for the referenced cell, specifying a hierarchical instance name:

```
set_hierarchy_separator |  
create_cell -reference dmaBlock -black_box usbEngine0|myDMA
```

Note The tool will return an error when **-black_box** is used, but the **-reference** cell already exists

See Also

- [remove_cell](#)
- [set_hierarchy_separator](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

create_clock

Create a clock object.

Syntax

```
create_clock -period arg [-name arg] [-waveform args] [-add] [-quiet]  
[-verbose] [objects]
```

Returns

New clock object

Usage

Name	Description
-period	Clock period: Value > 0
[-name]	Clock name
[-waveform]	Clock edge specification
[-add]	Add to the existing clock in source_objects
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[objects]	List of clock source ports, pins or nets

Categories

[SDC](#), [XDC](#)

Description

Create a clock object with the specified period or waveform. This command defines primary clocks which are used by the timing engine as the delay propagation starting point of any clock edge. The defined clock can be added to the definition of an existing clock, or overwrite the existing clock.

A virtual clock can be created that has no source in the design. A virtual clock can be used as a time reference for setting input and output delays but does not physically exist in the design.

A clock can also be generated from an existing physical clock, and derive many of its properties from the master clock. Use the **create_generated_clock** command to derive a clock from an existing physical clock.

Note This command returns the name of the clock object that is created.

Arguments

-period *arg* - (Required) Specifies the clock period of the clock object to be created. The value must be greater than zero (>0), and has a default value of 10.0 time units.

Note The time units are defined by the **set_units** command. The default time units is nanoseconds (ns), with a resolution of one picosecond (ps)

-name *arg* - (Optional) The name of the clock object to be created. If the name is omitted, a system-generated name will be used based on the specified source *objects*. You can also use the **-name** option without source *objects* to create a virtual clock for the design that is not associated with a physical source on the design.

-waveform *arg1 arg2 ...* - (Optional) The rise and fall edge times of the waveform of the defined clock, in nanoseconds, over one full clock cycle. There must be an even number of edges, representing both the rising and falling edges of the waveform. The first time specified is the first rising transition, and the second time specified is the falling edge. If the value for the falling edge is smaller than the value for the rising edge, it means that the falling edge happens before the rising edge.

Note If you do not specify the waveform, the default waveform is assumed to have a rising edge at time 0.0 and a falling edge at one half the specified period (**-period/2**)

-add - (Optional) Define multiple clocks on the same source for simultaneous analysis with different clock waveforms. Use **-name** to specify the new clock to add. If you do not specify this option, the **create_clock** command will automatically assign a name and will overwrite any existing clock of the same name.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) The ports, pins, or nets which are the source of the specified clock. If you specify a clock on a source object that already has a clock, the new clock will overwrite the original clock unless you also specify the **-add** option. If no *objects* are specified to attach the clock object to, the clock will be created as a virtual clock in the design.

Note The first driver pin of a specified net will be used as the source of the clock

Examples

The following example creates a physical clock called bftClk and defines the clock period:

```
create_clock -name bftClk -period 5.000 [get_ports bftClk]
```

Note If the **get_ports** command defining the objects is left off of this example, a virtual clock is created in the design rather than a physical clock.

The following example creates a clock named clk on the input port, clk_pin_p, with a period of 10ns, the rising edge at 2.4ns and the falling edge at 7.4ns:

```
create_clock -name clk -period 10.000 -waveform {2.4 7.4} [get_ports clk_pin_p]
```

The following example creates a virtual clock since no clock source is specified:

```
create_clock -name virtual_clock -period 5.000
```

The following example creates a clock with the falling edge at 2ns and the rising edge at 7ns:

```
create_clock -name clk -period 10.000 -waveform {7 2} [get_ports clk_pin_p]
```

See Also

- [all_clocks](#)
- [create_generated_clock](#)
- [get_clocks](#)
- [report_clocks](#)
- [report_clock_interaction](#)
- [report_clock_networks](#)
- [report_clock_utilization](#)
- [set_clock_groups](#)
- [set_clock_latency](#)
- [set_clock_uncertainty](#)
- [set_input_delay](#)
- [set_output_delay](#)
- [set_propagated_clock](#)
- [set_units](#)

create_debug_core

Create a new Integrated Logic Analyzer debug core.

Syntax

```
create_debug_core [-quiet] [-verbose] name type
```

Returns

New debug_core object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of the new debug core instance
<i>type</i>	Type of the new debug core

Categories

Description

Adds a new Integrated Logic Analyzer (ILA) debug core, labtools_ila_v2, to an open Netlist Design in the current project. The ILA debug core defines ports for connecting nets to for debug purposes.

Note A debug core can only be added to an open Netlist Design in the tool.

The default core that is created includes a CLK port and a trigger (TRIG) port. The CLK port only supports one clock signal, and so you must create a separate debug core for each clock domain.

Once the core is created you can add new ports to the debug core with the **create_debug_port** command, and connect signals to the ports using the **connect_debug_port** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the ILA debug core to add to the project.

type - (Required) The type of debug core to insert. Only the labtools_ila_v2 debug core is currently supported in the Vivado tool. The ILA debug core simply adds another load onto a connected net without otherwise altering it. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information on debug core types and purpose.

Note When the ILA core is added to the project, the tool also adds a Debug Hub core (labtools_xsdbmasterlib_v2) as a container for one or more ILA cores. However, you cannot directly add a Debug Hub to the project.

Examples

The following example opens the Netlist Design, and creates a new ILA debug core:

```
open_netlist_design -name netlist_1
create_debug_core myCore labtools_ila_v2
```

The following example creates a new debug core called myCore and returns the properties of the newly created core:

```
report_property [create_debug_core myCore labtools_ila_v2]
```

The properties of the debug core can be customized by using the **set_property** command as in the following example:

```
set_property enable_storage_qualification false [get_debug_cores myCore]
```

See Also

- [connect_debug_port](#)
- [create_debug_port](#)
- [delete_debug_core](#)
- [get_debug_cores](#)
- [implement_debug_core](#)
- [report_debug_core](#)
- [report_property](#)
- [set_property](#)
- [write_chipscope_cdc](#)

create_debug_port

Create a new debug port.

Syntax

```
create_debug_port [-quiet] [-verbose] name type
```

Returns

New debug_port object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of the debug core instance
<i>type</i>	Type of the new debug port

Categories

Description

Defines a new port to be added to an existing ILA debug core. The port provides connection points to a debug core to attach nets from the design for debug purposes.

When a new debug core is created using the **create_debug_core** command, it includes a CLK and trigger (TRIG) port by default. However, you can also add DATA and trigger_output (TRIG_OUT) ports to the debug core as well as additional TRIG ports.

A port can have one or more connection points to support one or more nets to debug. As a default new ports are defined as having a width of 1, allowing only one net to be attached. You can change the port width of TRIG and DATA ports to support multiple signals using the **set_property port_width** command (see Examples).

Note CLK and TRIG_OUT ports can only have a width of 1.

You can connect signals to ports using the **connect_debug_port** command, and disconnect signals with the **disconnect_debug_port** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the ILA debug core to add the new port to. The debug core must already exist in the project having been created with **create_debug_port**.

type - (Required) The type of debug port to insert. There are four port types supported: CLK, DATA, TRIG, and TRIG_OUT. Refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* for more information on port types and purpose.

Note Each ILA debug core can have only one CLK, DATA, and TRIG_OUT port. However, you can create multiple trigger (TRIG) ports.

Examples

The following example creates a new ChipScope debug core, and then adds a DATA port to that core:

```
create_debug_core myCore labtools_ila_v2
create_debug_port myCore DATA
```

The following example creates a new port on the myCore debug core, and then sets the port width to 8, and begins connecting signals to the port:

```
create_debug_port myCore TRIG
set_property PORT_WIDTH 8 [get_debug_ports myCore/TRIG0]
connect_debug_port -channel_start_index 1 myCore/TRIG0 {m1_cyc_i \
    m1_ack_o m1_err_o m1_rty_o}
```

Note The debug core is referenced by its name, and the debug port is referenced by the core_name/port_name.

See Also

- [connect_debug_port](#)
- [create_debug_core](#)
- [disconnect_debug_port](#)
- [set_property](#)

create_drc_check

Create a user defined drc rule.

Syntax

```
create_drc_check [-category arg] -name arg [-desc arg] [-msg arg]
-rule_body arg [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-category]	Specify the category for this rule. This is the major grouping for your rule. It is optional and will default to User Defined. Default: User Defined
-name	Specify the name for this rule. This must be of the form PREFIX-id where XXXX is a 4-6 letter abbreviation and id is an integer identifying a particular rule. Similar rules should have the same abbreviation and each a unique id.
[-desc]	Specify the short description for this rule. It is optional and will default to . Default: User rule - default description
[-msg]	Specify the full description for this rule. Including the substitutions. Values are: %MSG_STRING %NETLIST_ELEMENT %SITE_GROUP %CLOCK_REGION %BANK.
-rule_body	The string representing the body of the rule. This can be a tcl proc name or any string of tcl code to be evaluated.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

DRC, Object

Description

Create a new user-defined DRC rule check, `drc_check`, for use by the tool when running `report_drc`.

This command allows you to define a unique name or abbreviation for the user-defined rule check, optionally group the rule into a special category and provide a description of the rule, define a general placeholder message for the check when violations are encountered, and refer to the Tcl code associated with the design rule check to be run during the **report_drc** command.

The general placeholder message defined in this command is populated with specific information related to the design objects and violations found by the Tcl checker procedure, and by the **create_drc_violation** command.

The process in brief is:

- Write a Tcl checker procedure to define the method applied when checking the user-defined rule, and the objects to check against the rule. The Tcl checker procedure is defined in a separate Tcl script that must be loaded by the **source** command prior to running **report_drc**.
- Use **create_drc_violation** in the Tcl checker to identify and flag violations found when checking the rule against a design.
- Define a user-defined DRC rule check using the **create_drc_check** command that calls the Tcl checker proc from the **-rule_body**.
- Create a rule deck using the **create_drc_ruledeck** command, and add the user-defined rule check to the rule deck using the **add_drc_checks** command.
- Run **report_drc**, and specify either the rule deck, or the user-defined rule check to check for violations.

If a **drc_check** of the specified name is already defined in the tool, an error is returned. In this case, to overwrite or redefine an existing **drc_check**, you must first delete the check using the **delete_drc_check** command.

The DRC rule check object features the **is_enabled** property that can be set to TRUE or FALSE using the **set_property** command. When a new rule check is created, the **is_enabled** property is set to TRUE as a default. Set the **is_enabled** property to FALSE to disable the rule check from being used when **report_drc** is run. This lets you create new DRC checks, add them to rule decks using **add_drc_checks**, and then enable them or disable them as needed without having to remove them from the rule deck.

Each user defined DRC rule check has the 'USER_DEFINED' property, which lets you quickly identify and select user-defined rule checks.

Arguments

-category arg - (Optional) Defines a grouping for the rule. The default is "User Defined". This is used as the first level of hierarchy in the GUI when listing DRC rules. All newly created DRC checks are also added to the "all" category used by default by the **report_drc** command.

-name arg - (Required) The unique name for the design rule. This should match the name used by the **create_drc_violation** commands in the Tcl checker procedure specified in **-rule_body**. The name will appear in the DRC report with any associated violations. The name should consist of a short 4 to 6 letter abbreviation for the rule group, and an ID to differentiate it from other checks in the same group, for instance ABCD-1 or ABCD-23.

-desc arg - (Optional) A brief description of the rule. The default is "User Rule". This is displayed when listing DRC rules in the GUI. The description is also used in the DRC report and summary.

-msg arg - (Optional) This is the message displayed when a violation of the rule is found. The message can include placeholders for dynamic substitution with design elements found in violation of the rule. The design data is substituted into the message at the time **report_drc** is run. Each substitution key has a long form, and a short form as shown below. Valid substitutions keys are:

- **%MSG_STRING (%STR)** - This is the message string defined by the **-msg** option in the **create_drc_violation** command for the specific violation.

Note %STR is the default message for the **create_drc_check** command if the **-msg** option is not specified. In this case, any message defined by **create_drc_violation** in the **-rule_body** is simply passed through to the DRC report.

- **%NETLIST_ELEMENT (%ELG)** - Netlist elements including cells, pins, ports, and nets.
- **%SITE_GROUP (%SIG)** - Device site.
- **%CLOCK_REGION (%CRG)** - Clock region.
- **%BANK (%PBG)** - Package IO bank.

-rule_body arg - (Required) This is the name of the Tcl procedure which defines the rule checking functionality. The Tcl procedure can be embedded here, into the **-rule_body** option, or can be separately defined in a Tcl script that must be loaded with the **source** command when the tool is launched, or prior to running the **report_drc** command.

The Tcl checker procedure can create DRC violation objects, using the **create_drc_violation** command, containing the design elements that are associated with a design rule violation. The tool populates the substitution keys in the message defined by **-msg** with the design elements from the violation object.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example defines a new design rule check named RAMW-1, with the category and description defined, using the default severity of Warning, and calling the **dataWidthCheck** procedure when the check is run:

```
create_drc_check -name {RAMW-1} -category {RAMB} \
  -desc {Data Width Check} -rule_body dataWidthCheck
```

The following Tcl script defines the **dataWidthCheck** procedure which is called by the **-rule_body** argument of the RAMW-1 check. This Tcl script file must be loaded into the tool using the **source** command, prior to running the report_drc command.

```
# This is a simplistic check -- report BRAM cells with WRITE_WIDTH_B wider than 36.
proc dataWidthCheck {} {
    # list to hold violations
    set vios {}
    # iterate through the objects to be checked
    foreach bram [get_cells -hier -filter {PRIMITIVE_SUBGROUP == bram}] {
        set bwidth [get_property WRITE_WIDTH_B $bram]
        if { $bwidth > 36 } {
            # define the message to report when violations are found
            set msg "On cell %ELG, WRITE_WIDTH_B is $bwidth"
            set vio [ create_drc_violation -name {RAMW-1} -msg $msg $bram ]
            lappend vios $vio
        }
    }
    if {[llength $vios] > 0} {
        return -code error $vios
    } else {
        return {}
    }
}
create_drc_check -name {RAMW-1} \
    -category {RAMB Checks} \
    -desc {Data Width Check} \
    -rule_body dataWidthCheck
```

Note The script file can contain both the Tcl checker procedure, and the **create_drc_check** command that defines it for use by **report_drc** command. In this case, when the Tcl script file is sourced, both the **dataWidthCheck** proc and the RAMW-1 design rule check are loaded into the tool.

See Also

- [add_drc_checks](#)
- [create_drc_ruledeck](#)
- [create_drc_violation](#)
- [delete_drc_check](#)
- [get_drc_checks](#)
- [get_drc_vios](#)
- [report_drc](#)

create_drc_ruledeck

Create one or more user defined drc rule deck objects.

Syntax

```
create_drc_ruledeck [-quiet] [-verbose] ruledecks...
```

Returns

Drc_ruledeck

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>ruledecks</i>	Names of rule decks to create

Categories

DRC, Object

Description

Create one or more user-defined rule decks for use when running report_drc.

A drc_ruledeck object is a collection of design rule checks, grouped for convenience, to be run at different stages of the FPGA design flow, such as during I/O planning or placement. The tool comes with a set of factory predefined rule decks. Use the **get_drc_ruledecks** command to return a list of the currently defined rule decks.

The rule decks created by this command are empty, without any checks. You must add design rule checks to the rule deck using the **add_drc_checks** command. Checks can be removed from a rule deck using the **remove_drc_checks** command. To see a list of design rule checks that are available to include in the ruledeck, use the **get_drc_checks** command.

This command returns the list of drc_ruledecks created.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

ruledecks - (Required) Specify the name of one or more user-defined DRC rule decks to create.

Examples

The following example creates two new drc_ruledeck objects:

```
create_drc_ruledeck my_rules project_rules
```

See Also

- [add_drc_checks](#)
- [delete_drc_ruledeck](#)
- [get_drc_checks](#)
- [get_drc_ruledecks](#)
- [remove_drc_checks](#)
- [report_drc](#)

create_drc_violation

Create a drc violation.

Syntax

```
create_drc_violation -name arg [-severity arg] [-msg arg] [-quiet]  
[-verbose] [objects...]
```

Returns

Nothing

Usage

Name	Description
-name	Specify the name for this rule. This is typically a 4-6 letter specification for your rule.
[-severity]	Specify severity level for a drc rule. Default: WARNING. Values: FATAL, ERROR, CRITICAL WARNING, WARNING, ADVISORY.
[-msg]	Specify your message string for this drc rule.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[objects]	Cells, ports, pins, nets, clock regions, sites, package banks to query.

Categories

[DRC](#), [Report](#)

Description

Create a DRC violation object and manage the list of design objects associated with the violation for reporting by the **report_drc** command.

The **create_drc_violation** command is specified as part of the Tcl checker procedure that defines and implements the checking feature of a user-defined design rule check created by the **create_drc_check** command. A violation object is created by the Tcl checker each time a violation of the design rule is encountered.

The process in brief is:

- Write a Tcl checker procedure to define the method applied when checking the user-defined rule, and the objects to check against the rule. The Tcl checker procedure is defined in a separate Tcl script that must be loaded by the **source** command prior to running **report_drc**.
- Use **create_drc_violation** in the Tcl checker to identify and flag violations found when checking the rule against a design.
- Define a user-defined DRC rule check using the **create_drc_check** command that calls the Tcl checker proc from the **-rule_body**.
- Create a rule deck using the **create_drc_ruledeck** command, and add the user-defined rule check to the rule deck using the **add_drc_checks** command.
- Run **report_drc**, and specify either the rule deck, or the user-defined rule check to check for violations.

Violations are reported by the **report_drc** command, and violation objects can be returned by the **get_drc_vios** command.

Arguments

-name arg - (Required) The name of the design rule check associated with the violation. This should be the same name used by the **create_drc_check** command which calls the associated Tcl checker procedure from its **-rule_body** argument. Messages from the **create_drc_violation** command are passed up to the **drc_check** with the same **-name**.

-severity arg - (Optional) The severity of the violation. The default severity level for user-defined DRCs is WARNING. The supported values are:

- FATAL
- ERROR
- "CRITICAL WARNING"
- WARNING
- ADVISORY

Note The **SEVERITY** is stored as a property on the DRC rule associated with the DRC violation object. The severity can be changed on the rule by using the **set_property** command on the associated DRC check.

-msg arg - (Optional) This is a violation specific message that is substituted for the general string variable (%STR) specified in the optional placeholder message defined in the **create_drc_check** command.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) Cell, port, pin, net, clock region, site, and package I/O bank objects associated with violations found by the Tcl checker procedure that are substituted into the placeholder message of the `drc_object` with the same **-name**. Design objects map to substitution keys in the message as follows:

- %ELG - netlist elements such as cells, ports, pins, and nets.
- %CRG - clock regions.
- %SIG - device sites.
- %PBG - package I/O banks.

Note Both the order and the type of *objects* passed from the `create_drc_violation` command must match the **-msg** specification from the **create_drc_check** command, or the expected substitution will not occur

Examples

The following Tcl script defines the **dataWidthCheck** procedure which is called by the **-rule_body** argument of the RAMW-1 check. This Tcl script file must be loaded into the tool using the **source** command, prior to running the `report_drc` command.

Some features of the Tcl checker proc to notice are:

- A list variable is created to store violations (**\$vios**)
- A violation object is created, and added to the list variable, each time a violation is found.
- The placeholder key **%ELG** in the **\$msg** string is dynamically substituted with the specific **\$bram** cell associated with the violation.
- The **dataWidthCheck** proc returns an error code when any violations are found (**\$vios > 0**) to inform the **report_drc** command of the results of the check.
- The list of violations is passed along with the return code, and the violations are reported by **report_drc**.

```
# This is a simplistic check -- report BRAM cells with WRITE_WIDTH_B wider than 36.
proc dataWidthCheck {} {
    # list to hold violations
    set vios {}
    # iterate through the objects to be checked
    foreach bram [get_cells -hier -filter {PRIMITIVE_SUBGROUP == bram}] {
        set bwidth [get_property WRITE_WIDTH_B $bram]
        if { $bwidth > 36 } {
            # define the message to report when violations are found
            set msg "On cell %ELG, WRITE_WIDTH_B is $bwidth"
            set vio [ create_drc_violation -name {RAMW-1} -msg $msg $bram ]
            lappend vios $vio
        }
    }
    if {[llength $vios] > 0} {
        return -code error $vios
    } else {
        return {}
    }
}
create_drc_check -name {RAMW-1} \
    -category {RAMB Checks} \
    -desc {Data Width Check} \
    -rule_body dataWidthCheck
```

Note The script file can contain both the Tcl checker procedure, and the **create_drc_check** command that defines it for use by **report_drc** command. In this case, when the Tcl script file is sourced, both the **dataWidthCheck** proc and the RAMW-1 design rule check are loaded into the tool.

See Also

- [add_drc_checks](#)
- [create_drc_ruledeck](#)
- [create_drc_check](#)
- [get_drc_checks](#)
- [get_drc_vios](#)
- [report_drc](#)
- [set_property](#)

create_fileset

Create a new fileset.

Syntax

```
create_fileset [-constrset] [-simset] [-blockset]
[-clone_properties arg] -define_from arg [-quiet] [-verbose] name
```

Returns

New fileset object

Usage

Name	Description
[-constrset]	Create fileset as constraints fileset (default)
[-simset]	Create fileset as simulation source fileset
[-blockset]	Create fileset as block source fileset
[-clone_properties]	Fileset to initialize properties from
-define_from	Name of the module in the source fileset to be the top of the blockset
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of the fileset to be create

Categories

[Project](#), [Simulation](#)

Description

Defines a new fileset within your project.

A fileset is a list of files with a specific function within the project. One or more constraint files is a constraint set (**-constrset**); one or more simulation test benches is a simulation set (**-simset**). Only one fileset option can be specified when using the **create_fileset** command. As a default, the tool will create a constraint fileset if the type is not specified.

The **create_fileset** command returns the name of the newly created fileset, or will return an error message unless the **-quiet** argument has been specified.

Arguments

-constrset - (Optional) Creates a constraint set to hold one or more constraint files. This is the default fileset created if neither the **-constrset** or **-simset** argument is specified.

-simset - (Optional) Create a simulation fileset to hold one or more simulation source files. You can only specify one type of fileset argument, either **-constrset** or **-simset**. You will get an error if both are specified.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the fileset to be created.

Examples

The following example creates a new constraint file set named `constraints2`:

```
create_fileset -constrset -quiet constraints2
```

Note With **-quiet** specified, the tool will not return anything if it encounters an error in trying to create the specified fileset.

The following example creates a new simulation fileset named `sim_1`:

```
create_fileset -simset sim_1
```

Files can be added to the newly created fileset using the **add_files** command.

See Also

- [add_files](#)
- [current_fileset](#)

create_generated_clock

Create a generated clock object.

Syntax

```
create_generated_clock [-name arg] [-source args] [-edges args]
[-divide_by arg] [-multiply_by arg] [-combinational] [-duty_cycle arg]
[-edge_shift args] [-add] [-master_clock arg] [-quiet]
[-verbose] objects
```

Returns

New clock object

Usage

Name	Description
<code>[-name]</code>	Generated clock name
<code>[-source]</code>	Master clock source object pin/port
<code>[-edges]</code>	Edge Specification
<code>[-divide_by]</code>	Period division factor: Value ≥ 1 Default: 0
<code>[-multiply_by]</code>	Period multiplication factor: Value ≥ 1 Default: 0
<code>[-combinational]</code>	Create a divide_by 1 clock through combinational logic
<code>[-duty_cycle]</code>	Duty cycle for period multiplication: Range: 0.0 to 100.0 Default: 0.0
<code>[-edge_shift]</code>	Edge shift specification
<code>[-add]</code>	Add to the existing clock in source_objects
<code>[-master_clock]</code>	Use this clock if multiple clocks present at master pin
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>objects</code>	List of clock source ports, pins, or nets

Categories

[SDC](#), [XDC](#)

Description

Generate a new clock object from an existing physical clock object in the design.

Note This command returns the name of the clock object that is created.

Arguments

-name *arg* - (Optional) The name of the generated clock to create on the specified object. If no name is specified, the generated clock will be given the name of the *object* it is assigned to. If assigned to multiple *objects*, the name will be the first object in the list.

-source *arg* - (Optional) The pin or port of the master clock from which to derive the generated clock. The master clock must be a previously defined physical clock, not a virtual clock; but can be a primary clock or another generated clock. If the source pin or port currently has multiple clocks defined, the **-master_clock** option must be used to identify which clock on the source is to be used to define the generated clock.

-edges *arg* - (Optional) Specifies the edges of the master clock to use in defining transitions on the generated clock. Specify transitions on the generated clock in a sequence of 1, 2, 3, by referencing the appropriate edge count from the master clock in numerical order, counting from the first edge. The sequence of transitions on the generated clock defines the period and duty cycle of the clock: position 1 is the first rising edge of the generated clock, position 2 is the first falling edge of the generated clock and so defines the duty cycle, position 3 is the second rising edge of the generated clock and so defines the clock period. Enclose multiple edge numbers in braces {}. See the example below for specifying edge numbers.

-divide_by *arg* - (Optional) Divide the frequency of the master clock by the specified value to establish the frequency of the generated clock object. The value specified must be ≥ 1 , and must be specified as an integer.

-multiply_by *arg* - (Optional) Multiply the frequency of the master clock by the specified value to establish the frequency of the generated clock object. The value specified must be ≥ 1 , and must be specified as an integer.

-combinational - (Optional) Define a combinational path to create a "-divide_by 1" generated clock.

-duty_cycle *arg* - (Optional) The duty cycle of the generated clock defined as a percentage of the new clock period when used with the **-multiply_by** argument. The value is specified as a percentage from 0.0 to 100.

-edge_shift *arg* - (Optional) Shift the edges of the generated clock by the specified values relative to the master clock. See the example below for specifying edge shift.

-add - (Optional) Add the generated clock object to an existing clock group specified by *objects*.

Note **-master_clock** and **-name** options must be specified with **-add**

-master_clock *arg* - (Optional) If there are multiple clocks found on the source pin or port, the specified clock is the one to use as the master for the generated clock object.

Note **-add** and **-name** options must be specified with **-master_clock**

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) The pin or port objects to which the generated clock should be assigned. If the specified objects already have a clock defined, use the **-add** option to add the new generated clock and not overwrite any existing clocks on the object.

Examples

The following example defines a generated clock that is divided from the master clock found on the specified CLK pin. Since **-name** is not specified, the generated clock is assigned the same name as the pin it is assigned to:

```
create_generated_clock -divide_by 2 -source [get_pins clkgen/sysClk] fftEngine/clk
```

The following example defines a generated clock named CLK1 from the specified source clock, specifying the edges of the master clock to use as transition points for the generated clock, with edges shifted by the specified amount. In this example, the **-edges** option indicates that the second edge of the source clock is the first rising edge of the generated clock, the third edge of the source clock is the first falling edge of the generated clock, and the eighth edge of the source clock is the second rising edge of the generated clock. These values determine the period of the generated clock as the time from edge 2 to edge 8 of the source clock, and the duty cycle as the percentage of the period between edge 2 and edge 3 of the source clock. In addition, each edge of the generated clock is shifted by the specified amount:

```
create_generated_clock -name CLK1 -source CMB/CLKIN -edges {2 3 8} \
-edge_shift {0 -1.0 -2.0} CMB/CLKOUT
```

Note The waveform pattern of the generated clock is repeated based on the transitions defined by the **-edges** argument.

This example creates two generated clocks from the output of a MUX, using **-master_clock** to identify which clock to use, using **-add** to assign the generated clocks to the Q pin of a flip flop, and using **-name** to define a name for the generated clock, since the object it is assigned to has multiple clocks assigned:

```
create_generated_clock -source [get_pins muxOut] -master_clock M_CLKA \
-divide_by 2 -add -name gen_CLKA [get_pins flop_Q]
create_generated_clock -source [get_pins muxOut] -master_clock M_CLKB \
-divide_by 2 -add -name gen_CLKB [get_pins flop_Q]
```

See Also

- [check_timing](#)
- [create_clock](#)
- [get_generated_clocks](#)
- [get_pins](#)
- [set_clock_latency](#)
- [set_clock_uncertainty](#)
- [set_propagated_clock](#)

create_hw_sio_link

Create a new link between hardware RX and TX endpoints. There must be at least one hardware TX or RX endpoint specified. If one is missing, the endpoint will be treated as Unknown. The unknown endpoint can be renamed in a link property.

Syntax

```
create_hw_sio_link [-quiet] [-verbose] [hw_sio_rx] [hw_sio_tx]
```

Returns

The new hardware SIO link

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_sio_rx]	RX endpoint. Default: None
[hw_sio_tx]	TX endpoint. Default: None

Categories

[Hardware](#)

create_hw_sio_linkgroup

Create a new hardware SIO link group.

Syntax

```
create_hw_sio_linkgroup [-quiet] [-verbose] hw_sio_links
```

Returns

The new hardware SIO link group

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_sio_links</i>	hardware SIO links

Categories

[Hardware](#)

create_hw_sio_scan

Create a new hardware SIO scan.

Syntax

```
create_hw_sio_scan [-quiet] [-verbose] scan_type [hw_sio_rx]
```

Returns

The new hardware SIO scan

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>scan_type</i>	Scan Type Options: 1d_bathtub, 2d_full_eye
<i>[hw_sio_rx]</i>	RX endpoint to perform scan on. Default: None

Categories

[Hardware](#)

create_interface

Create a new I/O port interface.

Syntax

```
create_interface [-parent arg] [-quiet] [-verbose] name
```

Returns

New interface object

Usage

Name	Description
[-parent]	Assign new interface to this parent interface
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name for new I/O port interface

Categories

[PinPlanning](#)

Description

Creates a new interface for grouping scalar or differential I/O ports.

Arguments

-parent *arg* - (Optional) Assign the new interface to the specified parent interface.

Note If the specified parent interface does not exist, an error will be returned.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the I/O port interface to create.

Examples

Create a new USB interface:

```
create_interface USB0
```

Create an Ethernet interface within the specified parent interface:

```
create_interface -parent Top_Int ENET0
```

See Also

- [delete_interface](#)
- [create_port](#)
- [make_diff_pair_ports](#)
- [place_ports](#)
- [remove_port](#)
- [set_package_pin_val](#)
- [split_diff_pair_ports](#)

create_ip

Create an instance of a configurable IP and add it to the fileset.

Syntax

```
create_ip [-vlnv arg] -module_name arg [-dir arg] [-vendor arg]
[-library arg] [-name arg] [-version arg] [-quiet] [-verbose]
```

Returns

List of file objects that were added

Usage

Name	Description
<code>[-vlnv]</code>	VLNV string for the Catalog IP from which the new IP will be created (colon delimited Vendor, Library, Name, Version)
<code>-module_name</code>	Name for the new IP that will be added to the project
<code>[-dir]</code>	Directory path for remote IP to be created and managed outside the project
<code>[-vendor]</code>	IP Vendor name
<code>[-library]</code>	IP Library name
<code>[-name]</code>	IP Name
<code>[-version]</code>	IP Version
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

IPFlow

Description

This command creates an XCI file for a configurable IP core from the IP catalog, and adds it to the source files of the current project. This creates an IP source object which must be instantiated into the HDL design to create an instance of the IP core in the netlist.

For multiple instances of the same core, simply instantiate the core module into the HDL design as many times as needed. However, to use the same IP core with different customizations, use the **create_ip** command to create separate IP source objects.

The **create_ip** command is used to import IP cores from the current IP catalog. Use the **import_ip** command to read existing XCI and XCO files directly, without having to add IP to a catalog.

This command returns a transcript of the IP generation process, concluding with the file path and name of the imported IP core file.

Note IP cores are native to Vivado, and can be customized and regenerated within that tool. The **convert_ip** command lets you to convert legacy IP to native IP supported by Vivado.

Arguments

-vlnv <arg> - (Optional) Specifies the VLN string for the existing Catalog IP from which the new IP will be created. The VLN is the *Vendor:Library:Name:Version* string which identifies the IP in the catalog. The VLN string maps to the IPDEF property on the IP core.

Note You must specify either **-vlnv** or all of **-vendor**, **-library**, **-name**, and **-version**

-module_name <arg> - (Required) Specifies the name for the new IP instance that will be added to the project

-dir <arg> - (Optional) The directory to write the IP core files into. If this option is not specified, the IP core files (.xci, .ngc, .veo...) are written into the hierarchy of the <project_name>.srcs directory.

-vendor <arg> - (Optional) Specifies the vendor name for the IP's creator.

-library <arg> - (Optional) Specifies the IP library from which the core should be added.

-name <arg> - (Optional) Specifies the name of the IP core in the catalog.

-version <arg> - (Optional) Specifies the version number for the IP core.

Note You must specify either **-vlnv** or all of **-vendor**, **-library**, **-name**, and **-version**

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The example below imports the IP core specified by the **-vlnv** string, and gives it the specified module name in the current project:

```
create_ip -vlnv xilinx.com:ip:c_addsub:11.0 -module_name test_addr
```

The following example, from Vivado, creates an IP block with the specified **-vendor**, **-library**, **-name**, **-version** values, and assigns it the specified module name. After the IP is created, attributes of the IP are customized using **set_property** commands. Then the instantiation template and the synthesis targets are generated for the IP:

```
create_ip -name c_addsub -version 11.0 -vendor xilinx.com -library ip \
  -module_name c_addsub_v11_0_0
set_property -name CONFIG.Component_Name -value {c_addsub_v11_0_0} \
  -objects [get_ips c_addsub_v11_0_0]
set_property -name CONFIG.A_Width -value {32} \
  -objects [get_ips c_addsub_v11_0_0]
set_property -name CONFIG.B_Width -value {32} \
  -objects [get_ips c_addsub_v11_0_0]
set_property -name CONFIG.Add_Mode -value {Add_Subtract} \
  -objects [get_ips c_addsub_v11_0_0]
set_property -name CONFIG.C_In -value {true} \
  -objects [get_ips c_addsub_v11_0_0]
generate_target {instantiation_template synthesis} \
  [get_files C:/Data/c_addsub_v11_0_0/c_addsub_v11_0_0.xci \
  -of_objects [get_filesets sources_1]]
```

See Also

- [generate_target](#)
- [import_ip](#)
- [upgrade_ip](#)
- [validate_ip](#)

create_macro

Create A Macro.

Syntax

```
create_macro [-quiet] [-verbose] name
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Macro to create.

Categories

XDC

Description

Create a macro for the relative placement of cells.

Macros are primarily used to place small groups of associated cells together to improve resource efficiency and enable faster interconnections. The **create_macro** command lets you define macros in an open synthesized or implemented design for relative placement by **place_design**, like RPMs defined by the RLOC constraint in RTL source files. Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on defining relatively placed macros.

After creating the macro, specific cells can be assigned to the macro using the **update_macro** command. To change a currently defined macro, you must delete the macro with **delete_macro**, recreate the macro, and update the macro with the new contents. You cannot simply overwrite an existing macro.

Use **delete_macro** to delete a defined macro. Use **get_macros** to return a list of currently defined macros in the design.

This command operates silently and does not return anything.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) Specify the name of the macro to create.

Examples

The following example creates a macro called :

```
create_macro usbMacro1
```

See Also

- [delete_macros](#)
- [get_macros](#)
- [place_design](#)
- [update_macro](#)

create_net

Create nets in the current design.

Syntax

```
create_net [-from arg] [-to arg] [-quiet] [-verbose] nets...
```

Returns

Nothing

Usage

Name	Description
[-from]	Starting bus index
[-to]	Ending bus index
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>nets</i>	Names of nets to create

Categories

[Netlist](#)

Description

Create new nets in the current netlist of an open Synthesized or Implemented Design.

Note You cannot add nets to library macros, or macro-primitives.

Nets can be created hierarchically from the top-level of the design, or within any level of the hierarchy by specifying the hierarchical net name.

Bus nets can be created with increasing or decreasing bus indexes, using negative and positive index values.

New nets are unconnected in the netlist at the time of creation. You must connect nets as desired using the **connect_net** command. Connected nets can be unconnected using the **disconnect_net** command, and can be removed from the netlist using the **remove_net** command.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source files set, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

Arguments

-from *arg* - (Optional) The starting index of a new bus.

-to *arg* - (Optional) The ending index of a new bus.

Note Specifying **-from** or **-to** without the other will result in a one-bit bus with index value specified by the **-from** or **-to** argument.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

nets - (Required) The names of nets to create. Net names can be specified from the top-level, as name only (net1), or can be specified within the design hierarchy by specifying the hierarchical net name (cell1/cellA/net1).

Example

The following example creates a new 24-bit bus in the current Synthesized or Implemented Design:

```
create_net tempBus -from 23 -to 0
```

See Also

- [connect_net](#)
- [create_pin](#)
- [create_port](#)
- [disconnect_net](#)
- [get_nets](#)
- [remove_net](#)
- [resize_net_bus](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

create_pblock

Create a new Pblock.

Syntax

```
create_pblock [-quiet] [-verbose] name
```

Returns

New pblock object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of the new pblock

Categories

[XDC](#), [Floorplan](#)

Description

Defines a Pblock to allow you to add logic instances for floorplanning purposes.

You can add logic elements to the Pblock using the **add_cells_to_pblock** command, and then place the Pblocks onto the fabric of the FPGA using the **resize_pblocks** command. The **resize_pblock** command can also be used to manually move and resize pblocks.

You can nest one Pblock inside another for hierarchical floorplanning using the **-parent** option as shown in the first example. You can also nest an existing Pblock inside another Pblock using the **set_property** command to define the PARENT property as shown in the second example.

Arguments

-parent *arg* - (Optional) The name of the parent Pblock to allow creation of nested Pblocks. If the parent is not specified, the default parent of Root is assumed, placing the Pblock at the top of the design. You can use the **get_pblocks** command to report currently defined Pblocks that can be used as parents.

Note If the specified **parent** does not exist an error will be returned

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the Pblock to be created.

Examples

The following example creates a Pblock called Video1 inside another Pblock called Vid_Array:

```
create_pblock -parent Vid_Array Video1
```

The following example creates Pblocks called cpu1 and cpu2, and creates a third Pblock called cpuEngine. Then cpu1 and cpu2 are nested inside cpuEngine using the set_property command:

```
create_pblock cpu1
create_pblock cpu2
create_pblock cpuEngine
set_property PARENT cpuEngine [get_pblocks {cpu1 cpu2}]
```

See Also

- [add_cells_to_pblock](#)
- [get_pblocks](#)
- [place_pblocks](#)
- [resize_pblock](#)
- [set_property](#)

create_pin

Create pins in the current design.

Syntax

```
create_pin [-from arg] [-to arg] -direction arg [-quiet]  
[-verbose] pins...
```

Returns

Nothing

Usage

Name	Description
<code>[-from]</code>	Starting bus index
<code>[-to]</code>	Ending bus index
<code>-direction</code>	Pin direction Values: IN, OUT, INOUT
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>pins</code>	Names of pins to create

Categories

[Netlist](#)

Description

Add single pins or bus pins to the current netlist of an open Synthesized or Implemented Design. You may define attributes of the pin such as direction and bus width, as well as the pin name.

Bus pins can be created with increasing or decreasing bus indexes, using negative and positive index values.

The pins must be created on an existing cell instance, or it is considered a top-level pin which should be created using the **create_port** command. If the instance name of a cell is not specified as part of the pin name, an error will be returned.

Note You cannot add pins to library macros, or macro-primitives.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source filesset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

Arguments

-from *arg* - (Optional) The starting index of a bus pin.

-to *arg* - (Optional) The ending index of a bus pin.

-direction - (Required) The direction of the pin. Valid values are IN, OUT, and INOUT.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pins - (Required) The name of the pins to create. You must specify the pin names hierarchically from the cell instance the pin is assigned to. Pins created at the top-level of the design are ports, and should be created with the **create_port** command.

Examples

The following example creates a new input pin on the cpuEngine module with the specified pin name:

```
create_pin -direction IN cpuEngine/inPin
```

The following example sets the hierarchy separator, creates a new black box instance of the reference cell, and creates a twenty-four bit bidirectional bus for that instance:

```
set_hierarchy_separator |  
create_cell -reference dmaBlock -black_box usbEngine0|myDMA  
create_pin -direction INOUT -from 0 -to 23 usbEngine0|myDMA|dataBus
```

See Also

- [create_cell](#)
- [create_net](#)
- [create_port](#)
- [connect_net](#)
- [disconnect_net](#)
- [remove_cell](#)
- [remove_pin](#)
- [resize_pin_bus](#)
- [set_hierarchy_separator](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

create_port

Create scalar or bus port.

Syntax

```
create_port -direction arg [-from arg] [-to arg] [-diff_pair]
[-interface arg] [-quiet] [-verbose] name [negative_name]
```

Returns

List of port objects that were created

Usage

Name	Description
-direction	Direction of port. Valid arguments are IN, OUT and INOUT
[-from]	Beginning index of new bus
[-to]	Ending index of new bus
[-diff_pair]	Create differential pair of ports
[-interface]	Assign new port to this interface
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of the port
<i>[negative_name]</i>	Optional negative name of a diff-pair

Categories

[PinPlanning](#)

Description

Creates a port and specifies such parameters as direction, width, single-ended or differential, and optionally assigns it to an existing interface. New ports are added at the top-level of the design hierarchy.

Bus ports can be created with increasing or decreasing bus indexes, using negative and positive index values.

The **create_port** command can be used to create a new port in an I/O Planning project, or while editing the netlist of an open Synthesized or Implemented design.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

Arguments

-direction - (Required) The direction of the port. Valid arguments are IN, OUT, and INOUT.

-from *arg* - (Optional) The beginning index of a new bus. A bus can start from a negative index value.

-to *arg* - (Optional) The ending index of a new bus. A bus can end on a negative index value.

-diff_pair - (Optional) Create the specified port as a differential pair of ports. In this case both a positive and negative side port will be created. If only *name* is specified, the positive side port will be assigned the specified *name*, and the negative side port will be assigned *name_N*. If both *name* and *negative_name* are specified, the positive side port will be assigned *name*, and the negative side port will be assigned *negative_name*.

-interface *arg* - (Optional) Assign the port to the specified interface.

Note The interface must first be defined with the **create_interface** command.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the port to create. If **-diff_pair** is specified, *name* is assigned to the positive side port, and the negative side port is *name_N*.

negative_name - (Optional) Use this option to specify the name of the negative side port when **-diff_pair** is specified. In this case, *name* will be assigned to the positive side port, and *negative_name* will be assigned to the negative side port.

Examples

The following example creates a new input port, named PORT0:

```
create_port -direction IN PORT0
```

The following example creates a new interface called Group1, and then creates a four-bit, differential pair output bus utilizing the specified interface. Since the bus ports are defined as differential pairs, and only *name* is specified, the negative side ports are automatically named D_BUS_N:

```
create_interface Group1
create_port -direction OUT -from 0 -to 3 -diff_pair -interface Group1 D_BUS
```

Note This command results in the creation of eight ports: D_BUS[0] D_BUS_N[0] D_BUS[1] D_BUS_N[1] D_BUS[2] D_BUS_N[2] D_BUS[3] D_BUS_N[3]

With only *name* specified, the following example creates differential pair output ports named *data* and *data_N*.

```
create_port -direction OUT -diff_pair data
```

With both *name* and *negative_name* specified, the following example creates differential pair output ports named *data_P* and *data_N*.

```
create_port -direction OUT -diff_pair data_P data_N
```

See Also

- [create_interface](#)
- [make_diff_pair_ports](#)
- [place_ports](#)
- [remove_port](#)
- [resize_port_bus](#)
- [split_diff_pair_ports](#)

create_project

Create a new project.

Syntax

```
create_project [-part arg] [-force] [-quiet] [-verbose] [name] [dir]
```

Returns

New project object

Usage

Name	Description
[-part]	Target part
[-force]	Overwrite existing project directory
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[name]</i>	Project name
<i>[dir]</i>	Directory where the project file is saved Default: .

Categories

[Project](#)

Description

Creates a project file (.xpr) in the specified directory.

Arguments

-part *arg* - (Optional) Specifies the Xilinx part to be used for the project. This can be changed after the project is created. If the **-part** option is not specified, the default part will be used.

-force - (Optional) This option is required to overwrite an existing project. If the project name is already define in the specified *dir* then you must also specify the **-force** option for the tool to overwrite the existing project.

Note If the existing project is currently open in the tool, the new project will overwrite the existing project on the disk, but both projects will be opened in the tool. In this case you should probably run the **close_project** command prior to running **create_project**.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Optional) This argument does not require a parameter name, however, it must appear before the specified *dir*. Since these commands do not have parameters, the tool interprets the first argument as *name* and uses the second argument as *dir*. A project file is created *name.xpr*, and a project data folder is also created *name.data* and both are written into the specified directory *dir*.

Note The project file created by the tool is an RTL source file by default. You must use the **set_property** command to set the DESIGN_MODE property to change the project from an RTL source project to another type of project, such as an I/O Pin Planning project for instance.

dir - (Optional) This argument specifies the directory name to write the new project file into. If the specified directory does not exist a new directory will be created. If the directory is specified with the complete path, the tool uses the specified path name. However, if *dir* is specified without a path, the tool looks for or creates the directory in the current working directory, or the directory from which the tool was launched.

Note When creating a project in GUI-mode, the tool appends the filename *name* to the directory name *dir* and creates a project directory with the name *dir/name* and places the new project file and project data folder into that project directory.

Examples

The following example creates a project called Project1 in a directory called myDesigns:

```
create_project Project1 myDesigns
```

Note Because the *dir* is specified as the folder name only, the tool will create the project in the current working directory, or the directory from which the tool was launched.

The following example creates a project called Proj1 in a directory called FPGA in C:/Designs. In addition, the tool will overwrite an existing project if one is found to exist in the specified location. In the second and third lines, the location of **-force** is changed to show the flexibility of argument placement.

```
create_project Proj1 C:/Designs/FPGA -force
-or-
create_project Proj1 -force C:/Designs/FPGA
-or-
create_project -force Proj1 C:/Designs/FPGA
```

Note In all cases the first argument without a preceding keyword is interpreted as the *name* variable, and the second argument without a preceding keyword is the *dir* variable.

The following example creates a new project called *pin_project*, and then sets the **design_mode** property as required for an I/O Pin Planning project, and finally opens an IO design:

```
create_project pin_project C:/Designs/PinPlanning
set_property design_mode PinPlanning [current_fileset]
open_io_design -name io_1
```

See Also

- [current_project](#)
- [set_property](#)
- [open_io_design](#)

create_property

Create property for class of objects(s).

Syntax

```
create_property [-type arg] [-quiet] [-verbose] name class
```

Returns

The property that was created if success, "" if failure

Usage

Name	Description
[-type]	Type of property to create; valid values are: string, int, long, double, bool Default: string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of property to create
<i>class</i>	Object type to create property for; valid values are: design, net, cell, pin, port, pblock

Categories

[PropertyAndParameter](#)

Description

Creates a new property of the *type* specified with the user-defined *name* for the specified *class* of objects. The property that is created can be assigned to an object of the specified class with the **set_property** command, but is not automatically associated with all objects of that class.

The **report_property -all** command will not report the newly created property for an object of the specified class until the property has been assigned to that object.

Arguments

-type *arg* - (Optional) The type of property to create. There are four allowed property types:

- **string** - Allows the new property to be defined with string values. This is the default value when **-type** is not specified.
- **int** - Allows the new property to be defined with long integer values. If a decimal value is specified for an **int** property type, the tool will return an error.
- **double** - Allows the new property value to be defined with a floating point number.
- **bool** - Allows the new property to be defined as a boolean with a true (1) or false (0) value.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the property to be defined. The name is case sensitive.

class - (Required) The class of object to assign the new property to. All objects of the specified class will be assigned the newly defined property. Valid classes are: design, net, cell, pin, port, and pblock.

Examples

The following example defines a property called PURPOSE for cell objects:

```
create_property PURPOSE cell
```

Note Because the **-type** was not specified, the value will default to strings.

The following example creates a pin property called COUNT which holds an Integer value:

```
create_property -type int COUNT pin
```

See Also

- [get_property](#)
- [list_property](#)
- [list_property_value](#)
- [report_property](#)
- [reset_property](#)
- [set_property](#)

create_run

Define a synthesis or implementation run for the current project.

Syntax

```
create_run [-constrset arg] [-parent_run arg] [-part arg] -flow arg
[-strategy arg] [-quiet] [-verbose] name
```

Returns

Run object

Usage

Name	Description
[-constrset]	Constraint fileset to use
[-parent_run]	Synthesis run to link to new implementation run
[-part]	Target part
-flow	Flow name
[-strategy]	Strategy to apply to the run
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name for new run

Categories

[Project](#)

Description

Defines a synthesis or implementation run. The attributes of the run can be configured with the use of the **set_property** command.

Arguments

-constrset *arg* - (Optional) The constraint set to use for the synthesis or implementation run.

-parent_run *arg* - The synthesis run which the implementation run will implement. For an RTL sources project, the parent_run must be specified for implementation runs, but is not required for synthesis runs. For netlist-based projects the parent_run argument is not required to define an implementation run.

-part *partName* - (Optional) The Xilinx part to be used for the run. If the **-part** option is not specified, the default part defined for the project will be assigned as the part to use.

-flow *arg* - (Required) The tool flow and release version for the synthesis tool {Vivado Synthesis 2012} or the implementation tool {Vivado Implementation 2012}.

-strategy *arg* - (Optional) The strategy to employ for the synthesis or implementation run. There are many different strategies to choose from within the tool, including custom strategies you can define. Refer to the appropriate user guide for a discussion of the available synthesis and implementation strategies. If the strategy argument is not specified, "Synthesis Defaults" or "Implementation Defaults" will be used as appropriate.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the synthesis or implementation run to be created.

Examples

The following example creates a run named synth_1 referencing the Vivado synthesis tool flow:

```
create_run -flow {Vivado Synthesis 2013} synth_1
```

Note The defaults of sources_1, constrs_1, and the default part for the project will be used in the synthesis run. In addition, since this is a synthesis run, the **-parent_run** argument is not required.

The following example creates an implementation run based on the Vivado Implementation 2013 tool flow, and attaches it to the synth_1 synthesis run previously created:

```
create_run impl_2 -parent_run synth_1 -flow {Vivado Implementation 2013}
```

Note The **-parent_run** argument is required in this example because it is an implementation of synthesized RTL sources.

See Also

- [current_run](#)
- [launch_runs](#)
- [set_property](#)

create_slack_histogram

Create Histogram.

Syntax

```
create_slack_histogram [-to args] [-delay_type arg] [-num_bins arg]
[-slack_less_than arg] [-slack_greater_than arg] [-group args]
[-report_unconstrained] [-significant_digits arg] [-scale arg]
[-name arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-to]	To clock
[-delay_type]	Type of path delay: Values: max, min, min_max Default: max
[-num_bins]	Maximum number of bins: Value >=1 Default: 10
[-slack_less_than]	Display paths with slack less than this Default: 1e+30
[-slack_greater_than]	Display paths with slack greater than this Default: -1e+30
[-group]	Limit report to paths in this group(s)
[-report_unconstrained]	Report unconstrained end points
[-significant_digits]	Number of digits to display: Range: 0 to 3 Default: 3
[-scale]	Type of scale on which to draw the histogram; Values: linear, logarithmic Default: linear
[-name]	Output the results to GUI panel with this name
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Create a slack histogram grouping paths into slack ranges, and displaying the results graphically.

This command provides a graphical slack histogram that requires the tool to be running in GUI mode the **-name** argument to be used.

Arguments

-to *args* - (Optional) Specify a clock name, to analyze paths that end in the specified clock domain.

-delay_type *arg* - (Optional) Specifies the type of path delay to analyze when creating the slack report. The valid values are min, max, and min_max. The default setting for **-delay_type** is max.

-num_bins *args* - (Optional) Specify the number of slack bins to divide the results into. The number of bins determines the granularity of the histogram returned. The range of slack values calculated is divided evenly into the specified number of bins, and the paths are grouped into the bins according to their slack values. The value can be specified as a number => 1, with a default value of 10.

-slack_less_than *arg* - (Optional) Report slack on paths with a calculated slack value less than the specified value. Used with **-slack_greater_than** to provide a range of slack values of specific interest.

-slack_greater_than *arg* - (Optional) Report slack on paths with a calculated slack value greater than the specified value. Used with **-slack_less_than** to provide a range of slack values of specific interest.

-group *args* - (Optional) Report slack for paths in the specified path groups. Currently defined path groups can be determined with the **get_path_groups** command.

-report_unconstrained - (Optional) Report delay slack on unconstrained paths. By default, unconstrained paths are not analyzed.

-significant_digits *arg* - (Optional) The number of significant digits in the output results. The valid range is 0 to 3. The default setting is 3 significant digits.

-scale [linear | logarithmic] - (Optional) Specify the Y-axis scale to use when presenting the slack histogram. Logarithmic allows for a smoother presentation of greatly different values, but linear is the default.

-name *arg* - (Optional) Specifies the name of the results set for the GUI. If the name specified is currently opened, the **create_slack_histogram** will overwrite the current results.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example creates a slack histogram of the current design, using the default values, and outputting the results to the named result set in the GUI:

```
create_slack_histogram -name slack1
```

See Also

- [delete_timing_results](#)
- [get_path_groups](#)
- [report_timing](#)

create_sysgen

Create DSP source for Xilinx System Generator and add to the source files.

Syntax

```
create_sysgen [-quiet] [-verbose] name
```

Returns

Name for the new sub module

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>name</code>	Sub module name

Categories

[SysGen](#)

Description

Create a DSP sub-module for use in the current project, and add it to the source files.

This command will launch System Generator for DSP to let you design the hardware portion of the embedded processor system. System Generator is a DSP design tool from Xilinx that allows the RTL source files, Simulink and MATLAB software models, and C/C++ components of a DSP system to come together in a single simulation and implementation environment.

For more information on using specific features of the tool refer to *System Generator for DSP Getting Started Guide* (UG639).

You can also add existing DSP model files (.mdl) from System Generator into the current project using the **add_files** command.

The command returns the name of the DSP module created and added to the project.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the DSP module to create and add to the current project.

Examples

The following example launches System Generator and allows you to define and configure the specified DSP module:

```
create_sysgen DSP_mod1
```

See Also

- [add_files](#)
- [generate_target](#)
- [list_targets](#)
- [make_wrapper](#)

create_wave_config

Creates a new wave config.

Syntax

```
create_wave_config [-quiet] [-verbose] [name]
```

Returns

The new wave config

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[name]</i>	Creates a new wave configuration of the specified name, or a default name if no name given. A new wave window showing that WCFG is also created and made the current wave window

Categories

[Waveform](#)

create_xps

Create embedded source for XPS and add to the source fileset.

Syntax

```
create_xps [-quiet] [-verbose] name
```

Returns

Source file name that was created

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Source name

Categories

XPS

Description

Create an Embedded Processor source for use in the current project, and add it to the source files.

This command will launch the Xilinx Platform Studio (XPS) to let you design the hardware portion of the embedded processor system. In XPS you can define and configure the microprocessor, peripherals, and the interconnection of these components. After you exit XPS, the created files for the Embedded Processor sub-design will be written to the local project directory (*project_name*.srcs/sources_1/edk/*name*), and added to the source files.

For more information on using specific features of XPS refer to *EDK Concepts, Tools, and Techniques* (UG683).

You can also add existing Xilinx Microprocessor Project (.xmp) files from XPS in the current project using the **add_files** command.

The command returns the name of the Embedded Processor sub-design created.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the Embedded Processor sub-design to create and add to the current project.

Examples

The following example launches XPS to define and configure the specified Embedded Processor sub-design:

```
create_xps xpsTest1
```

See Also

- [add_files](#)
- [generate_target](#)
- [list_targets](#)
- [make_wrapper](#)

current_bd_design

Set or get current design.

Syntax

```
current_bd_design [-quiet] [-verbose] [design]
```

Returns

The current design object, "" if failed

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>design</i>]	Name of current design to be set

Categories

[IPIntegrator](#)

Description

Defines the current IP subsystem design for use with the IP Integrator feature of the Vivado Design Suite, or returns the name of the current design in the active project.

The current IP subsystem design and current IP subsystem instance are the target of most of the IP integrator Tcl commands and design changes made in the tool. The current IP subsystem instance can be defined using the **current_bd_instance** command.

You can use the **get_bd_designs** command to get a list of open IP subsystem designs in the active project.

A complete list of IP integrator Tcl commands can be returned using the following command from the Vivado Design Suite Tcl shell:

```
load_features IPIntegrator
help -category IPIntegrator
```

Note The **load_features** command is only needed if the IP Integrator feature is not currently loaded in the Vivado Design Suite.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

design - (Optional) The name of an IP subsystem design to set as the current design in the IP Integrator. If a *design* is not specified, the command returns the current IP subsystem design of the active project.

Examples

The following example sets the IP subsystem design as the current design:

```
current_design design_1
```

See Also

- [get_bd_designs](#)
- [open_bd_design](#)

current_bd_instance

Set or get current cell instance.

Syntax

```
current_bd_instance [-quiet] [-verbose] [instance]
```

Returns

The current cell instance object, "" if failed

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[instance]</i>	Name of current cell instance to be set

Categories

[IPIntegrator](#)

current_board

Get the current board object.

Syntax

```
current_board [-quiet] [-verbose]
```

Returns

Current board object

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Object](#), [Project](#), [Board](#)

Description

Return the board in use in the current project.

The board file, `board.xml` located in the `data/boards` folder of the Vivado Design Suite installation area, stores information regarding board attributes. The board provides a representation of the overall system that the Xilinx device is a part of, and can help define key aspects of the FPGA design, such as clock constraints, I/O port assignments, and supported interfaces.

The board is specified as part of a Targeted Reference Design, when the project is defined; or by setting the `BOARD` property on the current project as shown in the example; or by selecting the Project Device in the Project Settings dialog box in the Vivado IDE. Refer to the *Vivado Design Suite User Guide: Using the Vivado IDE (UG893)* for more information on project settings.

Important! When you specify the board with the **set_property** command, the target part is also changed to match the part required by the specified `BOARD` property.

The **current_board** command returns the **Vendor:Library:Name:Version** attributes of the current board. The command returns nothing when the project targets a Xilinx FPGA device instead of a TRD and board, or when the `BOARD` property has not been defined.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Example

The following example sets the BOARD property for the current project, then reports the board in use by the project:

```
set_property board xilinx.com:kintex7:kc705:1.0 [current_project]
current_board
    xilinx.com:kintex7:kc705:1.0
```

This example shows the results of setting the BOARD property, causing the target part to be changed as a result. The target part is changed automatically, and a warning is returned:

```
set_property board xilinx.com:artix7:ac701:1.0 [current_project]
WARNING: [Project 1-153] The current project part 'xc7k325tffg900-2' does
not match with the 'XILINX.COM:ARTIX7:AC701:1.0' board part settings. The
project part will be reset to 'XILINX.COM:ARTIX7:AC701:1.0' board part.
INFO: [Project 1-152] Project part set to artix7 (xc7a200tfbg676-2)
```

Note You can use the **report_property** command to check the BOARD and PART property on the **current_project** to see the changes

See Also

- [current_project](#)
- [get_board_interfaces](#)
- [get_board_pins](#)
- [get_boards](#)
- [report_property](#)
- [set_property](#)

current_design

Set or get the current design.

Syntax

```
current_design [-quiet] [-verbose] [design]
```

Returns

Design object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>design</i>]	Name of current design to be set

Categories

[SDC](#), [XDC](#)

Description

Defines the current design or returns the name of the current design in the active project.

The current design and current instance are the target of most Tcl commands, design edits and constraint changes made in the tool. The current instance can be defined using the **current_instance** command.

You can use the **get_designs** command to get a list of open designs in the active project, and use the **get_projects** command to get a list of open projects.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

design - (Optional) The name of design to set as the current design. If a *design* is not specified, the command returns the current design of the active project.

Examples

The following example sets the design rtl_1 as the current design:

```
current_design rtl_1
```

See Also

- [current_instance](#)
- [get_designs](#)
- [get_projects](#)

current_fileset

Get the current fileset (any type) or set the current fileset (applicable to simulation filesets only).

Syntax

```
current_fileset [-constrset] [-simset] [-quiet] [-verbose] [fileset...]
```

Returns

Current fileset (the current srcset by default)

Usage

Name	Description
[-constrset]	Get the current constraints fileset
[-simset]	Get the current active simulation fileset
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[fileset]</i>	Specify the simulation fileset to set as current (active); optional

Categories

[Project](#)

Description

Get the active source, constraint, or simulation fileset within the current project.

When used without any options, `current_fileset` sets and returns the `sources_1` set as the active fileset.

This command can also be used to set the current simulation fileset.

Note Use **set_property CONSTRSET** to define the active constraint set on a synthesis or implementation run

Arguments

-constrset - (Optional) Return the currently active constraint set.

-simset - (Optional) Return or set the currently active simulation fileset.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

fileset - (Optional) The name of the simulation fileset to make active. This argument sets the active simulation fileset in projects with multiple filesets. When *fileset* is not specified, the *sources_1* fileset is returned as the active fileset.

Examples

The following example returns the name of the currently active constraint fileset:

```
current_fileset -constrset
```

The following example sets *sim_2* as the active simulation set:

```
current_fileset -simset sim_2
```

See Also

- [create_fileset](#)
- [delete_fileset](#)
- [get_filesets](#)

current_hw_device

Get or set the current hardware device.

Syntax

```
current_hw_device [-quiet] [-verbose] [hw_device]
```

Returns

Hardware device

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_device]	hardware device to set as current; optional

Categories

[Hardware](#)

current_hw_ila

Get or set the current hardware ILA.

Syntax

```
current_hw_ila [-quiet] [-verbose] [hw_ila]
```

Returns

Hardware ILA

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[hw_ila]</code>	hardware ILA

Categories

[Hardware](#)

current_hw_ila_data

Get or set the current hardware ILA data.

Syntax

```
current_hw_ila_data [-quiet] [-verbose] [hw_ila_data]
```

Returns

Hardware ILA data

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_ila_data]	hardware ILA data

Categories

[Hardware](#)

current_hw_server

Get or set the current hardware server.

Syntax

```
current_hw_server [-quiet] [-verbose] [hw_server]
```

Returns

Hardware server

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_server]	hardware server

Categories

[Hardware](#)

current_hw_target

Get or set the current hardware target.

Syntax

```
current_hw_target [-quiet] [-verbose] [hw_target]
```

Returns

Hardware target

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_target]	hardware target

Categories

[Hardware](#)

current_instance

Set or get the current instance.

Syntax

```
current_instance [-quiet] [-verbose] [instance]
```

Returns

Instance name

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[instance]</i>	Name of instance

Categories

[SDC](#), [XDC](#)

Description

Set the current instance in the design hierarchy to the specified instance cell or to the top module. By default, **current_instance** points to the top module of the **current_design**, which is not an instantiated cell object. You can also set **current_instance** to reference an instantiated hierarchical cell in the design.

Since the top module is not an instantiated object, this command returns a string with the name of the current instance, rather than an object.

The current design and current instance are the target of most of the commands and design changes you will make. The current design can be defined using the **current_design** command.

You must specify the *instance* name relative to the currently defined instance, and use the established hierarchy separator to define instance paths. You can determine the current hierarchy separator with the **get_hierarchy_separator** command.

Use '..' to traverse up the hierarchical instance path when specifying the current instance.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

instance - (Optional) The name of the instance to be set as the current instance of the current design.

- The *instance* is specified relative to the presently defined current instance, using the defined hierarchy separator.
- Use `'..'` to move up one level of the hierarchy relative to the current instance.
- If the *instance* argument is omitted, the current instance is reset to the top module in the design hierarchy.
- If the *instance* is specified as `'.'` then the name of the current instance is returned, and the instance is not changed.

Examples

The following example sets the current instance to the top module of the current design:

```
current_instance
INFO: [Vivado 12-618] Current instance is the top level of design 'netlist_1'.
top
```

The following example first sets the hierarchy separator character, and then sets the current instance relative to the presently defined current instance:

```
set_hierarchy_separator |
current_instance ..|cpu_iwb_dat_o|buffer_fifo
```

The following example returns the name of the presently defined current instance:

```
current_instance .
cpuEngine|cpu_iwb_dat_o|buffer_fifo
```

See Also

- [current_design](#)
- [get_hierarchy_separator](#)
- [set_hierarchy_separator](#)

current_project

Set or get current project.

Syntax

```
current_project [-quiet] [-verbose] [project]
```

Returns

Current or newly set project object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[project]	Project to set as current

Categories

Project

Description

Specifies the current project or returns the current project when no project is specified.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

project - (Optional) The name of the project to make current. This command can be used prior to the **close_project** to make a specific project active and then to close the project.

Examples

The following example sets `project_2` as the current project:

```
current_project project_2
```

This command makes the current project the focus of all the tool commands. In the GUI mode, the current project is defined automatically when switching the GUI between projects.

The following example returns the name of the current project in the tool:

```
current_project
```

Note The returned value is the name of the project and not the name or path of the project file.

See Also

- [close_project](#)
- [current_design](#)

current_run

Set or get the current run.

Syntax

```
current_run [-synthesis] [-implementation] [-quiet] [-verbose] [run]
```

Returns

Run object

Usage

Name	Description
[-synthesis]	Set or get the current synthesis run
[-implementation]	Set or get the current implementation run (default unless '-synthesis' is specified)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[run]	Run to set as current; optional

Categories

[Project](#)

Description

Defines the current synthesis or implementation run, or returns the name of the current run. The current run is the one automatically selected when the Synthesize or Implement commands are launched.

You can use the **get_runs** command to determine the list of defined runs in the current design.

Arguments

-synthesis - (Optional) Specifies that the **current_run** command should set or return the name of the current synthesis run.

-implementation - (Optional) Specifies that the **current_run** command should set or return the name of the current implementation run. This is the default used when neither **-synthesis** or **-implementation** are specified.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

run - (Optional) Sets the name of the synthesis or implementation run to make the current run.

Examples

The following example defines the synth_1 run as the current_run:

```
current_run synth_1
```

Note The **-synthesis** and **-implementation** arguments are not required because the name allows the tool to identify the specific run of interest.

The following command returns the name of the current implementation run:

```
current_run -implementation -quiet
```

See Also

- [create_run](#)
- [get_runs](#)
- [launch_runs](#)

current_scope

Get the current scope or set the current scope.

Syntax

```
current_scope [-quiet] [-verbose] [hdl_scope]
```

Returns

The current scope

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[hdl_scope]</i>	Default: NULL

Categories

[Simulation](#)

current_sim

Set the current simulation object or get the current simulation object.

Syntax

```
current_sim [-quiet] [-verbose] [simulationObject]
```

Returns

Returns the current simulation object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[simulationObject]	Simulation Object to set the current simulation object to Default: NULL

Categories

[Simulation](#)

Description

Returns the name of the current Vivado simulation.

This command can be used after the Vivado simulator has been launched to return the simulation name.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the name of the current simulation:

```
current_sim
```

See Also

- [close_sim](#)
- [launch_xsim](#)

current_time

Report current simulation time.

Syntax

```
current_time [-quiet] [-verbose]
```

Returns

Prints the current simulation time on the console in textual format

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Simulation](#)

current_wave_config

Gets the current WCFG object and sets it to the specified WCFG object if given.

Syntax

```
current_wave_config [-quiet] [-verbose] [wcfgObj]
```

Returns

Returns the new or current wave configuration object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[wcfgObj]	Sets the current WCFG object to the given value of wcfgObj. Defaults to current

Categories

[Waveform](#)

data2mem

Data to memory translation.

Syntax

```
data2mem [-bd arg] [-bm arg] [-bt arg] [-bx arg] [-d arg] [-i]
[-o arg] [-ofmt arg] [-p arg] [-u] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-bd]	Input ELF or MEM files
[-bm]	Input Block RAM Memory Map (BMM) file
[-bt]	Input bitstream (BIT) file
[-bx]	File path to output individual memory device MEM files for performing HDL simulations.
[-d]	Dump mode Default: r
[-i]	Ignore ELF or MEM data that is outside the address space defined in BMM file
[-o]	Output file
[-ofmt]	Output format (.ucf/.v/.vhd/.bit/.dmp)
[-p]	Input target part
[-u]	Update -o text output files for all address spaces, even if no data has been transformed into an address space
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Simulation](#)

delete_bd_objs

Delete specified objects.

Syntax

```
delete_bd_objs [-quiet] [-verbose] objects...
```

Returns

Pass if successful in deleting objects

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	The objects to be deleted

Categories

[IPIntegrator](#)

delete_clock_networks_results

Clear a set of clock networks results from memory.

Syntax

```
delete_clock_networks_results [-quiet] [-verbose] name
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name for the set of results to clear

Categories

[Report](#)

delete_debug_core

Delete a debug core.

Syntax

```
delete_debug_core [-quiet] [-verbose] cores...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>cores</i>	Debug cores to delete

Categories

Description

Removes debug cores from the current project that were added by the **create_debug_core** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

cores - (Required) One or more debug core names to remove from the current project.

Examples

The following command deletes the myCore debug core from the current project:

```
delete_debug_core myCore
```

The following command deletes all debug cores from the current project:

```
delete_debug_core [get_debug_cores]
```

Note The **get_debug_cores** command returns all debug cores as a default.

See Also

- [create_debug_core](#)
- [get_debug_cores](#)

delete_debug_port

Delete debug port.

Syntax

```
delete_debug_port [-quiet] [-verbose] ports...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>ports</i>	Debug ports to delete

Categories

Description

Deletes ports from ChipScope debug cores in the current project. You can disconnect a signal from a debug port using **disconnect_debug_port**, or remove the port altogether using this command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

ports - (Required) The core_name/port_name of the debug port to be removed from the core.

Examples

The following example deletes the DATA port from myCore:

```
delete_debug_port myCore/DATA
```

Note Some ports cannot be deleted because an ILA port requires one CLK port and one TRIG port as a minimum.

The following example deletes the trigger ports (TRIG) from the myCore debug core:

```
delete_debug_port [get_debug_ports myCore/TRIG*]
```

Note This example will not delete all TRIG ports from myCore, because an ILA core must have at least one TRIG port. The effect of this command will be to delete the TRIG ports starting at TRIG0 and removing all of them except the last port.

See Also

- [disconnect_debug_port](#)
- [get_debug_ports](#)

delete_drc_check

Delete one or more user-defined drc checks.

Syntax

```
delete_drc_check [-quiet] [-verbose] name...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Specify the key for the check to remove. This is the typically of the form PREFIX-id where PREFIX is a 4-6 letter abbreviation and id is a unique identifier. Use <code>get_drc_checks</code> to determine the correct name to use. Only user-defined rules may be deleted.

Categories

DRC, Object

Description

Delete a single user-defined design rule checks from the current project. User-defined design rule checks are created using the **create_drc_checks** command.

Note You cannot delete factory defined rule checks.

Once it has been deleted there is no way to recover a rule check. The undo command will not work.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) Specify the name of a user-defined design rule check to be deleted from the current project.

Examples

The following example deletes the specified design rule check:

```
delete_drc_check L2H-1
```

See Also

[create_drc_check](#)

delete_drc_ruledeck

Delete one or more user defined drc rule deck objects.

Syntax

```
delete_drc_ruledeck [-regexp] [-nocase] [-filter arg] [-quiet]  
[-verbose] [patterns]
```

Returns

Drc_ruledeck

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'drc_ruledeck' objects against patterns. Default: *

Categories

[DRC](#), [Object](#)

Description

Delete one or more user-defined drc_ruledeck objects from the current project. The rule deck does not have to be empty to be deleted, and once it is deleted there is no way to recover it. The undo command will not restore a deleted rule deck.

Note You cannot delete factory defined rule decks.

A rule deck is a collection of design rule checks grouped for convenience, to be run with the **report_drc** command at different stages of the FPGA design flow, such as during I/O planning or placement. The tool comes with a set of factory defined rule decks, but you can also create new user-defined rule decks with the **create_drc_ruledeck** command.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by the search pattern, based on specified property values. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Delete the drc_ruledesk objects that match the specified patterns. The default pattern is the wildcard '*' which deletes all user-defined rule decks from the current project. More than one pattern can be specified to delete multiple rule decks based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example deletes all user-defined rule decks from the current project:

```
delete_drc_ruledesk
```

See Also

- [create_drc_ruledesk](#)
- [list_property](#)
- [report_property](#)

delete_fileset

Delete a fileset.

Syntax

```
delete_fileset [-merge arg] [-quiet] [-verbose] fileset
```

Returns

Nothing

Usage

Name	Description
[-merge]	Fileset to merge files from the deleted fileset into
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>fileset</i>	Fileset to be deleted

Categories

[Project](#), [Simulation](#)

Description

Deletes the specified fileset. However, if the fileset cannot be deleted, then no message is returned.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

fileset - (Required) The name of the fileset to delete. The last constraint or simulation fileset will not be deleted, and no error will be returned under these circumstances.

Examples

The following example deletes the sim_2 fileset from the current project.

```
delete_filesset sim_2
```

Note The fileset and all of its files are removed from the project. The files are not removed from the hard drive.

See Also

- [create_filesset](#)
- [current_filesset](#)

delete_interface

Delete I/O port interfaces from the project.

Syntax

```
delete_interface [-all] [-quiet] [-verbose] interfaces...
```

Returns

Nothing

Usage

Name	Description
[-all]	Also remove all of the ports and buses belonging to the interface
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>interfaces</i>	I/O port interfaces to remove

Categories

[PinPlanning](#)

Description

Deletes an existing interface and optionally deletes all of the associated ports and buses using the interface.

Arguments

-all - (Optional) Delete all ports, buses, or nested interfaces associated with the specified interface.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

interfaces - (Required) The name of interfaces to delete.

Examples

The following example deletes the specified interface and all of its associated ports and buses:

```
delete_interface USB0
```

See Also

- [create_interface](#)
- [create_port](#)

delete_macros

Delete a list of macros.

Syntax

```
delete_macros [-quiet] [-verbose] macros
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>macros</i>	Macros to delete

Categories

XDC

Description

Create a new relatively placed macro, similar to that created by the RPM property.

New cell instances can be added to the top-level of the design, or hierarchically within any module of the design. Instances can reference an existing cell from the library or design source files, or a black box instance can be added that reference cells that have not yet been created.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

This command returns the name of the created cell instance or instances.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) Specify the name of the macro to create.

Examples

The following example creates a macro called :

```
create_macro
```

See Also

- [remove_cell](#)
- [set_hierarchy_separator](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

delete_pblock

Remove Pblock.

Syntax

```
delete_pblock [-hier] [-quiet] [-verbose] pblocks...
```

Returns

Nothing

Usage

Name	Description
[-hier]	Also delete all the children of Pblock
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>pblocks</i>	Pblocks to delete

Categories

Floorplan, XDC

Description

Deletes the specified Pblocks from the design. Pblocks are created using the **create_pblock** command.

Arguments

-hier - (Optional) Specifies that Pblocks nested inside the specified Pblock should also be deleted. If the parent Pblock is deleted without the **-hier** option specified, the nested Pblocks will simply be moved up one level.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pblocks - (Required) One or more Pblocks to be deleted.

Examples

The following example deletes the specified Pblock as well as any Pblocks nested inside:

```
delete_pblock -hier cpuEngine
```

See Also

[create_pblock](#)

delete_power_results

Delete power results that were stored in memory under a given name.

Syntax

```
delete_power_results -name arg [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
-name	Name for the set of results to clear
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Power

Description

Deletes the power analysis results for the specified results set.

Note This command operates silently and does not return direct feedback of its operation

Arguments

-name *arg* - (Required) The name of the results set to delete. This name was either explicitly defined, or was automatically defined when the **report_power** command was run.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example runs power analysis, and then clears the results:

```
report_power -name my_set  
delete_power_results -name my_set
```

See Also

- [power_opt_design](#)
- [report_power](#)
- [reset_switching_activity](#)
- [set_switching_activity](#)

delete_rpm

Delete an RPM.

Syntax

```
delete_rpm [-quiet] [-verbose] rpm
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>rpm</i>	RPM to delete

Categories

[Floorplan](#)

Description

Deletes the specified Relationally Placed Macro (RPM) from the design.

An RPM is a list of logic elements (FFS, LUT, CY4, RAM, etc.) collected into a set (U_SET, H_SET, and HU_SET). The placement of each element within the set, relative to other elements of the set, is controlled by Relative Location Constraints (RLOCs). Logic elements with RLOC constraints and common set names are associated in an RPM. Refer to the Constraints Guide (UG625) for more information on defining these constraints.

Only user-defined RPMs can be deleted from the design. RPMs defined by the hierarchy or defined in the netlist cannot be deleted by this command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

rpm - (Required) The RPM to be deleted.

Examples

The following example deletes the specified RPM (cs_ila_0/U0) from the design:

```
delete_rpm cs_ila_0/U0
```

delete_run

Delete an existing run.

Syntax

```
delete_run [-noclean_dir] [-quiet] [-verbose] run
```

Returns

Nothing

Usage

Name	Description
<code>[-noclean_dir]</code>	Do not remove all output files and directories from disk
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>run</code>	Run to modify

Categories

Project

Description

Deletes the specified run from the project, and deletes all results of the run from the project directory on the hard drive unless otherwise specified.

Arguments

-noclean_dir - Do not delete the run results from the hard drive. The run will be deleted from the project, but the run files will remain in the project directory.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

run - (Required) The name of the synthesis or implementation run to delete from the project.

Examples

The following example deletes the `first_pass` run from the project:

```
delete_run first_pass
```

Note In this example, all run results will also be removed from the project directory on the hard drive.

The following command deletes the `first_pass` run, but leaves the run results on the hard drive:

```
delete_run -noclean_dir first_pass
```

See Also

- [create_run](#)
- [current_run](#)

delete_timing_results

Clear a set of timing results from memory.

Syntax

```
delete_timing_results [-type arg] [-quiet] [-verbose] name
```

Returns

Nothing

Usage

Name	Description
[-type]	Type of timing results to clear; Values: timing_path, slack_histogram, clock_interaction, check_timing, pulse_width, timing_summary
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name for the set of results to clear

Categories

[Report](#), [Timing](#)

Description

Clear the specified timing results from the named result set. Both the type of the timing report, and the name of the timing report must be specified, or the command will fail.

Arguments

-type *arg* - (Optional) Specifies the type of timing results to be cleared. The available types are:

- timing_path - Delete the named **report_timing** report.
- slack_histogram - Delete the named **create_slack_histogram** report.
- clock_interaction - Delete the named **report_clock_interaction** report.
- check_timing - Delete the named **check_timing** report.
- pulse_width - Delete the named **report_pulse_width** report.
- timing_summary - Delete the named **report_timing_summary** report.

Note The default **-type** is timing_path, to delete reports generated by the **report_timing** command.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

-name *arg* - (Required) Specifies the name of the timing results to be cleared.

Examples

The following example clears the specified results set from memory:

```
delete_timing_results -type clock_interaction -name clkNets
```

See Also

- [check_timing](#)
- [create_slack_histogram](#)
- [report_clock_interaction](#)
- [report_pulse_width](#)
- [report_timing](#)
- [report_timing_summary](#)

delete_utilization_results

Delete utilization results that were stored in memory under a given name.

Syntax

```
delete_utilization_results -name arg [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
-name	Name for the set of results to clear
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Clear the specified utilization results from the named result set.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

-name *arg* - (Required) Specifies the name of the results to be cleared.

Examples

The following example clears the specified results set from memory:

```
delete_utilization_results -name SS01
```

See Also

[report_utilization](#)

describe

Describe an HDL object (variable, signal, wire, or reg) by printing type and declaration information.

Syntax

```
describe [-quiet] [-verbose] hdl_object
```

Returns

The description of the selected objects

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hdl_object</i>	The hdl_object or hdl_scope to describe

Categories

[Simulation](#)

disconnect_bd_intf_net

Disconnect an intf_net.

Syntax

```
disconnect_bd_intf_net [-quiet] [-verbose] intf_net objects...
```

Returns

0 if successful, error otherwise

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>intf_net</i>	The IntfNet that the objects connect to
<i>objects</i>	The objects to disconnect from the intf_net

Categories

[IPIntegrator](#)

disconnect_bd_net

Disconnect a net from the object.

Syntax

```
disconnect_bd_net [-quiet] [-verbose] net objects...
```

Returns

0 if successful, error otherwise

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>net</i>	The Net that the objects connect to
<i>objects</i>	The objects to disconnect from the net

Categories

[IPIntegrator](#)

disconnect_debug_port

Disconnect nets and pins from debug port channels.

Syntax

```
disconnect_debug_port [-channel_index arg] [-quiet] [-verbose] port
```

Returns

Nothing

Usage

Name	Description
<code>[-channel_index]</code>	Disconnect the net at channel index
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>port</code>	Debug port name

Categories

Description

Disconnect signals from the debug ports.

Signals from the Netlist Design are connected to ports of a ChipScope debug core using the **connect_debug_port** command.

A port can also be deleted from the debug core rather than simply disconnected by using the **delete_debug_port** command.

If you need to determine the specific name of a port on a debug core, use the **get_debug_ports** command to list all ports on a core. You can also use the **report_debug_core** command to list all of the cores in the projects, and their specific parameters.

Arguments

-channel_index *value* - (Optional) The channel index of the port to disconnect.

Note The entire port is disconnected if **channel_index** is not specified.

port - (Required) The name of the port on the debug core to disconnect. The port name must be specified as *core_name/port_name*. See the examples below.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example disconnects only the specified channel index from the TRIG0 port of myCore:

```
disconnect_debug_port -channel_index 2 myCore/TRIG0
```

If you do not specify the channel_index, all of the channels of the specified port will be disconnected, as in the following example:

```
disconnect_debug_port myCore/TRIG0
```

See Also

- [connect_debug_port](#)
- [delete_debug_port](#)
- [get_debug_ports](#)
- [report_debug_core](#)

disconnect_hw_server

Close a connection to a hardware server.

Syntax

```
disconnect_hw_server [-quiet] [-verbose] [hw_server]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_server]	hardware server Default: current hardware server

Categories

[Hardware](#)

disconnect_net

Disconnect a net from pins or ports.

Syntax

```
disconnect_net [-prune] -net arg -objects args [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-prune]	When performing hierarchical disconnect (-hier), remove pins and ports which are left unconnected as a result of the disconnect_net operation.
-net	Net to disconnect
-objects	List of pins or ports to disconnect
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Netlist](#)

Description

This command allows the user to disconnect a specified net from one or more pins or ports in the netlist of an open Synthesized or Implemented Design.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

Arguments

-prune - (Optional) Remove pins or ports that are left unconnected after disconnecting the specified nets.

-net *arg* - (Required) Specifies the net to disconnect.

Note Although you can create a bus using the **-bus_from** and **-bus_to** arguments of the **create_net** command, you must disconnect each bit of the bus separately using the **disconnect_net** command

-objects *args* - (Required) The list of pin or port objects to disconnect the net from.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Example

The following example disconnects the specified bit of the dataBus:

```
disconnect_net -net dataBus[1] -objects {dataIN[1] myDMA/data[1]}
```

See Also

- [connect_net](#)
- [remove_net](#)
- [create_net](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

display_hw_ila_data

Display hardware ILA data in viewer.

Syntax

```
display_hw_ila_data [-wcfg arg] [-reset] [-quiet] [-verbose]  
[hw_ila_data...]
```

Returns

Nothing

Usage

Name	Description
[-wcfg]	Use alternate wave config file
[-reset]	Reset wave config file to default configuration
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[hw_ila_data]</i>	List of hardware ILA data objects. Default: Current hardware ILA data

Categories

[Hardware](#)

display_hw_sio_scan

Display an existing hardware SIO scan.

Syntax

```
display_hw_sio_scan [-quiet] [-verbose] hw_sio_scans
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_sio_scans</i>	hardware SIO scans

Categories

[Hardware](#)

endgroup

End a set of commands that can be undone/redone as a group.

Syntax

```
endgroup [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[GUIControl](#)

Description

Ends a sequence of commands that can be undone or redone as a series. Use **startgroup** to start the sequence of commands.

Note You can have multiple command groups to **undo** or **redo**, but you cannot nest command groups. You must use **endgroup** to end a command sequence before using **startgroup** to create a new command sequence

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example defines a startgroup, executes a sequence of related commands, and then executes the endgroup. This sequence of commands can be undone as a group:

```
startgroup
create_pblock pblock_wbArbEngine
create_pblock pblock_usbEng
add_cells_to_pblock pblock_wbArbEngine [get_cells [list wbArbEngine]] -clear_locs
add_cells_to_pblock pblock_usbEng [get_cells [list usbEngine1/usbEngineSRAM]] -clear_locs
endgroup
```

See Also

- [startgroup](#)
- [redo](#)
- [undo](#)

export_hardware

Export system hardware platform for SDK.

Syntax

```
export_hardware [-bitstream] [-dir arg] [-quiet] [-verbose] files
[run]
```

Returns

Nothing

Usage

Name	Description
[-bitstream]	Export bitstream data to SDK export directory
[-dir]	Export directory
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>files</i>	Source files for which the hardware data needs to be exported
<i>[run]</i>	Current implementation run

Categories

XPS

Description

Export hardware for use in software development.

Export the Embedded Processor system hardware platform for use by SDK to support the design of software for the embedded processor sources in your project. Specify the embedded processor XPS project files to export to SDK.

As a default, the tool will write the hardware specification file (.xml) for the specified embedded processors to the *project_name.sdk/SDK/SDK_Export/hw* directory, to a file named after the embedded processor in the design, with the .XML extension.

This is a copy of the *system.xml* file in the embedded processor directory of the project design sources located at: *project_name.srcs/sources_1/edk/robot/__xps/*. The *system.xml* file contains a description of the embedded processor, and the components of the XPS design.

The output of the **export_hardware** command can be redirected to a user-defined directory with the **-dir** option. The output of the command can be used to invoke SDK with the **launch_sdk** command.

The command returns a transcript of the export process.

```
export_hardware -bitstream [get_files *.xmp] # Exports hardware design for the XMP source
export_hardware -bitstream [get_files *.bd] # Exports hardware design for the BD source
```

Arguments

-bitstream - (Optional) Export the bitstream and BMM model data for the Embedded Processor, in addition to the hardware specification file.

Note The tool will return an error if the bitstream file does not exist

-dir arg - (Optional) Export directory. By default the hardware files will be written to the local project directory, under *project_name.sdk/SDK/SDK_Export/hw*.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Required) A files object that contains the list of XPS project files (.xmp) to export.

Note Use **get_files** to specify a files object, rather than specifying a file name.

run - (Optional) Specify an implementation run to export.

Examples

The following example exports the Embedded Processor design to the standard SDK export directory, and includes the Bitstream and BMM model data:

```
export_hardware -bitstream [get_files *.xmp]
```

See Also

- [create_xps](#)
- [generate_target](#)
- [get_files](#)
- [launch_sdk](#)

filter

Filter a list, resulting in new list.

Syntax

```
filter [-regexp] [-nocase] [-quiet] [-verbose] [objects] [filter]
```

Returns

New list

Usage

Name	Description
<code>[-regexp]</code>	Operators =~ and !~ use regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when -regexp specified)
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[objects]</code>	List of objects to filter
<code>[filter]</code>	Filter list with expression

Categories

[Object](#), [PropertyAndParameter](#), [XDC](#)

Description

Takes a list of objects, and returns a reduced list of objects that match the specified filter search pattern.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) A list of objects that should be filtered to reduce the set to the desired results. The list of objects can be obtained by using one of the many **get_*** commands such as **get_parts**.

filter - (Optional) The expression to use for filtering. The specified pattern filters the list of objects returned based on property values on the objects. You can find out which properties are on an object with the **report_property** or **list_property** command. Any property/value pair can be used as a filter. In the case of the "part" object, "DEVICE", "FAMILY" and "SPEED" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

Examples

The following example returns a list of parts filtered for the specified speed grade:

```
filter [get_parts] {speed == -3}
```

The following example filters parts based according to speed grade -3 OR speed grade -2. All parts matching either speed grade will be returned.

```
filter [get_parts] {speed == -3 || speed == -2}
```

The following example uses regular expression and returns a list of VStatus ports in the design, with zero or more wildcards, and the numbers 0 to 9 appearing one or more times within square brackets:

```
filter -regexp [get_ports] {NAME =~ VStatus.*\[[0-9]+\]}
```

See Also

- [get_parts](#)
- [get_ports](#)

find_bd_objs

Find a list of pins, ports or interfaces with a given relationship to the given object.

Syntax

```
find_bd_objs -relation arg [-quiet] [-verbose] objects...
```

Returns

List of pins, ports or interface objects, "" if failed

Usage

Name	Description
-relation	Relation to the input objs: CONNECTED_TO, ADDRESSABLE_SLAVE, ADDRESSING_MASTER. CONNECTED_TO will find corresponding pins, ports or interfaces that are connected to the given source objects, across subsystem boundaries.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	One or more source object to start finding from

Categories

[IPIntegrator](#)

find_top

Find top module candidates in the supplied files, fileset, or active fileset. Returns a rank ordered list of candidates.

Syntax

```
find_top [-fileset arg] [-files args] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-fileset]	Fileset to parse to search for top candidates
[-files]	Files to parse to search for top candidates
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Project](#)

Description

Find the most likely candidates for the top module in the files defined in the current fileset, or in the specified fileset, or in the specified list of files.

The command returns an ordered list of modules that the tool identifies as the best candidates for the top-level of the design. You can use the **index** command, and choose index 0 to select the best candidate for the top module.

Arguments

-fileset *arg* - (Optional) Search the specified simulation or source fileset for top module candidates. The default is to search the current fileset of the current design.

-files *arg* - (Optional) Find the top module candidates in the specified list of files.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example chooses the best top module of the current design for synthesis:

```
synth_design -top [lindex [find_top] 0]
```

Note Since **find_top** returns multiple possible candidates, choosing index 0 chooses the best top candidate for synthesis.

The following example returns the best top module candidate from the specified list of files:

```
find_top -files [get_files -filter {NAME =~ *or1200*}]
```

The following example sets the results of **find_top** into the variable **\$topVar**, then uses that variable to define the top module in the current fileset using the **set_property** command:

```
set topVar [ find_top -files [get_files -filter {NAME =~ *usb*} ] ]
usb_top
set_property top $topVar [current_fileset]
```

See Also

- [set_property](#)
- [synth_design](#)

flush_vcd

Flush VCD simulation output to the VCD output file (equivalent of \$dumpflush verilog system task).

Syntax

```
flush_vcd [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Simulation](#)

Description

Flush HDL signal information currently in memory into the specified Value Change Dump (VCD) file.

VCD is an ASCII file containing header information, variable definitions, and the value change details of a set of HDL signals. The VCD file can be used to view simulation results in a VCD viewer, or to estimate the power consumption of the design.

Note You must run the **open_vcd** command to open a VCD file to write to before using the **flush_vcd** command

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example flushes the VCD buffer into the current VCD file:

```
flush_vcd
```

See Also

[open_vcd](#)

generate_target

Generate target data for the specified source.

Syntax

```
generate_target [-force] [-quiet] [-verbose] name objects
```

Returns

Nothing

Usage

Name	Description
[-force]	Force target data regeneration
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	List of targets to be generated, or 'all' to generate all supported targets
<i>objects</i>	The objects for which data needs to be generated

Categories

[Project](#), [XPS](#), [IPFlow](#), [IPIntegrator](#)

Description

This command generates target data for the specified source file for IP cores (.xci and .xco), DSP modules (.mdl), or Embedded Processor sub-designs (.xmp). The target data that is generated are the files necessary to support the IP core, DSP module, or Embedded Processor, through the FPGA design flow.

For IP Cores, Instantiation Template, Synthesis, and Simulation are the standard targets. However, each IP in the catalog may also support its own set of targets such as Testbench, Example, Miscellaneous, etc.

Legacy IP support only the instantiation_template and synthesis targets. Native IP, available in the Vivado tool, can support all the different types of targets, though a specific IP core may not offer all targets.

For DSP modules and Embedded Processor sub-designs, Synthesis, Simulation, and Implementation are the standard targets.

You can use the **list_targets** command to list the targets supported by a specific source file.

Arguments

-force - (Optional) Force target data regeneration, and overwrite any existing target data files. Without **-force**, the tool will not regenerate any target data that is up-to-date.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The names of the types of target data to create for the specified source. The specific targets supported by an IP core are listed in the SUPPORTED_TARGETS property on the object. You can query this property to see which targets a specific core supports. Standard values are:

- **all** - Generate all targets for the specified core.
- **instantiation_template** - Generate the Instantiation template used to add the RTL module definition for the IP core into the current design. The instantiation template can be copied into any desired level of the design hierarchy.
- **synthesis** - Synthesis targets deliver HDL files that are used during synthesis for native IP, or deliver a synthesized netlist file (NGC) generated by XST.
- **simulation** - Simulation targets deliver HDL files that are used in simulation.
- **implementation** - Implementation generates the necessary data for implementing the IP core, DSP module, or Embedded Processor in the current design.
- **example** - Some native IP cores support the ability to open example projects containing the core. You must first generate the example target data before opening the core using the **open_example_project** command.
- **testbench** - Used to deliver a test bench that can be used to simulate the IP.
- **miscellaneous** - Some IP use the miscellaneous target to deliver documentation or scripts used in working with the IP.

objects - (Required) The object to generate the target from. Supported objects can include IP core objects, or the IP source files (XCI or XCO), DSP modules (MDL) imported from System Generator, and Embedded Processors (XMP) imported from Xilinx Platform Studio (XPS).

Note Use **get_files** to specify a files object, rather than specifying a file name

Examples

The following example generates the implementation template for all of the IP cores in the current project, forcing regeneration of any targets which are up-to-date:

```
generate_target instantiation_template [get_ips] -force
```

The following example generates the data files needed to support synthesis and simulation for the specified IP core (XCI file):

```
generate_target {Synthesis Simulation} \
[get_files C:/data/Projects/test_ip/test_addr_ip/test_addr_ip.xci \
-of_objects [get_filesets sources_1]]
```

The following example queries the specified IP object to report the SUPPORTED_TARGETS property, and then generates the Example target data:

```
report_property -all [get_ips blk_mem*]  
generate_target {example} [get_ips blk_mem*]
```

See Also

- [add_files](#)
- [create_ip](#)
- [create_sysgen](#)
- [create_xps](#)
- [import_ip](#)
- [list_targets](#)
- [open_example_project](#)
- [read_ip](#)
- [report_property](#)
- [reset_target](#)

get_bd_addr_segs

Get a list of segments.

Syntax

```
get_bd_addr_segs [-regexp] [-hierarchical] [-filter arg]  
[-of_objects args] [-addressed] [-addressables] [-quiet] [-verbose]  
[patterns]
```

Returns

List of segment objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-hierarchical]	Hierarchical cells included
[-filter]	Filter list with expression
[-of_objects]	Get segments of these pins or nets
[-addressed]	Get addressed segments of given -of_objects
[-addressables]	Get addressable segments of given -of_objects
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>patterns</i>]	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bd_addr_spaces

Get a list of addr_spaces.

Syntax

```
get_bd_addr_spaces [-regexp] [-hierarchical] [-filter arg]  
[-of_objects args] [-quiet] [-verbose] [patterns]
```

Returns

List of addr_space objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-hierarchical]	Hierarchical cells included
[-filter]	Filter list with expression
[-of_objects]	Get addr_spaces of these pins or nets
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bd_cells

Get a list of cells.

Syntax

```
get_bd_cells [-regexp] [-hierarchical] [-filter arg] [-of_objects args]  
[-quiet] [-verbose] [patterns]
```

Returns

List of cell objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-hierarchical]	Hierarchical cells included
[-filter]	Filter list with expression
[-of_objects]	Get cells of these pins or nets
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bd_designs

Get a list of designs.

Syntax

```
get_bd_designs [-regexp] [-filter arg] [-quiet] [-verbose]  
[patterns...]
```

Returns

List of design objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

Description

Gets a list of IP subsystem designs open in the current project that match a specified search pattern. The default command gets a list of all open IP subsystem designs in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_designs** based on property values on the designs. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "IP subsystem design" object, "NAME", and "FILE_NAME" are two of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match designs against the specified patterns. The default pattern is the wildcard '*' which gets all IP subsystem designs. More than one pattern can be specified to find multiple designs based on different search criteria.

Examples

The following example gets all open IP subsystem designs in the current project:

```
get_bd_designs
```

See Also

- [create_bd_design](#)
- [current_bd_design](#)
- [open_bd_design](#)
- [report_property](#)

get_bd_intf_nets

Get a list of intf_nets.

Syntax

```
get_bd_intf_nets [-regexp] [-hierarchical] [-filter arg]  
[-of_objects args] [-quiet] [-verbose] [patterns]
```

Returns

List of pin objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-hierarchical]	Hierarchical cells included
[-filter]	Filter list with expression
[-of_objects]	Of_objects, only one FullPathName for now
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>patterns</i>]	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bd_intf_pins

Get a list of intf_pins.

Syntax

```
get_bd_intf_pins [-regexp] [-hierarchical] [-filter arg]  
[-of_objects args] [-quiet] [-verbose] [patterns]
```

Returns

List of pin objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-hierarchical]	Hierarchical cells included
[-filter]	Filter list with expression
[-of_objects]	Of_objects, only one FullPathName for now
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bd_intf_ports

Get a list of intf_ports.

Syntax

```
get_bd_intf_ports [-regexp] [-hierarchical] [-filter arg]  
[-of_objects args] [-quiet] [-verbose] [patterns]
```

Returns

List of port objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-hierarchical]	Hierarchical cells included
[-filter]	Filter list with expression
[-of_objects]	Of_objects, only one FullPathName for now
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bd_nets

Get a list of nets.

Syntax

```
get_bd_nets [-regexp] [-hierarchical] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

Returns

List of pin objects, "" if failed

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-hierarchical]</code>	Hierarchical cells included
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Of_objects, only one FullPathName for now
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bd_pins

Get a list of pins.

Syntax

```
get_bd_pins [-regexp] [-hierarchical] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

Returns

List of pin objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-hierarchical]	Hierarchical cells included
[-filter]	Filter list with expression
[-of_objects]	Of_objects, only one FullPathName for now
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bd_ports

Get a list of ports.

Syntax

```
get_bd_ports [-regexp] [-filter arg] [-of_objects args] [-quiet]  
[-verbose] [patterns]
```

Returns

List of port objects, "" if failed

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Of_objects, only one FullPathName for now
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>patterns</i>]	Match engine names against patterns Default: *

Categories

[IPIntegrator](#)

get_bel_pins

Get a list of bel_pins. If a design is loaded, gets the constructed site type bels.

Syntax

```
get_bel_pins [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

Returns

Bel_pindef

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-of_objects]	Get the bel_pindef of these bels, sites, pins, or nets.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match bel_pindef against patterns Default: *

Categories

[Object](#)

Description

Returns a list of pins on the specified BELs, or matching a specified search pattern.

The default command gets a list of all pins on all BELs on the device.

Note To improve memory and performance, the get_* commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_bel_pins** based on property values on the pins. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the PIN object, "NAME" and "IS_INVERTED" are two of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) This option can be used with the **get_bels** command to return the pins of specified BELs.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match BEL pins against the specified patterns. The default pattern is the wildcard '*' which gets a list of all BEL pins on the device. More than one search pattern can be specified to find pins based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the pins of the specified BELs associated with the specified range of sites on the device:

```
get_bel_pins -of_objects [get_bels -of_objects [get_sites \
    -range {SLICE_X0Y0 SLICE_X1Y1}} ]
```

The following example returns the clock enable (CE) pins of all BELs on the device:

```
get_bel_pins *CE
```

See Also

- [get_bels](#)
- [get_sites](#)
- [list_property](#)
- [report_property](#)

get_bels

Get a list of bels. If a design is loaded, gets the constructed site type bels.

Syntax

```
get_bels [-regexp] [-nocase] [-filter arg] [-of_objects args] [-quiet]
[-verbose] [patterns]
```

Returns

Bels

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching. (valid only when -regexp specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get the bels of these sites cells clock_regions nets.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match bels against patterns Default: *

Categories

[Object](#)

Description

Basic Elements, or BELs, are building blocks of logic, such as flip-flops, LUTs, and carry logic, that make up a SLICE. This command returns a list of BELs on the target part that match a specified search pattern in an open design.

The default command gets a list of all BELs on the device.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_bels** based on property values on the BELs. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the BEL object, "IS_OCCUPIED" and "TYPE" are two of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) This option can be used with the **get_sites** command to return the BELs of specified site objects.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match BELs against the specified patterns. The default pattern is the wildcard '*' which gets a list of all BELs on the device. More than one search pattern can be specified to find BELs based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the total number of BELs on the target part:

```
llength [get_bels]
```

The following example returns the BELs associated with the specified site:

```
get_bels -of_objects [get_sites PHASER_IN_PHY_X0Y5]
```

See Also

- [get_sites](#)
- [list_property](#)
- [report_property](#)

get_board_interfaces

Gets the list of interfaces in the board that implement busdef specified by VLNV.

Syntax

```
get_board_interfaces [-regexp] [-nocase] [-filter arg] [-quiet]  
[-verbose] [patterns...]
```

Returns

List of bus interfaces

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match Bus Interface names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

Categories

[Object](#), [Project](#), [Board](#)

Description

Gets a list of board interfaces specified on the board in use by the current project.

The `current_board` command returns the board in use by the current project. The `get_boards` command returns a list of boards available for use by the current project.

The board file, `board.xml` located in the `data/boards` folder of the Vivado Design Suite installation area, stores information regarding board attributes. The board provides a representation of the overall system that the Xilinx device is a part of, and can help define key aspects of the FPGA design, such as clock constraints, I/O port assignments, and supported interfaces.

The interfaces defined on the board are stored in the `board_rtl.xml` file in the `data/boards` folder of the Vivado Design Suite installation area. The `board_rtl.xml` file is referenced in the `board.xml` file.

The interfaces available in the **current_board** can be used to define the interfaces required in the IP subsystem design, using **create_bd_interface_port** or **create_bd_port**, or in the overall FPGA design using **create_interface** and **create_port**.

Arguments

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_board_interfaces** based on property values on the interfaces. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example gets a list of all interfaces in the current board:

```
join [get_board_interfaces] \n
```

See Also

- [create_interface](#)
- [current_board](#)
- [current_project](#)
- [get_board_pins](#)
- [get_boards](#)

get_board_pins

Gets the list of board pins object.

Syntax

```
get_board_pins [-regexp] [-nocase] [-filter arg] [-of_objects args]  
[-quiet] [-verbose] [patterns...]
```

Returns

List of pins in the board

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get " objects of these types: 'unknown'.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match Board Pin names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

Categories

[Object](#), [Project](#), [Board](#)

Description

Gets a list of board pin objects in the board in use by the current project.

The `current_board` command returns the board in use by the current project. The `get_boards` command returns a list of boards available for use by the current project.

The board file, `board.xml` located in the `data/boards` folder of the Vivado Design Suite installation area, stores information regarding board attributes. The board provides a representation of the overall system that the Xilinx device is a part of, and can help define key aspects of the FPGA design, such as clock constraints, I/O port assignments, and supported interfaces.

The pins in use on the board are stored in the `board_pinmap.xml` file in the `data/boards` folder of the Vivado Design Suite installation area. The `board_pinmap.xml` file is referenced in the `board.xml` file.

The board pin represents the pin on the physical board, system, or targeted reference design (TRD) in which the Xilinx FPGA package pin is connected. It consists of properties like LOC, IOSTANDARD, and SLEW. Board pins can be scalar or vector, so it is always represented as bitwise.

The board_pins can be used to define and place PORTS in the FPGA design, using the **create_port** and **set_property PACKAGE_PIN** commands.

Arguments

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_board_pins** based on property values on the board pins. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects *args* - (Optional) Get the board pins of the specified board interface objects.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example gets a list of board pins assigned to the LED_8Bits board interface, stores those pins in a Tcl variable, and then prints the LOC property for each of those pins:

```
set boardPins [get_board_pins \
-of_objects [get_board_interfaces -filter {NAME == LED_8Bits}]]
foreach pin $boardPins {puts "The location of $pin is: [get_property LOC $pin]"}
The location of LEDs_8Bits_TRI_O[0] is: AB8
The location of LEDs_8Bits_TRI_O[1] is: AA8
The location of LEDs_8Bits_TRI_O[2] is: AC9
The location of LEDs_8Bits_TRI_O[3] is: AB9
The location of LEDs_8Bits_TRI_O[4] is: AE26
The location of LEDs_8Bits_TRI_O[5] is: G19
The location of LEDs_8Bits_TRI_O[6] is: E18
The location of LEDs_8Bits_TRI_O[7] is: F16
```

See Also

- [create_interface](#)
- [current_board](#)
- [current_project](#)
- [get_board_interfaces](#)
- [get_boards](#)

get_boards

Get the list of boards available in the project.

Syntax

```
get_boards [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns...]
```

Returns

List of board objects

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match Board names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

Categories

[Object](#), [Project](#), [XPS](#)

Description

Gets a list of evaluation boards available for use by the current project.

The board in use by the current project is returned by the **current_board** command.

The board file, `board.xml` located in the `data/boards` folder of the Vivado Design Suite installation area, stores information regarding board attributes. The board provides a representation of the overall system that the Xilinx device is a part of, and can help define key aspects of the FPGA design, such as clock constraints, I/O port assignments, and supported interfaces.

The board is specified as part of a Targeted Reference Design, when the project is defined; or by setting the BOARD property on the current project; or by selecting the Project Device in the Project Settings dialog box in the Vivado IDE. Refer to the *Vivado Design Suite User Guide: Using the Vivado IDE (UG893)* for more information on project settings.

Important! When you specify the board with the **set_property** command, the target part is also changed to match the part required by the specified BOARD property.

Arguments

-filter - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_boards** based on property values on the boards. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the board object, "NAME", "DEVICE", and "FAMILY" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the properties of the specified evaluation board:

```
report_property [get_boards -filter {LIBRARY_NAME==artix7}]
```

See Also

- [current_board](#)
- [get_board_interfaces](#)
- [get_board_pins](#)
- [list_property](#)
- [report_property](#)
- [set_property](#)

get_cells

Get a list of cells in the current design.

Syntax

```
get_cells [-hsc arg] [-hierarchical] [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-match_style arg] [-quiet] [-verbose] [patterns]
```

Returns

List of cell objects

Usage

Name	Description
<code>[-hsc]</code>	Hierarchy separator Default: /
<code>[-hierarchical]</code>	Search level-by-level in current instance
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when -regexp specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get cells of these pins, timing paths, nets, bels or sites
<code>[-match_style]</code>	Style of pattern matching Default: sdc Values: ucf, sdc
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match cell names against patterns Default: *

Categories

[SDC](#), [XDC](#), [Object](#)

Description

Gets a list of cell objects in the current design that match a specified search pattern. The default command returns a list of all cells in the design.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-hsc *arg* - (Optional) Set the hierarchy separator. The default hierarchy separator is '/'.

-hierarchical - (Optional) Get cells from all levels of the design hierarchy. Without this argument, the command will only get cells from the top of the design hierarchy. When using **-hierarchical**, the search pattern should not contain a hierarchy separator because the search pattern is applied at each level of the hierarchy, not to the full hierarchical cell name. For instance, searching for U1/* searches each level of the hierarchy for instances with U1/ in the name. This may not return the intended results. This is illustrated in the examples below.

Important! When used with **-regex**, the specified search string is matched against the full hierarchical name, and the U1/* search pattern will work as intended

-regex - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add ".*" to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regex** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regex.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regex** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_cells** based on property values on the cells. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "cell" object, "IS_PARTITION", "IS_PRIMITIVE" and "IS_LOC_FIXED" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects *arg* - (Optional) Get the cells connected to the specified pin or net objects.

Note **-of_objects** cannot be used with **-hierarchy** or with a search *pattern*

-match_style - (Optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match cells against the specified patterns. The default pattern is the wildcard '*' which gets a list of all cells in the project. More than one pattern can be specified to find multiple cells based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example searches all levels of the hierarchy for cells beginning with `cpu`, or `fft`, and joins each cell returned with the newline character to put it on a separate line:

```
join [get_cells -hier {cpu* fft*}] \n
```

This example gets a list of properties and property values attached to the second object of the list returned by `get_cells`:

```
report_property [lindex [get_cells] 1]
```

Note If there are no cells matching the pattern you will get a warning.

This example prints a list of the library cells instantiated into the design at all levels of the hierarchy, sorting the list for unique names so that each cell is only printed one time:

```
foreach cell [lsort -unique [get_property LIB_CELL [get_cells -hier \
-filter {IS_PRIMITIVE==1}]]] {puts $cell}
```

The following example demonstrates the effect of **-hierarchical** searches, without and with **-regexp**:

```
get_cells -hierarchical *mmcm*
mmcm_replicator_inst_1
mmcm_replicator_inst_1/mmcm_stage[0].mmcm_channel[0].mmcm
get_cells -hierarchical -regexp .*mmcm.*
mmcm_replicator_inst_1
mmcm_replicator_inst_1/mmcm_stage[0].mmcm_channel[0].mmcm
mmcm_replicator_inst_1/mmcm_stage[0].mmcm_channel[0].mmcm/GND
mmcm_replicator_inst_1/mmcm_stage[0].mmcm_channel[0].mmcm/MMCM_Base
```

Note The last two cells (GND and MMCM_Base) were not returned in the first example (without **-regexp**) because the cell names do not match the search pattern, as it is applied to each level of the hierarchy.

See Also

- [get_lib_cells](#)
- [get_nets](#)
- [get_pins](#)
- [list_property](#)
- [report_property](#)

get_clock_regions

Get the clock regions for the current device.

Syntax

```
get_clock_regions [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

Returns

Clock_regions

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions.
<code>[-nocase]</code>	Perform case-insensitive matching. (valid only when <code>-regexp</code> specified).
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get the clock_regions of these sites, or cells
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match objects' name against patterns. Default: *

Categories

[Object](#)

Description

Gets a list of clock regions on the target part that match a specified search pattern. The default command gets a list of all clock regions on the device in an open design.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_clock_regions** based on property values on the clock regions. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the clock region object, "COLUMN_INDEX", "HIGH_X", and "LOW_X" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) This option can be used with the **get_sites** command to return the clock region that the specified site is found in.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match clock regions against the specified patterns. The default pattern is the wildcard "*" which gets a list of all clock regions on the device. More than one search pattern can be specified to find clock regions based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the clock regions matching the search pattern:

```
get_clock_regions X0*
```

The following example returns the clock regions filtered by the specified property:

```
get_clock_regions -filter {LOW_X==0}
```

Note These two examples return the same set of clock regions

See Also

- [get_sites](#)
- [list_property](#)
- [report_property](#)

get_clocks

Get a list of clocks in the current design.

Syntax

```
get_clocks [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-match_style arg] [-include_generated_clocks] [-quiet] [-verbose]
[patterns]
```

Returns

List of clocks

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get clocks of these pins or nets
<code>[-match_style]</code>	Style of pattern matching, valid values are <code>ucf</code> , <code>sdcl</code> Default: <code>sdcl</code>
<code>[-include_generated_clocks]</code>	Include auto-inferred/generated_clocks also.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match clock names against patterns Default: *

Categories

[SDC](#), [XDC](#), [Object](#)

Description

Gets a list of clocks in the current design that match a specified search pattern. The default command gets a list of all clocks in the design, like the **all_clocks** command.

Clocks can be created using the **create_clock** or the **create_generated_clock** commands, or can be automatically generated by the tool, at the output of MMCM for instance.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `".*"` to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_clocks** based on property values on the clocks. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the clock object, "PERIOD", "WAVEFORM", and "IS_GENERATED" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (`==` and `!=`), and "contains" and "not-contains" (`=~` and `!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by AND and OR (`&&` and `||`). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) Get the clocks connected to the specified pin or net objects.

Note **-of_objects** cannot be used with a search *pattern*

-match_style - (Optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

-include_generated_clocks - (Optional) Returns all clocks, including generated clocks that match the specified pattern as the source or master clock. This argument should be used when clock *patterns* are specified in order to return generated clocks of the specified master clocks.

Note You can get just the generated clocks with the **get_generated_clocks** command.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match clocks against the specified patterns. The default pattern is the wildcard '*' which gets all clocks in the project. More than one pattern can be specified to find multiple clocks based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example gets a list of clocks matching the various search patterns:

```
get_clocks {*clock *ck *Clk}
```

Note If there are no clocks matching the pattern you will get a warning.

The following example gets the master clock object, and all generated clocks derived from that clock:

```
get_clocks -include_generated_clocks wbClk
```

The following example gets all properties and property values attached to the specified clock:

```
report_property -all [get_clocks wbClk]
```

See Also

- [all_clocks](#)
- [create_clock](#)
- [create_generated_clock](#)
- [get_generated_clocks](#)
- [list_property](#)
- [report_property](#)

get_debug_cores

Get a list of debug cores in the current design.

Syntax

```
get_debug_cores [-filter arg] [-of_objects args] [-regexp] [-nocase]
[-quiet] [-verbose] [patterns]
```

Returns

List of debug_core objects

Usage

Name	Description
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get cores of these debug ports or nets
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when -regexp specified)
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match debug cores against patterns Default: *

Categories

Object

Description

Gets a list of LabTools debug cores in the current project that match a specified search pattern. The default command gets a list of all debug cores in the project.

Debug cores are added to the project with the **create_debug_core** command. When a ILA debug core (labtools_ila_v2) is added to the project, it is contained within a Debug Hub core (labtools_xsdbmasterlib_v2), and includes a CLK port and a trigger port (TRIG) as a default. Additional ports can be added to the debug core with the use of the create_debug_port command.

Note To improve memory and performance, the get_* commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_debug_cores** based on property values on the parts. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects *args* - (Optional) Get the ChipScope debug cores associated with the specified debug ports, or nets.

Note **-of_objects** cannot be used with a search *pattern*

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match debug cores against the specified patterns. The default pattern is the wildcard "*" which gets all debug cores. More than one pattern can be specified to find multiple debug cores based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following command gets a list of the ChipScope debug cores in the current project:

```
get_debug_cores
```

Note An ICON core is returned as one of the debug cores in the project. You cannot directly create this core, but it is automatically added by the tool when you add any ILA cores to the project.

The following example gets the properties of the specified debug core:

```
report_property [get_debug_cores myCore]
```

The values of the properties returned depend on how the core is configured. You can use the **set_property** command to configure specific core properties as shown in the following example:

```
set_property enable_storage_qualification false [get_debug_cores myCore]
```

See Also

- [create_debug_core](#)
- [create_debug_port](#)
- [get_debug_ports](#)
- [list_property](#)
- [report_property](#)
- [set_property](#)

get_debug_ports

Get a list of debug ports in the current design.

Syntax

```
get_debug_ports [-filter arg] [-of_objects args] [-regexp] [-nocase]
[-quiet] [-verbose] [patterns]
```

Returns

List of debug_port objects

Usage

Name	Description
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get ports of these debug cores
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when -regexp specified)
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match debug ports against patterns Default: *

Categories

Object

Description

Gets a list of ports defined on ILA debug cores in the current project that match a specified search pattern. The default command gets a list of all debug ports in the project.

Debug ports are defined when ILA debug cores are created with the **create_debug_core** command. Ports can be added to existing debug cores with the **create_debug_port** command.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_debug_ports** based on property values on the ports. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the debug_port object, "PORT_WIDTH", and "MATCH_TYPE" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects *args* - (Optional) Get the ChipScope debug ports associated with the specified debug cores.

Note **-of_objects** cannot be used with a search *pattern*

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match debug ports against the specified patterns. The default pattern is the wildcard "*" which gets all debug ports. More than one pattern can be specified to find multiple debug ports based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following command gets a list of the ports from the ILA debug cores in the current project, with a PORT_WIDTH property of 8:

```
get_debug_ports -filter {PORT_WIDTH==8}
```

The following example gets the properties attached to the specified debug port:

```
report_property [get_debug_ports myCore/TRIG0]
```

Note The debug port is defined by the core_name/port_name combination.

See Also

- [create_debug_core](#)
- [create_debug_port](#)
- [list_property](#)
- [report_property](#)

get_delays

Returns delay objects.

Syntax

```
get_delays [-regexp] [-nocase] [-filter arg] [-of_objects args]  
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-of_objects]	Get the delays of the objects.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Object](#)

get_designs

Get a list of designs in the current project.

Syntax

```
get_designs [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns]
```

Returns

List of design objects

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match design names against patterns Default: *

Categories

[Object](#)

Description

Gets a list of open designs in the current project that match a specified search pattern. The default command gets a list of all open designs in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_designs** based on property values on the designs. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "design" object, "CONSTRSET", and "PART" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match designs against the specified patterns. The default pattern is the wildcard '*' which gets all designs. More than one pattern can be specified to find multiple designs based on different search criteria.

Examples

The following example gets all open designs in the current project:

```
get_designs
```

The following example gets the assigned properties of an open design matching the search pattern:

```
report_property [get_designs r*]
```

Note If there are no designs matching the pattern you will get a warning.

See Also

[report_property](#)

get_drc_checks

Get a list of drc rule check objects.

Syntax

```
get_drc_checks [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-abbrev arg] [-quiet] [-verbose] [patterns]
```

Returns

Drc_check

Usage

Name	Description
<code>[-of_objects]</code>	Get 'drc_rule' objects of these types: 'drc_ruledeck'.
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching. (valid only when -regexp specified)
<code>[-filter]</code>	Filter list with expression
<code>[-abbrev]</code>	Get the largest ID for this abbrev
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match the 'drc_rule' objects against patterns. Default: *

Categories

[DRC](#), [Object](#)

Description

Gets a list of the currently defined DRC checks. This list includes both factory defined design rule checks, and user-defined checks created by the **create_drc_check** command.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-of_objects *args* - (Optional) Get the design rule checks associated with a rule deck object. The ruledeck object must be specified by the **get_drc_ruledeck** command.

Note **-of_objects** cannot be used with a search *pattern*

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of rule checks returned by **get_drc_checks** based on property values on the rule checks. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-abbrev *arg* - (Optional) Get the design rule checks associated with the specified rule name or abbreviation.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match design rule checks against the specified patterns. The default pattern is the wildcard '*' which gets all rule checks. More than one pattern can be specified to find multiple rule checks based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following command gets a list of all AVAL design rule checks:

```
get_drc_checks AVAL*
```

The following example gets the checks associated with the specified rule deck:

```
get_drc_checks -of_objects [get_drc_ruledeck placer_checks]
```

See Also

- [create_drc_check](#)
- [get_drc_ruledecks](#)
- [list_property](#)
- [report_drc](#)
- [report_property](#)

get_drc_ruledecks

Get a list of drc rule deck objects.

Syntax

```
get_drc_ruledecks [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns]
```

Returns

Drc_ruledeck

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching. (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match the 'drc_ruledeck' objects against patterns. Default: *

Categories

[DRC](#), [Object](#)

Description

Gets a list of currently defined rule decks for use with the **report_drc** command.

A rule deck is a collection of design rule checks grouped for convenience, to be run at different stages of the FPGA design flow, such as during I/O planning or placement. The tool comes with a set of factory defined rule decks, but you can also create new user-defined rule decks with the **create_drc_ruledeck** command, and add checks to the rule deck using the **add_drc_checks** command.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_drc_ruledecks** based on property values on the parts. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match rule decks against the specified patterns. The default pattern is the wildcard '*' which gets all rule decks. More than one pattern can be specified to find multiple rule decks based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example gets a list of rule decks defined in the current project:

```
get_drc_ruledecks
```

The following example lists each of the checks associated with the placer_checks rule deck on a separate line:

```
foreach rule [get_drc_checks -of_objects [get_drc_ruledecks placer_checks]] \
    {puts $rule}
```

See Also

- [add_drc_checks](#)
- [create_drc_ruledeck](#)
- [list_property](#)
- [report_drc](#)
- [report_property](#)

get_drc_vios

Get a list of drc violations from a previous report_drc run.

Syntax

```
get_drc_vios [-name arg] [-regexp] [-filter arg] [-nocase] [-quiet]
[-verbose] [patterns]
```

Returns

List of vio objects

Usage

Name	Description
[-name]	Get the results with this name
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-nocase]	Perform case-insensitive matching (valid only when -regexp specified)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match drc_vios against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

Categories

DRC, Object

Description

Gets a list of violation objects created by the **create_drc_violation** command when the associated DRC checks are run by the **report_drc** command. The properties of individual violation objects can be queried using **report_property** or **list_property** commands for details of the violation.

Note To improve memory and performance, the get_* commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-name *arg* - (Optional) Get the violations associated with the named DRC result set. In this case the **report_drc** command must have been run with the **-name** option.

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add ".*" to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_drc_vios** based on property values on the violations. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match violations against the specified patterns. The default pattern is the wildcard '*' which gets all violations. More than one pattern can be specified to find multiple violations based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example reports the DRC violations found in the current design, then returns a list of all those violations:

```
report_drc
get_drc_vios
```

The following example gets the properties of the specified violation:

```
report_property [lindex [get_drc_vios] 0]
```

The following example returns the list of violations that match the specified search pattern in the named DRC results set:

```
get_drc_vios -name drc_1 {*BUF* *DPIP*}
```

See Also

- [create_drc_check](#)
- [create_drc_violation](#)
- [report_drc](#)

get_files

Get a list of source files.

Syntax

```
get_files [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-compile_order] [-quiet] [-verbose] [patterns]
```

Returns

List of file objects

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get files of these filesets or composite files
<code>[-compile_order]</code>	Give files ordered for synthesis or simulation for the object specified (valid only with <code>-of_objects</code>)
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[<i>patterns</i>]</code>	Match file names against patterns Default: *

Categories

[Object](#), [Project](#)

Description

Gets a list of files in the current project that match a specified search pattern. The default command gets a list of all files in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_files** based on property values on the files. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the "file" object, "FILE_TYPE", and "IS_ENABLED" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) Specifies one or more filesets to search for the files. The default is to search all filesets.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match files against the specified patterns. The default pattern is the wildcard '*' which gets all files in the project or of_objects. More than one pattern can be specified to find multiple files based on different search criteria.

Examples

The following example returns the Verilog files in the design:

```
get_files -filter {FILE_TYPE == Verilog}
```

The following example gets a list of the Verilog files (*.v) found in the constrs_1 and sim_1 filesets:

```
get_files -of_objects {constrs_1 sim_1} *.v
```

Note If there are no files matching the pattern you will get a warning.

See Also

[report_property](#)

get_filesets

Get a list of filesets in the current project.

Syntax

```
get_filesets [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns]
```

Returns

List of fileset objects

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>patterns</i>]	Match fileset names against patterns Default: *

Categories

[Object](#), [Project](#)

Description

Gets a list of filesets in the current project that match a specified search pattern. The default command gets a list of all filesets in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_filesets** based on property values on the filesets. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the fileset object, "DESIGN_MODE", and "FILESET_TYPE" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match fileset names against the specified patterns. The default pattern is the wildcard '*' which gets all filesets. More than one pattern can be specified to find filesets based on multiple search criteria.

Examples

The following example returns the source files in the Source Set:

```
get_files -of_objects [get_filesets sources_1]
```


The following example makes `project_2` the active project, and then gets a list of filesets beginning with the letter `s` or the letter `r`:

```
current_project project_2
get_filesets s* r* -quiet
```

Note If there are no filesets matching the pattern, such as `r*`, you will get a warning that no filesets matched the specified pattern. However, in the above example the use of **-quiet** will suppress that warning message.

The following example gets filesets beginning with the letter `C`, using a case insensitive regular expression:

```
get_filesets C.* -regexp -nocase
```

In the above example, `constrs_1` and `constrs_2` constraint sets would be returned if defined in the current project.

See Also

- [get_files](#)
- [report_property](#)

get_generated_clocks

Get a list of generated clocks in the current design.

Syntax

```
get_generated_clocks [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-match_style arg] [-quiet] [-verbose] [patterns]
```

Returns

List of clocks

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get generated clocks of these pins or nets
<code>[-match_style]</code>	Style of pattern matching, valid values are <code>ucf</code> , <code>sdh</code> Default: <code>sdh</code>
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match generated clock names against patterns Default: <code>*</code>

Categories

[XDC](#), [Object](#)

Description

Gets a list of generated clocks in the current project that match a specified search pattern. The default command gets a list of all generated clocks in the project.

A generated clock can be added to the design using the **create_generated_clock** command, or can be automatically generated by the tool, at the output of MMCM for instance.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "."*" to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_generated_clocks** based on property values on the clocks. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the generated_clock object, "DUTY_CYCLE" and "MASTER_CLOCK" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) One or more pins or nets to which the generated clocks are assigned.

Note **-of_objects** cannot be used with a search *pattern*

-match_style - (Optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match generated clocks against the specified patterns. The default pattern is the wildcard "*" which gets all generated clocks in the project.

Examples

The following example gets the names of all generated clocks in the current project:

```
get_generated_clocks
```

See Also

- [create_generated_clock](#)
- [get_clocks](#)
- [report_property](#)

get_hierarchy_separator

Get hierarchical separator character.

Syntax

```
get_hierarchy_separator [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

Gets the character currently used for separating levels of hierarchy in the design. You can set the hierarchy separator using the **set_hierarchy_separator** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example gets the currently defined hierarchy separator:

```
get_hierarchy_separator
```

See Also

[set_hierarchy_separator](#)

get_hw_devices

Get a list of hardware devices.

Syntax

```
get_hw_devices [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware devices

Usage

Name	Description
[-of_objects]	Get 'hw_device' objects of these types: 'hw_target'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_device' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_ila_datas

Get a list of hardware ILA data objects.

Syntax

```
get_hw_ila_datas [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware ILA data

Usage

Name	Description
[-of_objects]	Get 'hw_ila_data' objects of these types: 'hw_ila'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_ila_data' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_ilas

Get a list of hardware ILA.

Syntax

```
get_hw_ilas [-of_objects args] [-regexp] [-nocase] [-filter arg]  
[-quiet] [-verbose] [patterns]
```

Returns

Hardware ILAs

Usage

Name	Description
[-of_objects]	Get 'hw_ila' objects of these types: 'hw_device'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match the 'hw_ila' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_probes

Get a list of hardware probes.

Syntax

```
get_hw_probes [-of_objects args] [-regexp] [-nocase] [-filter arg]
               [-quiet] [-verbose] [patterns]
```

Returns

Hardware probes

Usage

Name	Description
[-of_objects]	Get 'hw_probe' objects of these types: 'hw_ila hw_vio'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match the 'hw_probe' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_servers

Get a list of hardware servers.

Syntax

```
get_hw_servers [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns]
```

Returns

Hardware servers

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_server' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_sio_commons

Get list of hardware SIO GT commons.

Syntax

```
get_hw_sio_commons [-of_objects args] [-regexp] [-nocase] [-filter arg]  
[-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO GT commons

Usage

Name	Description
[-of_objects]	Get 'hw_sio_common' objects of these types: 'hw_server hw_target hw_device hw_sio_ibert hw_sio_pll'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_sio_common' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_sio_gtgroups

Get list of hardware SIO GT groups.

Syntax

```
get_hw_sio_gtgroups [-of_objects args] [-regex] [-nocase]  
[-filter arg] [-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO GT groups

Usage

Name	Description
[-of_objects]	Get 'hw_sio_gtgroup' objects of these types: 'hw_server hw_target hw_device hw_sio_ibert hw_sio_common hw_sio_pll hw_sio_gt hw_sio_tx hw_sio_rx'.
[-regex]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regex specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>patterns</i>]	Match the 'hw_sio_gtgroup' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_sio_gts

Get list of hardware SIO GTs.

Syntax

```
get_hw_sio_gts [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO GTs

Usage

Name	Description
[-of_objects]	Get 'hw_sio_gt' objects of these types: 'hw_server hw_target hw_device hw_sio_ibert hw_sio_pll hw_sio_tx hw_sio_rx hw_sio_link'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_sio_gt' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_sio_iberts

Get list of hardware SIO IBERT cores.

Syntax

```
get_hw_sio_iberts [-of_objects args] [-regexp] [-nocase] [-filter arg]  
[-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO IBERT cores

Usage

Name	Description
[-of_objects]	Get 'hw_sio_ibert' objects of these types: 'hw_server hw_target hw_device hw_sio_gt hw_sio_common hw_sio_pll hw_sio_tx hw_sio_rx hw_sio_link'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_sio_ibert' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_sio_linkgroups

Get list of hardware SIO link groups.

Syntax

```
get_hw_sio_linkgroups [-of_objects args] [-regexp] [-nocase]  
[-filter arg] [-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO link groups

Usage

Name	Description
[-of_objects]	Get 'hw_sio_linkgroup' objects of these types: 'hw_sio_link'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match the 'hw_sio_linkgroup' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_sio_links

Get list of hardware SIO links.

Syntax

```
get_hw_sio_links [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO links

Usage

Name	Description
[-of_objects]	Get 'hw_sio_link' objects of these types: 'hw_server hw_target hw_device hw_sio_ibert hw_sio_gt hw_sio_tx hw_sio_rx hw_sio_linkgroup'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_sio_link' objects against patterns. Default: *

Categories

Hardware, Object

get_hw_sio_plls

Get list of hardware SIO PLLs.

Syntax

```
get_hw_sio_plls [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO PLLs

Usage

Name	Description
[-of_objects]	Get 'hw_sio_pll' objects of these types: 'hw_server hw_target hw_device hw_sio_ibert hw_sio_gt hw_sio_common'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_sio_pll' objects against patterns. Default: *

Categories

Hardware, Object

get_hw_sio_rxs

Get list of hardware SIO RXs.

Syntax

```
get_hw_sio_rxs [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO RXs

Usage

Name	Description
[-of_objects]	Get 'hw_sio_rx' objects of these types: 'hw_server hw_target hw_device hw_sio_ibert hw_sio_gt hw_sio_link'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_sio_rx' objects against patterns. Default: *

Categories

Hardware, Object

get_hw_sio_scans

Get list of hardware SIO scans.

Syntax

```
get_hw_sio_scans [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO scans

Usage

Name	Description
[-of_objects]	Get 'hw_sio_scan' objects of these types: 'hw_sio_rx'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_sio_scan' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_sio_txs

Get list of hardware SIO TXs.

Syntax

```
get_hw_sio_txs [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware SIO TXs

Usage

Name	Description
[-of_objects]	Get 'hw_sio_tx' objects of these types: 'hw_server hw_target hw_device hw_sio_ibert hw_sio_gt hw_sio_link'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_sio_tx' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_targets

Get a list of hardware targets.

Syntax

```
get_hw_targets [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware targets

Usage

Name	Description
[-of_objects]	Get 'hw_target' objects of these types: 'hw_server'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match the 'hw_target' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_hw_vios

Get a list of hardware VIOs.

Syntax

```
get_hw_vios [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Hardware VIOs

Usage

Name	Description
[-of_objects]	Get 'hw_vio' objects of these types: 'hw_device'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'hw_vio' objects against patterns. Default: *

Categories

[Hardware](#), [Object](#)

get_interfaces

Get a list of I/O port interfaces in the current design.

Syntax

```
get_interfaces [-regexp] [-nocase] [-filter arg] [-of_objects args]
               [-quiet] [-verbose] [patterns]
```

Returns

List of interface objects

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get interfaces of these pins or nets
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match I/O port interfaces against patterns Default: *

Categories

[Object](#)

Description

Gets a list of IO interfaces in the current project that match a specified search pattern. The default command gets a list of all IO interfaces in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_interfaces** based on property values on the interfaces. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) One or more pins or nets to which the interfaces are assigned.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - Match interfaces against the specified pattern. The default pattern is the wildcard '*' which gets a list of all interfaces in the project.

Examples

The following example gets a list of all interfaces in the project:

```
get_interfaces
```

See Also

- [create_interface](#)
- [delete_interface](#)

get_io_standards

Get a list of IO standards.

Syntax

```
get_io_standards [-regexp] [-nocase] [-filter arg] [-of_objects args]  
[-quiet] [-verbose] [patterns]
```

Returns

IO standards

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-of_objects]	Get the IO standards of these bels, sites, package_pins, io_banks, ports.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match IO standards against patterns Default: *

Categories

[Object](#)

get_iobanks

Get a list of iobanks.

Syntax

```
get_iobanks [-regexp] [-nocase] [-filter arg] [-of_objects args]  
[-quiet] [-verbose] [patterns]
```

Returns

Iobanks

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching. (valid only when -regexp specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get the iobanks of these package_pins.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match iobanks against patterns Default: *

Categories

[XDC](#), [Object](#)

Description

Gets a list of I/O Banks on the target device in the current project that match a specified search pattern. The default command gets a list of all I/O Banks on the target device.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_iobanks** based on property values on the I/O Banks. You can find the properties on an object with the **report_property** or **list_property** commands. Some of the properties that can be used with for an iobank object include "DCI_CASCADE", and "INTERNAL_VREF".

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects arg - (Optional) Get a list of the I/O Banks connected to these objects. Valid object types are package_pins, ports, and sites.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match I/O Banks against the specified pattern. The default pattern is the wildcard "*" which gets a list of all I/O Banks in the design.

Examples

The following example returns the I/O Bank of the specified package pin:

```
get_iobanks -of_objects [get_package_pins H4]
```

See Also

- [get_package_pins](#)
- [get_ports](#)
- [get_sites](#)
- [place_ports](#)
- [report_property](#)

get_ipdefs

Get a list of IP from the current IP Catalog.

Syntax

```
get_ipdefs [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]


```

Returns

List of Catalog IP objects

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match Catalog IP names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

Categories

Object, IPFlow

Description

Get a list of IP cores defined in the IP catalog of the current project, based on the specified search pattern. The default is to return all IP cores defined in the catalog.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_ipdefs** based on property values on the objects. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "ipdefs" object, "VLNV", "NAME" and "IS_AXI" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match IP core definitions in the IP catalog against the specified search patterns. The default pattern is the wildcard '*' which gets a list of all IP cores in the catalog. More than one pattern can be specified to find multiple core definitions based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns a list of all IP cores with NAME property matching the specified pattern:

```
get_ipdefs -filter {NAME=~*agilent*}
```

Note The filter operator '=~' loosely matches the specified pattern

The following example returns a list of all AXI compliant IP cores:

```
get_ipdefs -filter {IS_AXI==1}
```

See Also

- [create_ip](#)
- [generate_target](#)
- [get_ips](#)
- [import_ip](#)
- [update_ip_catalog](#)

get_ips

Get a list of IPs in the current design.

Syntax

```
get_ips [-regex] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns...]
```

Returns

List of IP objects

Usage

Name	Description
<code>[-regex]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching
<code>[-filter]</code>	Filter list with expression
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match IP names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regex is specified.

Categories

[Object](#), [Project](#), [IPFlow](#)

Description

Get a list of IP cores in the current project based on the specified search pattern. The default command returns a list of all IPs in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_ips** based on property values on the objects. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "IP" object, "NAME" and "DELIVERED_TARGETS" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match IP cores in the design against the specified search patterns. The default pattern is the wildcard '*' which gets a list of all IP cores in the project. More than one pattern can be specified to find multiple cores based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns a list of IP cores with names beginning with the string "EDK":

```
get_ips EDK*
```

See Also

- [create_ip](#)
- [generate_target](#)
- [get_ipdefs](#)
- [import_ip](#)
- [update_ip_catalog](#)

get_lib_cells

Get a list of Library Cells.

Syntax

```
get_lib_cells [-regexp] [-filter arg] [-nocase] [-include_unsupported]
[-of_objects args] [-quiet] [-verbose] patterns
```

Returns

List of library cells

Usage

Name	Description
[-regexp]	Patterns are regular expressions
[-filter]	Filter list with expression
[-nocase]	Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of -regexp only.
[-include_unsupported]	Include test-only library cells.
[-of_objects]	Get the library cells of the objects passed in here: .
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>patterns</i>	Match library cell names against patterns.

Categories

Object

Description

Get a list of cells in the library for the target part of the current design. Use this command to query and look for a specific library cell, or type of cell and to get the properties of the cells.

This command requires a hierarchical name which includes the library name as well as the cell name: lib_name/cell_name.

Note To improve memory and performance, the get_* commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_lib_cells** based on property values on the cells. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects arg - (Optional) Get a list of library cells of specific instances (cells/insts), or library pins (get_lib_pins).

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Required) Match library cells against the specified patterns. The pattern must specify both the library name and the cell name.

Examples

The following example gets the number of the cells in the library for the target part in the current design, and then gets the number of AND type cells in that library:

```
llength [get_lib_cells [get_libs]/*]
795
llength [get_lib_cells [get_libs]/AND*]
18
```

The following example gets the library cell for the specified cell object:

```
get_lib_cells -of_objects [lindex [get_cells] 1]
```

See Also

- [get_cells](#)
- [get_libs](#)
- [get_lib_pins](#)
- [list_property](#)
- [report_property](#)

get_lib_pins

Get a list of Library Cell Pins.

Syntax

```
get_lib_pins [-regexp] [-filter arg] [-nocase] [-of_objects args]  
[-quiet] [-verbose] patterns
```

Returns

List of library cell pins

Usage

Name	Description
[-regexp]	Patterns are regular expressions
[-filter]	Filter list with expression
[-nocase]	Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of -regexp only.
[-of_objects]	Get the library cell pins of the objects passed in here.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>patterns</i>	Match library cell pin names against patterns of the form /.

Categories

[Object](#)

Description

Gets a list of the pins on a specified cell of the cell library for the target part in the current design.

Note This command requires a hierarchical name which includes the library name and cell name as well as the pins: lib_name/cell_name/pins.

Note To improve memory and performance, the get_* commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regex - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regex** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regex.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regex** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_lib_pins** based on property values on the pins. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects *arg* - (Optional) Get a list of library cell pins of the specified pin objects or library cells (**get_lib_cells**).

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Required) Match lib pins against the specified patterns. The pattern must specify the library name, cell name, and the pins.

Examples

The following example gets a list of all library cell pins:

```
get_lib_pins xt_virtex6/AND2/*
```

The following example gets a list of all pins, of all cells in the cell library for the target device:

```
get_lib_pins [get_libs]/*/*
```

See Also

- [get_libs](#)
- [get_lib_cells](#)
- [list_property](#)
- [report_property](#)

get_libs

Get a list of Libraries.

Syntax

```
get_libs [-regexp] [-filter arg] [-nocase] [-quiet] [-verbose]  
[patterns]
```

Returns

List of libraries

Usage

Name	Description
[-regexp]	Patterns are regular expressions
[-filter]	Filter list with expression
[-nocase]	Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of -regexp only.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match library names against patterns. Default: *

Categories

[Object](#)

Description

Gets the cell library for the target device in the current design. There is a library for each device family because there are primitives that may be available in one device family but not in others.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by `get_libs` based on property values on the libs. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects arg - (Optional) Get a list of libraries of the specified object.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match libraries against the specified patterns. The default pattern is the wildcard '*' which gets a list of all libraries in the project.

Examples

The following example gets the cell library for the target part:

```
get_libs
```

See Also

- [get_lib_cells](#)
- [get_lib_pins](#)
- [list_property](#)
- [report_property](#)

get_macros

Get a list of macros in the current design.

Syntax

```
get_macros [-regexp] [-nocase] [-filter arg] [-of_objects args]  
[-quiet] [-verbose] [patterns]
```

Returns

List of macro objects

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get macros of these cells
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match macro names against patterns Default: *

Categories

[XDC](#), [Object](#)

Description

Gets a list of macros in the current design that match a specified search pattern. The default command returns a list of all macros in the design.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_macros** based on property values on the macros. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "macro" object, "NAME", "ABSOLUTE_GRID" and "RLOCS" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects arg - (Optional) Get the macros connected to the specified pin or net objects.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match macros against the specified patterns. The default pattern is the wildcard '*' which gets a list of all macros in the project. More than one pattern can be specified to find multiple macros based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the properties currently assigned to the macro matching the specified search pattern:

```
report_property [get_macro *Macro1]
```

Note If there are no macros matching the pattern you will get a warning.

See Also

- [create_macro](#)
- [list_property](#)
- [report_property](#)
- [update_macro](#)

get_msg_config

Returns the current message count, limit, or the message configuration rules previously defined by the `set_msg_config` command.

Syntax

```
get_msg_config [-id arg] [-severity arg] [-rules] [-limit] [-count]
               [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-id]</code>	The message id to match. Should be used in conjunction with <code>-limit</code> or <code>-count</code> Default: empty
<code>[-severity]</code>	The message severity to match. Should be used in conjunction with <code>-limit</code> or <code>-count</code> Default: empty
<code>[-rules]</code>	Show a table displaying all message control rules for the current project
<code>[-limit]</code>	Show the limit for the number of messages matching either <code>-id</code> or <code>-severity</code> that will be displayed
<code>[-count]</code>	Show the number of messages matching either <code>-id</code> or <code>-severity</code> that have been displayed
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Report](#)

Description

Returns the current message limit or count applied to a specified message ID or severity, or returns all message configuration rules defined in the current project. Message configuration rules are defined using the **set_msg_config** command.

When used with **-count** this command will display the total number of messages that have been generated with the matching message id, or for the specified severity.

When used with **-limit** this command will display the current limit of messages with the matching message id, or for the specified severity.

When used with **-rules**, it will display a table of all message configuration rules currently in effect.

Note You can only return the limit, the count, or the rules in a single **get_msg_config** command. An error is returned if more than one action is attempted.

Arguments

-id *arg* - (Optional) The message ID, which is included in all returned messages. For example, "Common 17-54" and "Netlist 29-28".

-severity *value* - (Optional) The severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

Note Since this is a two word value, it must be enclosed in {}.

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

-rules - (Optional) Return the message configuration rules in the current project as defined by the **set_msg_config** command.

Note When **-rule** is specified, all configuration rules for the current project are returned regardless of any message qualifier such as **-id** or **-severity**.

-limit - (Optional) Return the current limit of messages that match the message ID or the message severity.

-count - (Optional) Return the current count of messages that match the specified message ID or message severity.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example gets the current count of the specified INFO message:

```
get_msg_config -id "Common 17-81" -count
```


The following example returns the message configuration rules in the current project:

```
get_msg_config -rules
```

See Also

- [reset_msg_config](#)
- [set_msg_config](#)

get_msg_count

Get message count.

Syntax

```
get_msg_count [-severity arg] [-id arg] [-quiet] [-verbose]
```

Returns

Message count

Usage

Name	Description
[-severity]	Message severity to query (not valid with -id,) e.g. "ERROR" or "CRITICAL WARNING" Default: ALL
[-id]	Unique message id to be queried (not valid with -severity,) e.g. "Common 17-99"
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Gets the number of messages, of a specific severity or message ID, that have been returned by the tool since it was invoked.

Every message delivered by the tool has a unique global message ID that consists of an application sub-system code and a message identifier. This results in a message ID that looks like the following:

```
"Common 17-54"  
"Netlist 29-28"  
"Synth 8-3295"
```

This command can give you an idea of how close to the message limit the tool may be getting. You can check the current message limit with the **get_msg_limit** command. You can change the message limit with the **set_msg_limit** command.

By default this command returns the message count for all messages. You can also get the count of a specific severity of message, or for a specific message ID.

Arguments

-severity *value* - (Optional) Specifies the severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.
Note Since this is a two word value, it must be enclosed in {} or "".
- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

-id *value* - (Optional) The message ID is found in the tool in the Messages view or other reports when the message is reported. Use the specific message ID to get the count for that message.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example gets the message count for all messages:

```
get_msg_count -severity ALL  
get_msg_count
```

Note Both lines return the same thing since the default is to return the count for all messages when -severity or -id is not specified.

The following example gets the message count of the specified message ID:

```
get_msg_count -id "Netlist 29-28"
```

See Also

- [get_msg_limit](#)
- [set_msg_limit](#)

get_msg_limit

Get message limit.

Syntax

```
get_msg_limit [-severity arg] [-id arg] [-quiet] [-verbose]
```

Returns

Message limit

Usage

Name	Description
[-severity]	Message severity to query (not valid with -id,) e.g. "ERROR" or "CRITICAL WARNING" Default: ALL
[-id]	Unique message id to be queried (not valid with -severity,) e.g "Common 17-99"
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Gets the number of messages that will be reported by the tool while invoked. When the tool reaches the defined message limit, it stops reporting messages. The default value is 4,294,967,295. This default value can be changed with the **set_msg_limit** command.

Every message delivered by the tool has a unique global message ID that consists of an application sub-system code and a message identifier. This results in a message ID that looks like the following:

```
"Common 17-54"  
"Netlist 29-28"  
"Synth 8-3295"
```

Arguments

-id arg - (Optional) The message ID to return the limit of. For example, "Common 17-54" or "Netlist 29-28".

-severity *value* - (Optional) Specifies the severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

Note Since this is a two word value, it must be enclosed in {}.

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the limit for CRITICAL WARNING messages:

```
get_msg_limit -severity {CRITICAL WARNING}
```

The default when -severity or -id is not specified is to return the limit for all messages.

The following example returns the message limit of the specified message ID:

```
get_msg_limit -id "Netlist 29-28"
```

See Also

- [set_msg_limit](#)
- [set_msg_severity](#)

get_net_delays

Get the routed or estimated delays on a net from the driver to each load pin.

Syntax

```
get_net_delays [-regexp] [-nocase] [-filter arg] [-of_objects args]  
[-through] [-to args] [-quiet] [-verbose] [patterns]
```

Returns

Net_delays

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-of_objects]	Get the net_delays of these nets.
[-through]	Get the delay through the given pip. Default is to not include the pip delay.
[-to]	Get the net_delay to the given load(s).
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>patterns</i>]	Match net_delays against patterns Default: *

Categories

Timing, Netlist, Object

get_nets

Get a list of nets in the current design.

Syntax

```
get_nets [-hsc arg] [-hierarchical] [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-match_style arg] [-top_net_of_hierarchical_group]
[-segments] [-boundary_type arg] [-quiet] [-verbose] [patterns]
```

Returns

List of net objects

Usage

Name	Description
[-hsc]	Hierarchy separator Default: /
[-hierarchical]	Search level-by-level in current instance
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching (valid only when -regexp specified)
[-filter]	Filter list with expression
[-of_objects]	Get nets of these pins/ports,cells,timing paths or clocks
[-match_style]	Style of pattern matching, valid values are ucf, sdc Default: sdc
[-top_net_of_hierarchical_group]	Return net segment(s) which belong(s) to the high level of a hierarchical net
[-segments]	Return all segments of a net across the hierarchy
[-boundary_type]	Return net segment connected to a hierarchical pin which resides at the same level as the pin (upper) or at the level below (lower), or both. Valid values are : upper, lower, both Default: upper
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match net names against patterns Default: *

Categories

SDC, XDC, Object

Description

Gets a list of nets in the current design that match a specified search pattern. The default command gets a list of all nets in the design.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-hsc *arg* - (Optional) The default hierarchy separator is '/'. Use this argument to specify a different hierarchy separator.

-hierarchical - (Optional) Get nets from all levels of the design hierarchy. Without this argument, the command will only get nets from the top of the design hierarchy. When using **-hierarchical**, the search pattern should not contain a hierarchy separator because the search pattern is applied at each level of the hierarchy, not to the full hierarchical cell name. For instance, searching for `U1/*` searches each level of the hierarchy for instances with `U1/` in the name. This may not return the intended results. See **get_cells** for examples of **-hierarchical** searches.

Note When used with **-regexpr**, the specified search string is matched against the full hierarchical name, and the `U1/*` search pattern will work as intended

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `".*"` to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_nets** based on property values on the nets. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the nets object, "PARENT", "TYPE" and "MARK_DEBUG" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (`==` and `!=`), and "contains" and "not-contains" (`=~` and `!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by AND and OR (`&&` and `||`). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```


-of_objects *arg* - (Optional) Get a list of the nets connected to the specified cell, pin, port, or clock.

Note **-of_objects** cannot be used with **-hierarchy** or with a search *pattern*

-match_style [sdc | ucf] - (Optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

-top_net_of_hierarchical_group - (Optional) Get the top net segments of a hierarchical net. Use this argument to return the top-level net name from a lower-level net segment, or to return the top-level net segments of all nets.

-segments - (Optional) Get all the segments of a hierarchical net, across all levels of the hierarchy. This differs from the **-hierarchical** argument in that it returns all segments of the specified net, rather than just the specified net.

-boundary_type - (Optional) Gets the net segment at the level (upper) of a specified hierarchical pin, at the level below (lower) the pin or port, or both the level of and the level below. Valid values are upper, lower, or both. The default value is upper.

Note This argument must be used with the **-of_objects** argument to specify the hierarchical pin.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match nets against the specified patterns. The default pattern is the wildcard '*' which returns a list of all nets in the project. More than one pattern can be specified to find multiple nets based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example gets a list of nets attached to the specified cell:

```
get_nets -of_objects [lindex [get_cells] 1]
```

Note If there are no nets matching the pattern you will get a warning.

The following example returns a list of nets that have been marked for debug with the connect_debug_port command:

```
get_nets -hier -filter {MARK_DEBUG==1}
```

See Also

- [connect_debug_port](#)
- [get_cells](#)
- [get_clocks](#)
- [get_pins](#)
- [get_ports](#)
- [list_property](#)
- [report_property](#)

get_nodes

Get a list of nodes in the device.

Syntax

```
get_nodes [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-uphill] [-downhill] [-flyover] [-from args] [-to args] [-quiet]
[-verbose] [patterns]
```

Returns

Nodes

Usage

Name	Description
[-of_objects]	Get 'node' objects of these types: 'net tile node bel_pin site_pin wire pip'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-uphill]	Get the nodes uphill (driver) from the site_pin, pip, node or tile(s) provided in the -of_objects.
[-downhill]	Get the nodes downhill (loads) from the site_pin, pip, node or tile(s) provided in the -of_objects.
[-flyover]	Get the nodes that fly over the given tile(s).
[-from]	-from Return the nodes beginning at this pip or site pin. May be used in combination with uphill. Default is downhill. -all is implied.
[-to]	-to Return the nodes ending at this wire or site pin. May be used in combination with uphill. Default is downhill. -all is implied.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'node' objects against patterns. Default: *

Categories

Object

Description

Returns a list of nodes on the device that match a specified search pattern in an open design. The default command gets a list of all nodes on the device.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `".*"` to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_nodes** based on property values on the nodes. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the node object, "IS_INPUT_PIN", "IS_BEL_PIN" and "NUM_WIRES" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) Return the nodes of the specified site_pins, nodes, tiles, or wires.

Note **-of_objects** cannot be used with a search *pattern*

-uphill - (Optional) Return the nodes uphill of objects specified by the **-of_objects** option. Uphill nodes precede the specified object in the logic network.

-downhill - (Optional) Return the nodes downhill of objects specified by the **-of_objects** option. Downhill nodes follow the specified object in the logic network.

-flyover - (Optional) Return the nodes that pass through (or flyover) the specified tiles.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Return nodes matching the specified search patterns. The default pattern is the wildcard '*' which gets a list of all nodes on the device. More than one search pattern can be specified to find nodes based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the nodes associated with the specified tile:

```
get_nodes -of_objects [get_tiles CLBLM_R_X11Y158]
```

The following example returns the nodes downhill from the specified node:

```
get_nodes -downhill -of_objects [get_nodes LIOB33_SING_X0Y199/IOB_PADOUT0]
```

See Also

- [get_nodes](#)
- [get_site_pins](#)
- [get_tiles](#)
- [get_wires](#)
- [list_property](#)
- [report_property](#)

get_objects

Get a list of HDL objects in one or more HDL scopes as per the specified pattern.

Syntax

```
get_objects [-filter arg] [-recursive] [-r] [-regexp] [-nocase]
[-quiet] [-verbose] [patterns...]
```

Returns

Returns all the objects found given the specified pattern

Usage

Name	Description
[-filter]	Filter the results with the specified expression
[-recursive]	Searches recursively for objects
[-r]	Searches recursively for objects
[-regexp]	Search using regular expressions, search design objects from which to create wave objects by design object name. The application supplying the design objects determines how the match is to be performed. Items must be strings.
[-nocase]	Perform a case insensitive match (only used with regexp)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>patterns</i>]	Patterns to search for. Default is * where all HDL objects are returned

Categories

[Simulation](#)

Description

Returns a list of HDL objects matching the specified search pattern in one or more HDL scopes.

HDL objects include HDL signals, variables, or constants as defined in the Verilog or VHDL testbench and source files. An HDL signal includes Verilog wire or reg entities, and VHDL signals. Examples of HDL variables include Verilog real, realtime, time, and event. HDL constants include Verilog parameters and localparams, and VHDL generic and constants.

The HDL scope, or scope, is defined by a declarative region in the HDL code such as a module, function, task, process, or begin-end blocks in Verilog. VHDL scopes include entity/architecture definitions, block, function, procedure, and process blocks.

Arguments

-recursive | -r - (Optional) Apply the command to the current scope, and all sub-scopes of the current scope.

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add ".*" to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_cells** based on property values on the cells. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the HDL object, "NAME", "SCOPE" and "TYPE" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match HDL objects against the specified patterns. The default pattern is the wildcard '*' which returns all the children in the current scope. The search pattern can be defined in two ways:

- *patterns* - Specifies only the search pattern for the objects to get. This method returns all objects in the current scope (and any sub-scopes when **-recursive** is used).
- *scope/pattern* - Specifies the scope of interest, relative to the current scope, and the pattern for objects to locate. In this case, the specified *scope*, and any sub-scopes of it if **-recursive** is used, are identified starting from the current scope. Then all objects matching the search **pattern** are identified and returned.

Examples

The following example specifies the `current_scope`, then gets all HDL objects in that scope:

```
current_scope ./cpuEngine
get_objects
```

The following example returns the count of all objects in the current scope, and then returns the count of all objects in the current scope, and all sub-scopes of it:

```
llength [get_objects]
182
llength [get_objects -recursive ]
2182
```

The following example specifies the *scope/pattern* search pattern as discussed above. Notice that the `cpuEngine` scope and various sub-scopes of it are identified, then objects matching the **cl*** pattern in those scopes are returned:

```
get_objects filter {type == internal_signal} cpuEngine/cl* -recursive
/top/cpuEngine/clk_i
/top/cpuEngine/iwb_biu/clk
/top/cpuEngine/iwb_biu/clmode
/top/cpuEngine/or1200_cpu/clk
...
/top/cpuEngine/or1200_immu_top/or1200_immu_tlb/itlb_mr_ram/clk
```

Search the current scope, and all sub-scopes, for any internal signals whose names start with `cl` or `ma`:

```
get_objects filter {type == internal_signal} ma* cl* -recursive
```

See Also

- [current_scope](#)
- [list_property](#)
- [report_objects](#)
- [report_property](#)

get_package_pins

Get a list of package pins.

Syntax

```
get_package_pins [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

Returns

List of package pin objects

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-of_objects]	Get the list of package pin objects of these sites iobanks ports.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match list of package pin objects against patterns Default: *

Categories

[XDC](#), [Object](#)

Description

Gets a list of the pins on the selected package for the target device. The default command gets a list of all pins on the package.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regex - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regex** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regex.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regex** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_package_pins** based on property values on the pins. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the package pin object, "IS_CLK_CAPABLE", "IS_VREF" and "IS_GLOBAL_CLK" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects *arg* - (Optional) Get the package pins connected to the specified objects. Valid objects include sites, I/O Banks, or ports.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match pins against the specified patterns. The default pattern is the wildcard '*' which returns all pins on the package. More than one pattern can be specified to find multiple pins based on different search criteria.

Examples

The following example gets a list of all pins on the package of the target device:

```
get_package_pins
```

The following example gets the number of clock capable (CC) pins on the package:

```
llength [get_package_pins -filter {IS_CLK_CAPABLE==1}]
```

Note If there are no pins matching the pattern you will get a warning.

See Also

- [get_iobanks](#)
- [get_sites](#)
- [list_property](#)
- [report_property](#)

get_param

Get a parameter value.

Syntax

```
get_param [-quiet] [-verbose] name
```

Returns

Parameter value

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Parameter name

Categories

[PropertyAndParameter](#)

Description

This command gets the currently defined value for a specified tool parameter. These parameters are user-definable configuration settings that control various behaviors within the tool. Refer to **report_param** for a description of what each parameter configures or controls.

Arguments

name - (Required) The name of the parameter to get the value of. The list of user-definable parameters can be obtained with **list_param**. This command requires the full name of the desired parameter. It does not perform any pattern matching, and accepts only one parameter at a time.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the current value of the MaxThreads parameter used for multi-threaded processes:

```
get_param general.MaxThreads
```

See Also

- [list_param](#)
- [report_param](#)
- [reset_param](#)
- [set_param](#)

get_parts

Get a list of parts available in the software.

Syntax

```
get_parts [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns]
```

Returns

List of part objects

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching. (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match part names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when <code>-regexp</code> is specified.

Categories

Object

Description

Gets a list of parts in the current project that match a specified search pattern. The default command gets a list of all parts in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_parts** based on property values on the parts. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the part object, "DEVICE", "FAMILY" and "SPEED" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match parts against the specified patterns. The default pattern is the wildcard '*' which gets a list of all parts. More than one search pattern can be specified to find parts based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example gets a list of 7vx485t parts, speed grade -1:

```
get_parts -filter {DEVICE =~ xc7vx485t* && speed == -1}
```

The following example gets the number of 7 series and 6 series Virtex parts:

```
llength [get_parts -regexp {xc7v.* xc6v.*} -nocase]
```

Note If there are no parts matching the pattern, the tool will return a warning.

See Also

- [list_property](#)
- [report_property](#)

get_path_groups

Get a list of path groups in the current design.

Syntax

```
get_path_groups [-regexp] [-nocase] [-quiet] [-verbose] [patterns]
```

Returns

List of path groups

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match path group names against patterns Default: *

Categories

[XDC](#), [Object](#)

Description

Gets a list of timing path groups in the current project that match a specified search pattern. The default command gets a list of all path groups in the design.

Path groups are automatically created when a new clock is created in the design, containing all paths in that clock's domain. Path groups can also be manually created with the use of the **group_path** command.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match path groups against the specified patterns. The default pattern is the wildcard '*' which gets all path groups in the project.

Examples

The following example gets a list of all the path groups in the design.

```
get_path_groups
```

The following example gets all path groups with the string "Clk" somewhere in the name:

```
get_path_groups *Clk*
```

Note If no path groups match the pattern you will get a warning.

See Also

[group_path](#)

get_pblocks

Get a list of pblocks in the current design.

Syntax

```
get_pblocks [-regexp] [-nocase] [-filter arg] [-of_objects args]  
[-quiet] [-verbose] [patterns]
```

Returns

List of pblock objects

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get pblocks of these cells
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[<i>patterns</i>]</code>	Match pblock names against patterns Default: *

Categories

[Object](#), [Floorplan](#), [XDC](#)

Description

Gets a list of Pblocks defined in the current project that match a specific pattern. The default command gets a list of all Pblocks in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_pblocks** based on property values on the Pblocks. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the Pblock object, "NAME" and "GRID_RANGES" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects arg - (Optional) Get the Pblocks to which the specified cells are assigned.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match Pblocks against the specified patterns. The default pattern is the wildcard '*' which returns all Pblocks in the project.

Examples

The following example gets a list of all Pblocks in the current project:

```
get_pblocks
```

The following example gets a list of all Pblocks which do not have a Slice Range defined:

```
get_pblocks -filter {GRIDTYPES !~ SLICE}
```

The following example gets the Pblock assignments of the specified cell:

```
get_pblocks -of [get_cells CORE/BR_TOP/RLD67_MUX/REG_PMBIST_C1]
```

See Also

- [create_pblock](#)
- [get_cells](#)

get_pins

Get a list of pins in the current design.

Syntax

```
get_pins [-hsc arg] [-hierarchical] [-regexp] [-nocase] [-leaf]
[-filter arg] [-of_objects args] [-match_style arg] [-quiet]
[-verbose] [patterns]
```

Returns

List of pin objects

Usage

Name	Description
[-hsc]	Hierarchy separator Default: /
[-hierarchical]	Search level-by-level in current instance
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching (valid only when -regexp specified)
[-leaf]	Get leaf/global pins of nets with -of_objects
[-filter]	Filter list with expression
[-of_objects]	Get pins of these cells, nets, timing paths or clocks
[-match_style]	Style of pattern matching, valid values are ucf, sdc Default: sdc
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match pin names against patterns Default: *

Categories

[SDC](#), [XDC](#), [Object](#)

Description

Gets a list of pin objects in the current design that match a specified search pattern. The default command gets a list of all pins in the design.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-hsc *arg* - (Optional) The default hierarchy separator is '/'. Use this argument to specify a different hierarchy separator.

-hierarchical - (Optional) Get pins from all levels of the design hierarchy. Without this argument, the command will only get pins from the top of the design hierarchy. When using **-hierarchical**, the search pattern should not contain a hierarchy separator because the search pattern is applied at each level of the hierarchy, not to the full hierarchical cell name. For instance, searching for `U1/*` searches each level of the hierarchy for instances with `U1/` in the name. This may not return the intended results. See **get_cells** for examples of **-hierarchical** searches.

Note When used with **-regexpr**, the specified search string is matched against the full hierarchical name, and the `U1/*` search pattern will work as intended

-regexpr - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `".*"` to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexpr** only.

-leaf - (Optional) Include leaf pins, from primitive or black box cells, for the objects specified with the **-of_object** argument.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by `get_pins` based on property values on the pins. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the pins object, "PARENT" and "TYPE" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (`==` and `!=`), and "contains" and "not-contains" (`=~` and `!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by AND and OR (`&&` and `||`). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects *arg* - (Optional) Get the pins connected to the specified cell, port, or clock.

Note **-of_objects** cannot be used with **-hierarchy** or with a search *pattern*

-match_style [sdc | ucf] - (Optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match pins against the specified patterns. The default pattern is the wildcard '*' which gets a list of all pins in the project. More than one pattern can be specified to find multiple pins based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example gets a list of pins attached to the specified cell:

```
get_pins -of_objects [lindex [get_cells] 1]
```

Note If there are no pins matching the pattern, the tool will return a warning.

See Also

- [list_property](#)
- [report_property](#)

get_pips

Get a list of programmable interconnect points (pips) on the current device.

Syntax

```
get_pips [-regexp] [-nocase] [-filter arg] [-of_objects args] [-uphill]
[-downhill] [-from args] [-to args] [-quiet] [-verbose] [patterns]
```

Returns

Pips

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-of_objects]	Get the pips of these sites, tiles, wires, nodes, pips, or nets.
[-uphill]	Get the pips uphill from the provided wire or pip.
[-downhill]	Get the pips downhill from the provided wire or pip.
[-from]	-from Return the ordered list of pips beginning at this pip or site pin. May be used in combination with uphill. Default is downhill. -all is implied.
[-to]	-to Return the ordered list of pips ending at this wire or site pin. May be used in combination with uphill. Default is downhill. -all is implied.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match pips against patterns Default: *

Categories

Object

Description

Programmable interconnect points, or PIPs, provide the physical routing paths on the device used to connect logic networks. This command returns a list of PIPs on the device that match a specified search pattern. The command requires a design to be open.

The default command gets a list of all PIPs on the device. However, this is not a recommended use of the command due to the number of pips on a device. You should specify the **-of_objects** argument to limit the number of pips returned.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `".*"` to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_pips** based on property values on the PIPs. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the PIP object, "IS_DIRECTIONAL" and "FROM_PIN" are two of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (`==` and `!=`), and "contains" and "not-contains" (`=~` and `!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by AND and OR (`&&` and `||`). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) Return the PIPs of the specified site, tile, or wire objects.

Note Xilinx recommends that you always use the **-of_objects** argument to limit the runtime and memory used by the **get_pips** command. The number of programmable interconnect points returned can be considerable. The **-of_objects** option cannot be used with a search *pattern*

-uphill - (Optional) Return the PIPs uphill of the specified wire or PIPs. Uphill PIPs precede the specified object in the logic network.

-downhill - (Optional) Return the PIPs downhill of the specified wire or PIPs. Downhill PIPs follow the specified object in the logic network.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Return PIPs matching the specified search patterns. The default pattern is the wildcard '*' which gets a list of all PIPs on the device. More than one search pattern can be specified to find PIPs based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the PIPs associated with the specified tile:

```
get_pips -of_object [get_tiles DSP_R_X9Y75]
```

See Also

- [get_sites](#)
- [get_tiles](#)
- [get_wires](#)
- [list_property](#)
- [report_property](#)

get_ports

Get a list of ports in the current design.

Syntax

```
get_ports [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-match_style arg] [-quiet] [-verbose] [patterns]
```

Returns

List of port objects

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get ports of these nets, instances, sites, clocks, timing paths, io standards, io banks, package pins
<code>[-match_style]</code>	Style of pattern matching, valid values are <code>ucf</code> , <code>sdcl</code> Default: <code>sdcl</code>
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match port names against patterns Default: <code>*</code>

Categories

[SDC](#), [XDC](#), [Object](#)

Description

Gets a list of port objects in the current design that match a specified search pattern. The default command gets a list of all ports in the design.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by `get_ports` based on property values on the ports. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "ports" object, "PARENT" and "TYPE" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects arg - (Optional) Get the ports connected to the specified cell, net, clock, or timing path objects.

Note **-of_objects** cannot be used with a search *pattern*

-match_style [sdc | ucf] - (Optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match ports against the specified patterns. The default pattern is the wildcard '*' which gets a list of all ports in the project. More than one pattern can be specified to find multiple ports based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example gets a list of pins attached to the specified cell:

```
get_ports -of_objects [lindex [get_cells] 1]
```

Note If there are no ports matching the pattern, the tool will return a warning.

See Also

- [list_property](#)
- [report_property](#)

get_projects

Get a list of projects.

Syntax

```
get_projects [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns]
```

Returns

List of project objects

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match project names against patterns Default: *

Categories

[Object](#), [Project](#)

Description

Gets a list of open projects that match the specified search pattern. The default gets a list of all open projects.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_projects** based on property values on the projects. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "projects" object, "NAME", "DIRECTORY" and "TARGET_LANGUAGE" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match projects against the specified patterns. The default pattern is the wildcard '*' which gets a list of all parts. More than one pattern can be specified to find multiple projects based on different search criteria.

Examples

The following example gets a list of all open projects.

```
get_projects
```


The following example sets a variable called `project_found` to the length of the list of projects returned by `get_projects`, then prints either that projects were found or were not found as appropriate:

```
set project_found [llength [get_projects ISC*] ]  
if {$project_found > 0} {puts "Project Found."} else {puts "No Projects Found."}
```

Note If there are no projects matching the pattern you will get a warning.

See Also

- [create_project](#)
- [current_project](#)
- [open_project](#)

get_property

Get properties of object.

Syntax

```
get_property [-quiet] [-verbose] name object
```

Returns

Property value

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of property whose value is to be retrieved
<i>object</i>	Object to query for properties

Categories

[Object](#), [PropertyAndParameter](#)

Description

Gets the current value of the named property from the specified object. If the property is not currently assigned to the object, or is assigned without a value, then the **get_property** command returns nothing.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the property to be returned. The name is not case sensitive.

object - (Required) The object to query.

Examples

The following example gets the NAME property from the specified cell:

```
get_property NAME [lindex [get_cells] 3]
```

See Also

- [create_property](#)
- [get_cells](#)
- [list_property](#)
- [list_property_value](#)
- [report_property](#)
- [reset_property](#)
- [set_property](#)

get_runs

Get a list of runs.

Syntax

```
get_runs [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]  
[patterns]
```

Returns

List of run objects

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match run names against patterns Default: *

Categories

[Object](#), [Project](#)

Description

Gets a list of synthesis and implementation runs in the current project that match a specified search pattern. The default command gets a list of all runs defined in the project.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by `get_runs` based on property values on the runs. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the runs object, "CONSTRSET", "IS_IMPLEMENTATION", "IS_SYNTHESIS", and "FLOW" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match run names against the specified patterns. The default pattern is the wildcard '*' which gets a list of all defined runs in the project. More than one pattern can be specified to find multiple runs based on different search criteria.

Examples

The following example gets a list of all incomplete runs in the current project:

```
get_runs -filter {PROGRESS < 100}
```

The following example gets a list of runs matching the specified pattern:

```
get_runs imp*
```

Note If there are no runs matching the pattern you will get a warning.

See Also

- [create_run](#)
- [current_run](#)
- [report_property](#)

get_scopes

Get a list of children HDL scopes of a scope.

Syntax

```
get_scopes [-filter arg] [-regexp] [-nocase] [-recursive] [-r] [-quiet]
[-verbose] [patterns...]
```

Returns

Returns HDL scope objects from the given arguments

Usage

Name	Description
[-filter]	filter filter_expression: Filter the results list with the specified expression. The filter argument filters the list of HDL scopes returned by get_scopes based on property values on the HDL scope Tcl object. You can find out what properties are on a Tcl object with the report_property or report_property commands. In the case of the HDL scopes, "process", "block" and "task" are some of the property values that can be used to filter results.
[-regexp]	regexp: using regular expressions, search design objects from which to create wave objects by design object name. The application supplying the design objects determines how the match is to be performed. items must be strings.
[-nocase]	nocase: Only when regexp is used, perform a case insensitive match
[-recursive]	Recursive: Applicable only when a glob or regular expression pattern is used. Descend recursively into children scopes and apply the pattern
[-r]	r: Recursive: Applicable only when a glob or regular expression pattern is used. Descend recursively into children scopes and apply the pattern
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>patterns</i>]	Default for patterns is * i.e. all children scopes are returned

Categories

Simulation

get_selected_objects

Get selected objects.

Syntax

```
get_selected_objects [-primary] [-quiet] [-verbose]
```

Returns

List of selected objects

Usage

Name	Description
[-primary]	Do not include objects that were selected due to selection rules
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Object](#), [GUIControl](#)

Description

Gets the objects currently selected in the Vivado IDE by the **select_objects** command. Can get the primary selected object and any secondary selected objects as well.

Note This Tcl command works only when Vivado is run in GUI mode

Primary objects are directly selected, while secondary objects are selected based on the selection rules currently defined by the **Tools > Options** command. Refer to the *Vivado Design Suite User Guide: Using the IDE (UG893)* for more information on setting selection rules.

Arguments

-primary - (Optional) Indicates that only the primary selected object or objects should be returned; not secondary objects. As a default **get_selected_objects** will return all currently selected objects.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the properties of all currently selected objects, both primary and secondary:

```
report_property [get_selected_objects]
```

See Also

[select_objects](#)

get_site_pins

Get a list of site_pins.

Syntax

```
get_site_pins [-of_objects args] [-regexp] [-nocase] [-filter arg]
[-quiet] [-verbose] [patterns]
```

Returns

Site_pins

Usage

Name	Description
[-of_objects]	Get 'site_pin' objects of these types: 'site xdef_site node pin net'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'site_pin' objects against patterns. Default: *

Categories

[Object](#)

Description

Returns a list of site pins of the specified site, BELs, or logical cell pin objects in an open design. This command requires the use of the **-of_objects** argument.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_site_pins** based on property values on the site pins. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the site pin object, "IS_CLOCK", "IS_DATA" and "IS_PART_OF_BUS" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) Return the site pins of specified site, routed sites, BELs, or logical cell pin objects.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the site_pins of the specified BELs:

```
get_site_pins -of_objects [get_bels SLICE_X21Y92/B5FF]
CE CK D SR Q
get_site_pins -of_objects [get_bels SLICE_X21Y92/A5LUT]
A1 A2 A3 A4 A5 O5
get_site_pins -of_objects [get_bels SLICE_X21Y92/CARRY4_CMUX]
O 1 S0 OUT
```

The following example returns the site_pins associated with the specified site:

```
get_site_pins -of_objects [get_sites SLICE_X21Y92]  
A1 A2 A3 A4 A5 A6 AX B1 B2 B3 B4 B5 B6 BX C1 C2 C3 C4 C5 C6 CE CIN CLK CX \  
D1 D2 D3 D4 D5 D6 DX SR A AMUX AQ B BMUX BQ C CMUX COUT CQ D DMUX DQ
```

See Also

- [get_bels](#)
- [get_sites](#)
- [list_property](#)
- [report_property](#)

get_site_pips

Get a list of site_pips from the given object.

Syntax

```
get_site_pips [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

Returns

Site_pips

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching. (valid only when -regexp specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get the site_pips of these sites.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match site_pips against patterns Default: *

Categories

[Object](#)

Description

Programmable interconnect points, or PIPs, provide the physical routing paths on the device used to connect logic networks. This command returns a list of PIPs on specified sites that match a specified search pattern. The command requires a design to be open.

This command requires the use of the -of_objects option to specify the sites to return PIPs from.

Note To improve memory and performance, the get_* commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_site_pips** based on property values on the PIPs. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the PIP object, "IS_DIRECTIONAL" and "FROM_PIN" are two of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) This option can be used with the **get_bels** command to return the pins of specified BELs.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match BEL pins against the specified patterns. The default pattern is the wildcard "*" which gets a list of all BEL pins on the device. More than one search pattern can be specified to find pins based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the pins of the specified BELs associated with the specified range of sites on the device:

```
get_site_pins -of_objects [get_sites SLICE_X21Y92]
```

The following example returns the clock enable (CE) pins of all BELs on the device:

```
get_bel_pins *CE
```

See Also

- [get_bels](#)
- [get_sites](#)
- [list_property](#)
- [report_property](#)

get_sites

Get a list of Sites.

Syntax

```
get_sites [-regexp] [-filter arg] [-range args] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

Returns

List of site objects

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-range]	Match site names which fall into the range. Range is defined by exactly two site names.
[-of_objects]	Get the sites of the objects passed in here.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match site names against patterns. Bonded sites will also match on package pin names. Default: *

Categories

[XDC](#), [Object](#)

Description

Gets a list of sites on the target device that match a specified search pattern. The default command gets a list of all sites on the target device.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by `get_sites` based on property values on the sites. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the site object, "SITE_TYPE", "IS_OCCUPIED", "NUM_INPUTS", and "NUM_OUTPUTS" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-range *arg* - (Optional) Get all the sites that fall into a specified range. The range of sites must be specified with two site values, of the same SITE_TYPE, such as {SLICE_X2Y12 SLICE_X3Y15}. The SITE_TYPE of a site can be determined by the **report_property** command.

Note Specifying a range with two different types will result in an error

-of_objects *arg* - (Optional) Get sites from the specified object or objects. Valid objects include: tiles, BELs, pins, package pins, ports, Pblocks, I/O Banks, cells, and clock_regions.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match sites against the specified patterns. The default pattern is the wildcard '*' which gets a list of all sites on the target device.

Examples

The following example gets a list of all sites available on the target device:

```
get_sites
```

The following example gets the number of unoccupied sites on the device:

```
llength [get_sites -filter {IS_OCCUPIED==0}]
```

Note If no sites match the pattern you will get a warning.

The following example gets all of the sites on the device, and returns the unique SITE_TYPES:

```
set sites [get_sites]
set type {}
foreach x $sites {
    set prop [get_property SITE_TYPE $x]
    if { [lsearch -exact $type $prop] == -1 } {
        lappend type $prop
    }
}
foreach y $type {
    puts "SITE_TYPE: $y"
}
```

The following example shows three different forms for specifying the range of sites to return:

```
get_sites -range {SLICE_X0Y0 SLICE_X1Y1}
SLICE_X0Y0 SLICE_X0Y1 SLICE_X1Y0 SLICE_X1Y1
get_sites -range SLICE_X0Y0 -range SLICE_X1Y1
SLICE_X0Y0 SLICE_X0Y1 SLICE_X1Y0 SLICE_X1Y1
get_sites -range {SLICE_X0Y0:SLICE_X1Y1}
SLICE_X0Y0 SLICE_X0Y1 SLICE_X1Y0 SLICE_X1Y1
```

See Also

- [get_cells](#)
- [get_property](#)
- [list_property](#)
- [report_property](#)

get_slrs

Get a list of Super Logic Regions (SLRs).

Syntax

```
get_slrs [-regexp] [-filter arg] [-no_case] [-of_objects args] [-quiet]  
[-verbose] [patterns]
```

Returns

List of SLRs

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-no_case]	Perform case-insensitive matching. (valid only when -regexp specified)
[-of_objects]	Get the SLRs of the objects passed in here.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[patterns]</i>	Match SLR names against patterns. Default: *

Categories

[Object](#)

get_tiles

Get a list of tiles.

Syntax

```
get_tiles [-regexp] [-nocase] [-filter arg] [-of_objects args] [-quiet]
[-verbose] [patterns]
```

Returns

Tiles

Usage

Name	Description
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching. (valid only when <code>-regexp</code> specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get the tiles of these sites, bels, site_pins, bel_pins, nodes, wires, pips, nets, clock_regions.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[patterns]</code>	Match tiles against patterns Default: *

Categories

[Object](#)

Description

This command returns a list of tiles on the device in an open design. The default command gets a list of all tiles on the device.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_tiles** based on property values on the tile objects. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the tile object, "NUM_ARCS", "NUM_SITES", and "IS_GT_SITE_TILE" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) Can be used to return the tiles associated with specified sites, BELs, site_pins, nodes, wires, and PIPs.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Match tiles against the specified patterns. The default pattern is the wildcard '*' which gets a list of all tiles on the device. More than one search pattern can be specified to find tiles based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the total number of tiles where the number of timing arcs is greater than 100 and 150 respectively:

```
llength [get_tiles -filter {NUM_ARCS>100} ]  
13468  
llength [get_tiles -filter {NUM_ARCS>150} ]  
11691
```

See Also

- [get_bels](#)
- [get_nodes](#)
- [get_pips](#)
- [get_site_pins](#)
- [get_sites](#)
- [get_wires](#)
- [list_property](#)
- [report_property](#)

get_timing_arcs

Get a list of timing arcs.

Syntax

```
get_timing_arcs [-from args] [-to args] [-filter arg]
[-of_objects args] [-quiet] [-verbose]
```

Returns

List of timing arc objects

Usage

Name	Description
<code>[-from]</code>	List of pin or ports
<code>[-to]</code>	List of pin or ports
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get timing arcs for these cells
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#), [Object](#), [Timing](#)

Description

Gets a list of timing arcs for the specified objects. You can filter the timing arcs according to specified properties.

Timing arcs are a part of a timing path. A timing arc can be a wire between two pins, or can be the internal path of a logic instance between an input pin and output pin, or clock input and data output pins.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-from *args* - (Optional) The starting points of the timing arcs to be returned. Ports, pins, or cells can be specified as startpoints. You can also specify a clock object, and all timing arcs with the specified startpoints clocked by the named clock will be returned.

-to *args* - (Optional) The endpoints or destination objects of timing arcs to be returned. Ports, pins, and cell objects can be specified as endpoints. A clock object can also be specified, in which case the timing arcs with endpoints clocked by the named clock are returned.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_timing_arcs** based on property values on the objects. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the "timing arc" object, "FROM_PIN", "TO_PIN" and "LIB_CELL" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects *args* - (Optional) Get timing arcs from the Specified cell objects. If a cell is specified, all cell_arcs of that cell are returned.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the timing arc from the output pin of the specified buffer:

```
report_property -all [get_timing_arcs -of_objects [get_cells go_IBUF_inst]]
```

The following example returns the timing arcs of the specified cell:

```
get_timing_arcs -of_objects [get_cells count_reg[6]]
{count_reg[6]/C --> count_reg[6]/Q [Reg Clk to Q] }
{count_reg[6]/C --> count_reg[6]/D [setup] }
{count_reg[6]/C --> count_reg[6]/D [hold] }
{count_reg[6]/C --> count_reg[6]/CLR [recovery] }
{count_reg[6]/C --> count_reg[6]/CE [hold] }
{count_reg[6]/C --> count_reg[6]/CLR [removal] }
{count_reg[6]/C --> count_reg[6]/CE [setup] }
{count_reg[6]/CLR --> count_reg[6]/Q [Reg Set/Clr] }
```


See Also

- [report_timing](#)
- [set_msg_limit](#)

get_timing_paths

Get timing paths.

Syntax

```
get_timing_paths [-from args] [-rise_from args] [-fall_from args]
[-to args] [-rise_to args] [-fall_to args] [-through args]
[-rise_through args] [-fall_through args] [-delay_type arg]
[-setup] [-hold] [-max_paths arg] [-nworst arg] [-unique_pins]
[-slack_lesser_than arg] [-slack_greater_than arg] [-group args]
[-no_report_unconstrained] [-user_ignored] [-sort_by arg] [-filter arg]
[-regexp] [-nocase] [-match_style arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-from]	From pins, ports, cells or clocks
[-rise_from]	Rising from pins, ports, cells or clocks
[-fall_from]	Falling from pins, ports, cells or clocks
[-to]	To pins, ports, cells or clocks
[-rise_to]	Rising to pins, ports, cells or clocks
[-fall_to]	Falling to pins, ports, cells or clocks
[-through]	Through pins, ports, cells or nets
[-rise_through]	Rising through pins, ports, cells or nets
[-fall_through]	Falling through pins, ports, cells or nets
[-delay_type]	Type of path delay: Values: max, min, min_max, max_rise, max_fall, min_rise, min_fall Default: max
[-setup]	Get max delay timing paths (equivalent to -delay_type max)
[-hold]	Get min delay timing paths (equivalent to -delay_type min)
[-max_paths]	Maximum number of paths to return: Value >=1 Default: 1
[-nworst]	List N worst paths to endpoint: Value >=1 Default: 1
[-unique_pins]	for each unique set of pins, show at most 1 path per path group
[-slack_lesser_than]	Include paths with slack less than this Default: 1e+30
[-slack_greater_than]	Include paths with slack greater than this Default: -1e+30
[-group]	Limit paths in this group(s)

Name	Description
[-no_report_unconstrained]	Do not get unconstrained paths
[-user_ignored]	only report paths which have infinite slack because of set_false_path or set_clock_groups timing constraints
[-sort_by]	Sorting order of paths: Values: group, slack Default: slack
[-filter]	Filter list with expression
[-regex]	Patterns specified in filter are full regular expressions
[-nocase]	Perform case-insensitive matching for patterns specified in filter (valid only when -regex specified)
[-match_style]	Style of pattern matching, valid values are ucf, sdc Default: ucf
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

SDC, Object, Timing

Description

Gets timing path objects that meet the specified criteria. This command can be used to predefine timing paths to pass to the **report_timing** command for instance. Another usage of this command is to create custom reporting and analysis.

The **get_timing_paths** command is very similar to the **report_timing** command. However, **get_timing_paths** returns timing path objects which can be queried for properties, or passed to other Tcl commands for processing, where **report_timing** returns a file or a string.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-from *args* - (Optional) Defines the starting points of the timing paths to be analyzed. Ports, pins, or cells can be specified as timing path startpoints. You can also specify a clock object, and all startpoints clocked by the named clock will be analyzed.

-rise_from *args* - (Optional) Similar to the **-from** option, but only the rising edge of signals coming from the startpoints are considered for timing analysis. If a clock object is specified, only the paths launched by the rising edge of the clock are considered as startpoints.

-fall_from *args* - (Optional) Similar to the **-from** option, but only the falling edge of signals coming from the startpoints are considered for timing analysis. If a clock object is specified, only the paths launched by the falling edge of the clock are considered as startpoints.

-to args - (Optional) Specifies the endpoints, or destination objects of timing paths to be analyzed. Ports, pins, and cell objects can be specified as endpoints. A clock object can also be specified, in which case endpoints clocked by the named clock are analyzed.

-rise_to args - (Optional) Similar to the **-to** option, but only the rising edge of signals going to the endpoints is considered for timing analysis. If a clock object is specified, only the paths captured by the rising edge of the named clock are considered as endpoints.

-fall_to args - (Optional) Similar to the **-to** option, but only the falling edge of signals going to the endpoints is considered for timing analysis. If a clock object is specified, only the paths captured by the falling edge of the named clock are considered as endpoints.

-through args - (Optional) Specifies that only paths through the specified pins, cell instance, or nets are considered during timing analysis. You can specify individual **-through** (or **-rise_through** and **-fall_through**) points in sequence to define a specific path through the design for analysis. The order of the specified through points is important to define a specific path. You can also specify through points with multiple objects, in which case the timing path can pass through any of the specified through objects.

-rise_through args - (Optional) Similar to the **-through** option, but timing analysis is only performed on paths with a rising transition at the specified objects.

-fall_through args - (Optional) Similar to the **-through** option, but timing analysis is only performed on paths with a falling transition at the specified objects.

-delay_type arg - (Optional) Specifies the type of delay to analyze when running the timing report. The valid values are min, max, min_max, max_rise, max_fall, min_rise, min_fall. The default setting for **-delay_type** is max.

-setup - (Optional) Check for setup violations. This is the same as specifying **-delay_type max**.

-hold - (Optional) Check for hold violations. This is the same as specifying **-delay_type min**.

Note **-setup** and **-hold** can be specified together, which is the same as specifying **-delay_type min_max**.

-max_paths arg - (Optional) The maximum number of paths to output when sorted by slack; or the maximum number of paths per path group when sorted by group, as specified by **-sort_by**. This is specified as a value greater than or equal to 1. The default value is 1, returning the single worst timing path, or the worst path per group.

-nworst arg - (Optional) The number of timing paths to show to each endpoint. The timing report will report the N worst paths based on the specified value. This is specified as a value greater than or equal to 1. The default setting is 1.

-slack_greater_than arg - (Optional) Report timing on paths with a calculated slack value greater than the specified value. Used with **-slack_lesser_than** to provide a range of slack values of specific interest.

-slack_lesser_than arg - (Optional) Report timing on paths with a calculated slack value less than the specified value. Used with **-slack_greater_than** to provide a range of slack values of specific interest.

-group args - (Optional) Report timing for paths in the specified path groups.

-no_report_unconstrained - (Optional) Do not report timing on unconstrained paths.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_timing_paths** based on property values on the objects. You can find the properties on an object with the **report_property** or **list_property** commands. In the case of the timing path object, "DATAPATH_DELAY", "ENDPOINT_PIN" and "ENDPOINT_CLOCK" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

The following example gets the first 100 most critical timing paths objects and returns only those from the path group clk_tx_clk_core_1:

```
get_timing_paths -max_paths 100 -filter {GROUP == clk_tx_clk_core_1}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-match_style [sdc | ucf] - (Optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is UCF.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example gets the five worst timing paths from the specified endpoint, and reports all the properties of the fourth timing path in the list:

```
report_property -all [lindex [get_timing_paths -to [get_ports led_pins[*]]\n\nworst 5] 3]
```

The following example defines a procedure called `custom_report`, then reports the 100 worst paths from the `clk_tx_clk_core_1` path group using that proc:

```
proc custom_report { listOfPaths } {
    puts [format {%-40s %-40s %-20s %-20s %7s} "Startpoint" "Endpoint" "Launch Clock" "Capture Clock" "Slack"]
    puts [string repeat "-" 140]
    foreach path $listOfPaths {
        set startpoint [get_property STARTPOINT_PIN $path]
        set startclock [get_property STARTPOINT_CLOCK $path]
        set endpoint [get_property ENDPOINT_PIN $path]
        set endclock [get_property ENDPOINT_CLOCK $path]
        set slack [get_property SLACK $path]
        puts [format {%-40s %-40s %-20s %-20s %7s} $startpoint $endpoint $startclock $endclock $slack]
    }
}
set paths [get_timing_paths -group clk_tx_clk_core_1 -max_paths 100]\
custom_report $paths
```

The following example illustrates how timing path objects can be used with the **report_timing** command:

```
set paths [get_timing_paths -group clk_tx_clk_core_1 -max_paths 100]
report_timing -of_objects $paths
```

Which is the equivalent of:

```
report_timing -group clk_tx_clk_core_1 -max_paths 100
```

See Also

- [report_property](#)
- [report_timing](#)

get_value

Get current value of the selected HDL object (variable, signal, wire, reg).

Syntax

```
get_value [-radix arg] [-quiet] [-verbose] hdl_object
```

Returns

Returns a string representation of value of a *hdl_object*

Usage

Name	Description
[-radix]	radix specifies the radix to use for printing the values of the <i>hdl_objects</i> . Allowed values are: default, dec, bin, oct, hex, unsigned, ascii
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hdl_object</i>	The <i>hdl_object</i> to retrieve the current value

Categories

[Simulation](#)

Description

Get the value of a single HDL object at the current simulation run time.

Tip Use the **report_values** command to return the values of more than one HDL objects.

HDL objects include HDL signals, variables, or constants as defined in the Verilog or VHDL testbench and source files. An HDL signal includes Verilog wire or reg entities, and VHDL signals. Examples of HDL variables include Verilog real, realtime, time, and event.

HDL constants include Verilog parameters and localparams, and VHDL generic and constants. The HDL scope, or scope, is defined by a declarative region in the HDL code such as a module, function, task, process, or begin-end blocks in Verilog. VHDL scopes include entity/architecture definitions, block, function, procedure, and process blocks.

Arguments

-radix *arg* - (Optional) Specifies the radix to use when returning the value of the specified object. Allowed values are: **default**, **dec**, **bin**, **oct**, **hex**, **unsigned**, and **ascii**.

Note The radix **dec** indicates a signed decimal. Specify the radix **unsigned** when dealing with unsigned data

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

hdl_object - (Required) Specifies a single HDL object to get the value of. The object can be specified by name, or can be returned as an object from the **get_objects** command.

Examples

The following example gets the value of the sysClk signal:

```
get_value sysClk
Z
```

This example shows the difference between the **bin**, **dec**, and **unsigned** radix on the value returned from the specified bus:

```
get_value -radix bin /test/bench_VStatus_pad_0_i[7:0]
10100101
get_value -radix unsigned /test/bench_VStatus_pad_0_i[7:0]
165
get_value -radix dec /test/bench_VStatus_pad_0_i[7:0]
-91
```

See Also

- [current_time](#)
- [get_objects](#)
- [set_value](#)
- [report_values](#)

get_wave_configs

Gets the wave configs that match the given options.

Syntax

```
get_wave_configs [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]
[patterns...]
```

Returns

Wave configs that match the given options

Usage

Name	Description
[-regexp]	regexp: using regular expressions, search design objects from which to create wave objects by design object name. The application supplying the design objects determines how the match is to be performed. items must be strings.
[-nocase]	nocase: Only when regexp is used, perform a case insensitive match
[-filter]	filter args: Filter the results list with the specified expression. The filter argument filters the list of wave configuration objects based on property values of the wave configuration objects. You can find out what properties are on an object with the report_property or list_property commands. In the case of the wave configuration objects, "has_time", and "needs_save" are some of the properties that can be used to filter results. The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and between filter patterns.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Searches all wave configurations for the ones whose name property matches patterns

Categories

Waveform

get_wires

Get a list of wires.

Syntax

```
get_wires [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-uphill] [-downhill] [-from args] [-to args] [-quiet] [-verbose]
[patterns]
```

Returns

Wires

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
[-of_objects]	Get the wires of these tiles, nodes, pips, or nets.
[-uphill]	Get the wires uphill from the provided pip.
[-downhill]	Get the wires downhill from the provided pip.
[-from]	-from Return the ordered list of wires beginning at this pip or site pin. May be used in combination with uphill. Default is downhill. -all is implied.
[-to]	-to Return the ordered list of wires ending at this wire or site pin. May be used in combination with uphill. Default is downhill. -all is implied.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match wires against patterns Default: *

Categories

Object

Description

Returns a list of wires in the design that match a specified search pattern in an open design.

The default command gets a list of all wires in the design.

Note To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using **lappend** for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add `".*"` to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter args - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get_wires** based on property values on the wires. You can find the properties on an object with the **report_property** or **list_property** commands. Any property/value pair can be used as a filter. In the case of the wire object, "NAME", "NUM_DOWNHILL_PIPS" and "NUM_UPHILL_PIPS" are some of the properties that can be used to filter results.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (`==` and `!=`), and "contains" and "not-contains" (`=~` and `!~`). Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. Multiple filter expressions can be joined by AND and OR (`&&` and `||`). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-of_objects args - (Optional) Return the wires of the specified nodes, PIPs, or tiles.

Note **-of_objects** cannot be used with a search *pattern*

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Return wires matching the specified search patterns. The default pattern is the wildcard '*' which gets a list of all wires in the design. More than one search pattern can be specified to find wires based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example returns the wires associated with the specified tile:

```
get_wires -of_objects [get_tiles IO_INT_INTERFACE_L_X0Y198]
```

See Also

- [get_nodes](#)
- [get_pips](#)
- [get_tiles](#)
- [list_property](#)
- [report_property](#)

group_bd_cells

Create a hierarchical cell, and then move the group of cells into the hierarchy cell. The connections between these cells are maintained; the connections between these cells and other cells are maintained through crossing hierarchy cell.

Syntax

```
group_bd_cells [-prefix arg] [-quiet] [-verbose] [target_cell_name ]  
[cells...]
```

Returns

0 if success

Usage

Name	Description
[-prefix]	Prefix name to add to cells
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>target_cell_name</i>]	Target cell
[<i>cells</i>]	Match engine names against cell names Default: *

Categories

[IPIntegrator](#)

group_path

Groups paths for cost function calculations.

Syntax

```
group_path [-name arg] [-from args] [-to args] [-through args]  
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-name]	Name of the group
[-from]	Filter by paths starting at these path startpoints
[-to]	Filter by paths terminating at these path endpoints
[-through]	Consider paths through pins, cells or nets
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

Groups a set of paths for cost function calculations, primarily for timing analysis. Timing paths can be specified generally as from a startpoint, or to an endpoint, or as from-through-to specific points. Once a path group has been created, some timing analysis can be performed against it with the `report_timing` command.

Note This command operates silently and does not return direct feedback of its operation.

The path groups currently defined in a design can be found by using the **get_path_groups** command.

Arguments

-name *<arg>* - (Optional) Specifies the name of the path group. If the path group name already exists, the specified paths will be added to the existing group.

- from** <args> - (Optional) Include paths starting at the specified startpoints. The startpoints can be specified as pins, ports, or clocks.
- to** <path_names> - (Optional) Include all paths to the specified endpoints. Endpoints can be specified as pins, ports, or clocks.
- through** <element_names> - (Optional) Include paths routed through the specified pins, cells, or nets.
- quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.
- verbose** - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example creates a group named `signal_grp` to the specified registers endpoints matching `*signal*reg/D`, and then reports timing on the specified group:

```
group_path -to [get_pins *signal*reg/D -hierarchical] -name signal_grp
report_timing -group signal_grp
```

The path group `signal_grp` is also returned by the `get_path_groups` command:

```
get_path_groups
signal_grp
```

See Also

- [get_path_groups](#)
- [report_timing](#)

help

Display help for one or more topics.

Syntax

```
help [-category arg] [-args] [-syntax] [-long] [-prop arg]
[-class arg] [-quiet] [-verbose] [pattern_or_object]
```

Returns

Nothing

Usage

Name	Description
[-category]	Search for topics in the specified category
[-args]	Display arguments description
[-syntax]	Display syntax description
[-long]	Display long help description
[-prop]	Display property help for matching property names Default: *
[-class]	Display object type help
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[pattern_or_object]</i>	Display help for topics that match the specified pattern Default: *

Categories

[Project](#)

Description

Returns a long description of the specified Tcl command; or a list of available Xilinx Tcl command categories; or a list of commands matching a specific pattern.

The default **help** command without any arguments returns a list of Tcl command categories that can be further explored. Command categories are groups of commands performing a specific function, like File I/O commands for instance.

Available options for the **help** command can return just the command syntax for a quick reminder of how the command should be structured; the command syntax and a brief description of each argument; or the long form of the command with more detailed descriptions and examples of the command.

To limit the memory usage of the Vivado Design Suite, some features of the tool are only loaded into memory when that feature set is used. To access the complete list of Tcl commands and help text associated with a given feature, you must load the feature into memory using the **load_features** command.

The help command can also return any available information related to various properties assignable to design objects. Use the **-prop** and **-class** options to return help information for properties.

This command returns the specified help text, or an error.

Arguments

-category *arg* - (Optional) Get a list of the commands grouped under the specified command category.

-syntax - (Optional) Returns only the syntax line for the command as a quick reminder of the proper form for using the command.

-args - (Optional) Get abbreviated help text for the specified command. The default is to return the extended help for the specified command. Use this argument to keep it brief.

-long - (Optional) Returns the extended help description for the command, including the syntax, a brief description of the arguments, and a more detailed description of the command with examples. This is the default setting for the help command.

-prop *arg* - (Optional) Return information related to a specific property of a design object or class of design objects. This option requires the addition of **-class**, or the specification of a single design object.

-class *arg* - (Optional) Return information related to the properties of a specified class of objects.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pattern - (Optional) Returns information related to the specified command, or a list of commands that match the specified pattern.

Note A design object must be specified when used with **-prop** or **-class** to return information about properties.

Examples

The following example returns a list of Xilinx Tcl command categories:

```
help
```

This example loads the simulator feature of the Vivado Design Suite, and then returns a list of Tcl commands in the simulation and waveform categories:

```
load_features simulator
help -category simulation
help -category waveform
```

Returns a list of all commands matching the specified search pattern:

```
help *file*
```

This list can be used to quickly locate a command for a specific purpose, such as **remove_files** or **delete_files**.

The following help command returns a long description of the **remove_files** command and its arguments:

```
help remove_files
```

Note You can also use the **-args** option to get a brief description of the command.

This example defines a procedure called `short`, and returns the **-args** form of help for the specified command:

```
proc short cmdName {help -args $cmdName}
```

Note You can add this procedure to your `init.tcl` file to load this command every time the tool is launched. Refer to *Chapter 1, Introduction of the Vivado Design Suite Tcl Command Reference (UG835)* for more information on the `init.tcl` file

The following examples show how to obtain help for properties of design objects, or a class of design objects:

```
help -prop NAME -class cell  
help -prop NAME [get_cell cpuEngine]
```

Note In the preceding example, the first command returns general information related to the `NAME` property, while the second command also returns the value of the `NAME` property on the specified design object

See Also

- [list_features](#)
- [list_property](#)
- [load_features](#)
- [report_property](#)

highlight_objects

Highlight objects in different colors.

Syntax

```
highlight_objects [-color_index arg] [-rgb args] [-color arg] [-quiet]  
[-verbose] objects
```

Returns

Nothing

Usage

Name	Description
<code>[-color_index]</code>	Color index
<code>[-rgb]</code>	RGB color index list
<code>[-color]</code>	Valid values are red green blue magenta yellow cyan and orange
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>objects</code>	Objects to highlight

Categories

[GUIControl](#)

Description

Highlights the specified object or objects in a color as determined by one of the color options. Objects can be unhighlighted with the **unhighlight_objects** command.

Note Only one color option should be used to specify the highlight color. However, if more than one color option is specified, the order of precedence used to define the color is -rgb, -color_index, and -color

Arguments

-color_index arg - (Optional) The color index to use for highlighting the selected object or objects. The color index is defined by the Highlight category of the Tools > Options > Themes command. Refer to the *Vivado Design Suite User Guide: Using the IDE (UG893)* for more information on setting themes.

-rgb args - (Optional) The color to use in the form of an RGB code specified as {R G B}. For instance, {255 255 0} specifies the color yellow.

-color arg - (Optional) The color to use for highlighting the specified object or objects. Supported highlight colors are: red, green, blue, magenta, yellow, cyan, and orange.

Note White is the color used to display selected objects with the **select_objects** command

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) Specifies one or more objects to be highlighted.

Examples

The following example highlights the currently selected objects in the color red:

```
highlight_objects -color red [get_selected_objects]
```

See Also

- [get_selected_objects](#)
- [unhighlight_objects](#)

implement_debug_core

Implement debug core.

Syntax

```
implement_debug_core [-quiet] [-verbose] [cores...]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[cores]	Debug core

Categories

Description

Implements the debug cores in the Vivado tool. The tools will be run once for any ILA debug cores specified, and run one more time for the Debug Hub core if all cores are specified. The ILA core (labtools_ila_v2) is the only core type currently supported by the **create_debug_core** command. The tool automatically adds a Debug Hub core (labtools_xsdbmasterlib_v2) to contain and configure the ILA cores in the project.

The Vivado tool creates Debug Hub core and ILA cores initially as black boxes. These cores must be implemented prior to running through place and route. After the core is created with **create_debug_core**, and while ports are being added and connected with **create_debug_port** and **connect_debug_port**, the content of the debug core is not defined or visible within the design.

Debug core implementation is automatic when you launch an implementation run using the **launch_runs** command. However, you can also use the **implement_debug_core** command to implement one or more of the cores in the CORE Generator tool without having to implement the whole design.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

cores - (Optional) One or more debug cores to implement. All debug cores will be implemented if no cores are specified.

Examples

The following example implements all debug cores in the current project:

```
implement_debug_core [get_debug_cores]
```

See Also

- [connect_debug_port](#)
- [create_debug_core](#)
- [create_debug_port](#)
- [get_debug_cores](#)
- [launch_runs](#)

import_files

Import files and/or directories into the active fileset.

Syntax

```
import_files [-fileset arg] [-force] [-norecurse] [-flat]  
[-relative_to arg] [-quiet] [-verbose] [files...]
```

Returns

A list of file objects that were imported

Usage

Name	Description
[-fileset]	Fileset name
[-force]	Overwrite files of the same name in project directory
[-norecurse]	Disables the default behavior of recursive directory searches
[-flat]	Import the files into a flat directory structure
[-relative_to]	Import the files with respect to the given relative directory
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>files</i>]	Name of the files to import into fileset

Categories

[Project](#), [Simulation](#)

Description

Imports one or more files or the source file contents of one or more directories to the specified fileset.

This command is different from the **add_files** command, which adds files by reference into the specified fileset. This command imports the files into the local project folders under `project.srcs\<fileset>\imports` and then adds the file to the specified fileset.

Arguments

-fileset *name* - (Optional) The fileset to which the specified source files should be added. If the specified fileset does not exist, the tool will return an error. If no fileset is specified the files will be added to the source fileset by default.

-force - (Optional) Overwrite files of the same name in the local project directory and in the fileset.

-norecurse - (Optional) Do not recurse through subdirectories of any specified directories. Without this argument the tool will also search through any subdirectories for additional source files that can be added to a project.

-flat - (Optional) Import all files into the imports folder without preserving their relative paths. By default the directory structure of files is preserved as they are imported into the design.

-relative_to arg - (Optional) Import the files relative to the specified directory. This allows you to preserve the path to the imported files in the directory structure of the local project. The files will be imported to the imports folder with the path relative to the specified directory.

Note The **relative_to** argument is ignored if the **-flat** argument is also specified. The **-flat** command eliminates the directory structure of the imported files.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Optional) One or more file names or directory names to be added to the specified fileset. If a directory name is specified, all valid source files found in the directory, and in subdirectories of the directory, will be added. If no files are specified, the tool imports files in the source set for the current project.

Note If the path is not specified as part of the file name, the current working directory is used, or the directory from which the tool was launched.

Examples

The following example imports the top.ucf file into the constrs_1 constraint filesset.

```
import_files -fileset constrs_1 top.ucf
```

The following example imports the valid source files into the source fileset (sources_1) as a default since the **-fileset** argument is not specified. In addition, the **-norecurse** argument restricts the tool to looking only in the specified \level1 directory and not searching any subdirectories. All valid source files will be imported into the \imports folder of the project because the **-flat** argument has been specified.

```
import_files C:/Data/FPGA_Design/level1 -norecurse -flat
```

Note Without the **-flat** option a \level1 directory would be created inside of the \imports folder of the project.

The following example imports files into the source fileset (sources_1) because the **-fileset** argument is not specified. Valid source files are imported from the \level1 directory, and all subdirectories, and the files will be written into the \imports folder of the project starting at the \Data directory due to the use of the **-relative_to** argument.

```
import_files C:/Data/FPGA_Design/level1 -relative_to C:/Data
```


See Also

[add_files](#)

import_ip

Import an IP file and add it to the fileset.

Syntax

```
import_ip [-srcset arg] [-name arg] [-quiet] [-verbose] [files]
```

Returns

List of file objects that were added

Usage

Name	Description
<code>[-srcset]</code>	Source set name
<code>[-name]</code>	New name for the imported IP, may not be used with multiple files
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[files]</code>	Names of the IP files to be imported

Categories

[Project](#), [IPFlow](#)

Description

Imports an existing XCI or XCO file as an IP source into the current project.

The **import_ip** command allows you to read existing IP files directly, and import them into the local project folders. Use the **read_ip** or **add_files** command to add IP files by reference into the current project.

Use the **create_ip** command to create new IP files from the current IP catalog.

Arguments

-files *arg* - (Optional) The IP file to be imported into the current project. The IP must be in the form of an existing XCI file or XCO file. An XCI file is an IP-XACT format file that contains information about the IP parameterization. An XCO file is a CORE Generator log file that records all the customization parameters used to create the IP core and the project options in effect when the core was generated. The XCI or XCO files are used to recreate the core in the current project.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

-name *arg* - (Optional) The name to assign to the IP object as it is added to the current source fileset.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example imports the 10gig ethernet core into the current project, and assigns it a name of IP_block1:

```
import_ip C:/Data/FPGA_Design/10gig_eth.xci -name IP_block1
```

See Also

- [add_files](#)
- [create_ip](#)
- [generate_target](#)
- [read_ip](#)

import_synplify

Imports the given Synplify project file.

Syntax

```
import_synplify [-copy_sources] [-quiet] [-verbose] file
```

Returns

List of files object that were imported from the Synplify file

Usage

Name	Description
[-copy_sources]	Copy all the sources from synplify project file into the created project
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Name of the Synplify project file to be imported

Categories

[Project](#)

Description

Imports Synplify synthesis project files (.prj) into the current project, including the various source files used in the synthesis run.

Arguments

-copy_sources - (Optional) Copy Synplify project source files to the local project directory structure rather than referencing them from their current location. The default is to reference source files from their current location.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The name of the Synplify project file from which to import the source files.

Examples

The following example creates a new project and imports the specified Synplify project file, copying the various source files from the Synplify project into the local project directories:

```
create_project syn_test C:/Data/FPGA_Design/syn_test  
import_synplify -copy_sources C:/Data/syn_data.prj
```

See Also

[create_project](#)

import_xise

Import XISE project file settings into the created project.

Syntax

```
import_xise [-copy_sources] [-quiet] [-verbose] file
```

Returns

True

Usage

Name	Description
[-copy_sources]	Copy all ISE sources into the created project
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Name of the XISE project file to be imported

Categories

[Project](#)

Description

Imports an ISE project file (XISE) into the current project. This allows ISE projects to be quickly migrated into the Vivado Design Suite for synthesis, simulation, and implementation. All project source files, constraint files, simulation files, and run settings are imported from the ISE project and recreated in the current project.

This command should be run on a new empty project. Since source files, constraints, and run settings are imported from the ISE project, any existing source files or constraints may be overwritten.

Arguments

-copy_sources - (Optional) Copy source files in the ISE project to the local project directory structure rather than referencing them from their current location. The default is to reference source files from their current location.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The name of the ISE project file (.XISE) to be imported into the current project.

Examples

The following example creates a new project called importISE, and then imports the ISE project file (first_use.xise) into the new project.

```
create_project importISE C:/Data/importISE import_xise \  
C:/Data/FPGA_design/ise_designs/drp_des/first_use.xise
```

Note This example does not specify the **-copy_sources** argument, so all source files in the ISE project will be added to the current project by reference.

See Also

[create_project](#)

import_xst

Imports the given XST project file.

Syntax

```
import_xst [-copy_sources] [-quiet] [-verbose] file
```

Returns

List of files object that were imported from the XST file

Usage

Name	Description
[-copy_sources]	Copy all the sources from xst project file into the created project
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Name of the XST project file to be imported

Categories

[Project](#)

Description

Imports XST synthesis project files into the current project, including the various source files used in the XST run.

Arguments

-copy_sources - (Optional) Copy XST project source files to the local project directory structure rather than referencing them from their current location. The default is to reference source files from their current location.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The name of the XST project file from which to import the source files.

Examples

The following example creates a new project called `xst_test`, and imports the `drp_des.xst` file:

```
create_project xst_test C:/Data/FPGA_Design/xst_test
import_xst C:/Data/ise_designs/drp_des.xst
```

See Also

[create_project](#)

infer_diff_pairs

Infer differential pairs, typically for ports just imported from a CSV or XDC file.

Syntax

```
infer_diff_pairs [-file_type arg] [-quiet] [-verbose] [file...]
```

Returns

Nothing

Usage

Name	Description
<code>[-file_type]</code>	Input file type: 'csv' or 'xdc' Default: file type
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[file]</code>	Pin Planning CSV or XDC file Default: file

Categories

[FileIO](#)

Description

The **infer_diff_pairs** command can be used in an I/O Pin Planning project, after importing the I/O pin information using the **read_csv** or **read_xdc** command.

There are several attributes that identify differential pairs in the file: Signal Name, DiffPair Signal, DiffPair Type, and I/O Standard.

The tool will identify differential pairs using the following methods:

- Matching Diff Pair - This is a direct definition of the two signals which make up a differential pair. Two port entries, each have DiffPair Signal values linking to the Signal Name of the other, and have complementary DiffPair Type values, one N and one P. The tool checks to ensure that the other attributes such as I/O Standard are compatible when forming the diff pair.
- Unmatched Diff Pair - Two port entries, with complementary DiffPair Type values (one N, one P), but only one port has a DiffPair Signal linking to the other Signal Name. The tool will create the differential pair if all other attributes are compatible.
- Single Port Diff Pair - A single port entry with a differential I/O Standard, a DiffPair Type value, and a DiffPair Signal that does not otherwise appear in the CSV. The tool will create the opposite side of the differential pair (the N or P side), with all properties matching those on the original port.

- Inferred Diff Pair - Two ports entries, with Signal Names that imply the N and P side. The tool will infer a differential pair if all other attributes are compatible.

After reading the port definitions from a CSV or XDC file, the tool will report that some differential pairs can be inferred from the data. You can run the **infer_diff_pairs** command to infer these differential pairs if you choose.

Arguments

-file_type [csv | xdc] - (Optional) Specify the type of file to import when inferring differential pairs. The valid file types are CSV and XDC. There is no default; the **-file_type** must be specified.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Optional) The name of the file previously imported.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example imports the specified XDC file, and then infers differential pairs from the file:

```
read_xdc C:/Vivado_Install/io_1.xdc
infer_diff_pairs C:/Vivado_Install/io_1.xdc -file_type xdc
```

See Also

- [read_csv](#)
- [read_xdc](#)

launch_chipscope_analyzer

Launch ChipScope Analyzer tool for a run.

Syntax

```
launch_chipscope_analyzer [-run arg] [-csproject arg] [-quiet]  
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[-run]	Implemented run to launch ChipScope Analyzer with
[-csproject]	ChipScope project
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[ToolLaunch](#), [ChipScope](#)

Description

Launches the ChipScope™ Pro Analyzer tool for the active run, or a specified Implemented Design run. You can setup a Netlist Design for use with ChipScope prior to implementation, using the **create_debug_core**, **create_debug_port**, and **connect_debug_port** commands.

The Implemented Design must also have a bitstream file generated by BitGen for **launch_chipscope_analyzer** to run. If BitGen has not been run, an error will be returned.

Note It is not enough to use the **write_bitstream** command to create a bitstream file. You must follow the steps outlined below in the second example

Arguments

-run *arg* - The run name to use when launching the ChipScope Pro Analyzer. The specified run must be implemented and have a bitstream (.bit) file generated. ChipScope will use the bitstream file and the `debug_nets.cdc` file from the specified run.

-csproject *arg* - The name of the project to open in ChipScope Pro Analyzer. If you do not specify the project name, the default project name of `csdefaultproj.cpj` will be used. When you specify the project name, you should also specify the `.cpj` extension.

Note The project is created in the `project/project.data/sources_1/cs` folder.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example launches ChipScope Pro Analyzer, specifying the implementation run to use and the name of the ChipScope project to create:

```
launch_chipscope_analyzer -run impl_3 -csproject impl_3_cs_project
```

The following example sets the **add_step Bitgen** property for the `impl_4` run, launches the `impl_4` run, and then launches the ChipScope Pro Analyzer on the specified run:

```
set_property add_step Bitgen [get_runs impl_4]
launch_runs impl_4 -jobs 2
launch_chipscope_analyzer -run impl_4
```

Note In this example the ChipScope project will be called `csdefaultproj.cpj`.

See Also

- [connect_debug_port](#)
- [create_debug_core](#)
- [create_debug_port](#)
- [launch_runs](#)
- [set_property](#)
- [write_bitstream](#)

launch_impact

Launch iMPACT configuration tool for a run.

Syntax

```
launch_impact [-run arg] [-ipf arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-run]	Implemented run to launch iMPACT with
[-ipf]	Project for iMPACT
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[ToolLaunch](#), [ChipScope](#)

Description

Launch iMPACT to configure your device and generate programming files. You can also read back and verify design configuration data, debug configuration problems, or execute XSVF files.

You must generate the bitstream file using **write_bitstream** prior to using iMPACT.

The command returns the list of files read.

Arguments

-run - (Optional) Launch iMPACT with the specified run. If no run is specified, then iMPACT is launched with the active implementation run.

-ipf - (Optional) Specify the iMPACT project file to use to save the results to. The iMPACT Project File (IPF) contains information from a previous session of iMPACT. The target device is configured according to the settings in the specified IPF file. If you do not specify **-ipf**, the target device is configured according to the default settings.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example launches iMPACT using the specified implementation run:

```
launch_impact -run impl_3
```

See Also

[write_bitstream](#)

launch_modelsim

Launch simulation using ModelSim simulator.

Syntax

```
launch_modelsim [-simset arg] [-mode arg] [-type arg] [-noclean_dir]  
[-scripts_only] [-install_path arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-simset]	Name of the simulation fileset
[-mode]	Simulation mode. Values: behavioral, post-synthesis, post-implementation Default: behavioral
[-type]	Netlist type. Values: functional, timing. This is only applicable when mode is set to post-synthesis or post-implementation
[-noclean_dir]	Do not remove simulation run directory files
[-scripts_only]	Only generate scripts
[-install_path]	Custom ModelSim installation directory path
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[ToolLaunch](#), [Simulation](#)

Description

Launch the Mentor Graphics ModelSim or Questa Advanced Simulator tool. The specified simulator must be installed, and appear in your \$PATH in order to be properly invoked when launching simulation.

To launch ModelSim or Questa, you must first set the target simulator property for the project:

```
set_property target_simulator ModelSim [current_project]
```


In order to support the use of ModelSim/Questa you must compile the Xilinx simulation libraries for use with the target simulator using the **compile_simlib** command. After the libraries are compiled, the simulator will reference these compiled libraries using the `modelsim.ini` file. The `modelsim.ini` file is the default initialization file and contains control variables that specify reference library paths, optimization, compiler and simulator settings. The `modelsim.ini` is located as follows:

- The path specified by **-directory** argument at the time **compile_simlib** is run.
- The path defined by `MODELSIM` environment variable.
- The path defined by `MGC_WD` environment variable.
- The simulation run directory of the project.

Note If the `modelsim.ini` file is not found at any of these locations a warning message is returned by the simulator.

The command returns the transcript of the simulator.

Arguments

-simset *arg* - (Optional) Name of the simulation fileset containing the simulation test benches and sources to be used during simulation. If not specified, the current simulation fileset is used.

-mode [**behavioral** | **post_synthesis** | **post_implementation**] - (Optional) Simulation mode. Specifies either a behavioral simulation of the HDL design sources to verify syntax and confirm that the design performs as intended, a functional or timing simulation of the post-synthesis netlist, or a functional or timing simulation of the post implementation design to verify circuit operation after place and route. The default mode is **behavioral**.

-type [**functional** | **timing**] - (Optional) Cannot be used with **-mode behavioral**. Specifies functional simulation of just the netlist, or timing simulation of the netlist and SDF file. The default is **functional**. Post-synthesis timing simulation uses SDF component delays from the **synth_design** command. Post-implementation timing simulation uses SDF delays from the **place_design** and **route_design** commands.

Note Do not use **-type** with **-mode behavioral**, or the tool will return an error.

-noclean_dir - (Optional) Do not remove simulation run directory files prior to launching the simulator. However, some of the files generated for use by the simulator will be overwritten or updated by re-launching the simulator. The default is to remove the simulation run directory before launching the simulator.

-scripts_only - (Optional) Indicates that the command scripts for launching ModelSim should be generated at this time. The scripts can be used to launch the simulator at a later time.

-install_path *arg* - (Optional) Specifies the directory containing the ModelSim executables (`vlog.exe`, `vcom.exe` and `vsim.exe`). If this option is not specified, the tool will be looked for in the current `PATH` definition.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example launches the ModelSim simulator in post-implementation timing mode:

```
launch_modelsim -mode post-implementation -type timing
```

See Also

- [compile_simlib](#)
- [launch_xsim](#)
- [set_property](#)

launch_runs

Launch a set of runs.

Syntax

```
launch_runs [-jobs arg] [-scripts_only] [-all_placement] [-dir arg]
[-to_step arg] [-next_step] [-host args] [-remote_cmd arg]
[-email_to args] [-email_all] [-pre_launch_script arg]
[-post_launch_script arg] [-force] [-quiet] [-verbose] runs...
```

Returns

Nothing

Usage

Name	Description
[-jobs]	Number of jobs Default: 1
[-scripts_only]	Only generate scripts
[-all_placement]	Export all fixed and non-fixed placement to ISE (by default only fixed placement will be exported)
[-dir]	Launch directory
[-to_step]	Last Step to run. Ignored when launching multiple runs. Not valid with -next_step
[-next_step]	Run next step. Ignored when launching multiple runs. Not valid with -to_step.
[-host]	Launch on specified remote host with a specified number of jobs. Example: -host {machine1 2} -host {machine2 4}
[-remote_cmd]	Command to log in to remote hosts Default: ssh -q -o BatchMode=yes
[-email_to]	List of email addresses to notify when jobs complete
[-email_all]	Send email after each job completes
[-pre_launch_script]	Script to run before launching each job
[-post_launch_script]	Script to run after each job completes
[-force]	Run the command, even if there are pending constraint changes, which will be lost (in a Partial Reconfig design)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>runs</i>	Runs to launch

Categories

Project

Description

Launches synthesis and implementation runs when running the Vivado tools in Project Mode. Refer to the *Vivado Design Suite User Guide: Design Flows Overview (UG892)* for a complete description of Project Mode and Non-Project Mode.

A run must be previously defined using the **create_run** command, and the properties of the run must be previously configured using the **set_property** command. Both synthesis and implementation runs can be specified in the same **launch_runs** command. However, to launch an implementation run, the parent synthesis run must already be complete.

In Non-Project Mode, Vivado synthesis can be launched directly using the **synth_design** command, and does not require the use of a defined run.

In Non-Project Mode, Vivado implementation steps can be launched individually with the **opt_design**, **power_opt_design**, **place_design**, **route_design**, **phys_opt_design**, and **write_bitstream** commands.

Arguments

-jobs arg - (Optional) The number of parallel jobs to run on the local host. The number of jobs for a remote host is specified as part of the **-host** argument. You do not need to specify both **-jobs** and **-host**.

-scripts_only - (Optional) Generate a script called `runme.bat` for each specified run so you can queue the runs to be launched at a later time.

-all_placement - (Optional) Export all user-assigned (fixed) placements as well as auto-assigned (unfixed) placements for implementation. As a default, the tool will export only the fixed or user-assigned placement for implementation.

-dir arg - (Optional) The directory for the tool to write run results into. A separate folder for each run is created under the specified directory. As a default the tool will write the results of each run into a separate folder under the `project.runs` directory.

-to_step *arg* - (Optional) Launch the run through the specified step in the implementation process, and then stop. For instance, run implementation through the place_design step, and then stop. This will allow you to look at specific stages of a run without completing the entire run. The following are the valid steps for implementation runs.

- **opt_design** - Optionally optimize the logical design to more efficiently use the target device resources. This step is usually enabled by default even though it is an optional step.
- **power_opt_design** - Optionally optimize elements of the logic design to reduce power demands of the implemented FPGA.
- **place_design** - Place logic cells onto the target device. This is a required step.
- **power_opt_design (Post-Place)** - Optionally optimize power demands of the placed logic elements. This step must be enclosed in quotes or braces since it includes multiple words (e.g. **-to_step "power_opt_design (Post-Place)"**).
- **phys_opt_design** - Optionally optimize design timing by replicating drives of high-fanout nets to better distribute the loads.
- **route_design** - Route the connections of the design onto the target FPGA. This is a required step.
- **write_bitstream** - Generate a bitstream file for Xilinx device configuration. This is a required step.

Note The specified **-to_step** must be enabled for the implementation run using the **set_property** command, or the Vivado tool will return an error

-next_step - (Optional) Continue a prior run from the step at which it was stopped. This option can be used to complete a run previously launched with the **-to_step** argument.

Note The **-to_step** and **-next_step** arguments may not be specified together, and are ignored when launching multiple runs

-host *args* - (Optional) Launch on the named remote host with a specified number of jobs. The argument is in the form of {*hostname jobs*}, for example: **-host {machine1 2}**. If the **-host** argument is not specified, the runs will be launched from the local host.

Note This argument is supported on the Linux platform only.

-remote_cmd *arg* - (Optional) The command to use to login to the remote host to launch jobs. The default remote command is "ssh -q -o BatchMode=yes".

-email_to *args* - (Optional) Email addresses to send a notification when the runs have completed processing.

-email_all - (Optional) Send a separate Email for each run as it completes.

-pre_launch_script *arg* - (Optional) A Tcl script to run before launching each job.

-post_launch_script *arg* - (Optional) A Tcl script to run after completion of all jobs.

-force - (Optional) Launch the run regardless of any pending constraint changes for Partial Reconfiguration designs.

Note This argument applies only to Partial Reconfiguration projects. Any pending constraint changes will be lost to the specified runs.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

runs - (Required) The names of synthesis and implementation runs to launch. One or more run names may be specified.

Examples

The following command launches three different synthesis runs with two parallel jobs:

```
launch_runs synth_1 synth_2 synth_4 -jobs 2
```

Note The results for each run will be written to a separate folder `synth_1`, `synth_2`, and `synth_4` inside of the `project.runs` directory.

The following example creates a results directory to write run results. In this case a separate folder named `impl_3`, `impl_4`, and `synth_3` will be written to the specified directory. In addition, the **-scripts_only** argument tells the tool to write `runme.bat` scripts to each of these folders but not to launch the runs at this time.

```
launch_runs impl_3 impl_4 synth_3 -dir C:/Data/FPGA_Design/results -scripts_only
```

The following example configures the `impl_1` run, setting options for Vivado Implementation 2013, enabling some of the optional optimizations, and then launches the run to the **place_design** step:

```
set_property flow {Vivado Implementation 2013} [get_runs impl_1]
set_property STEPS.POWER_OPT_DESIGN.IS_ENABLED true [get_runs impl_1]
set_property STEPS.POST_PLACE_POWER_OPT_DESIGN.IS_ENABLED true [get_runs impl_1]
set_property STEPS.PHYS_OPT_DESIGN.IS_ENABLED true [get_runs impl_1]
launch_runs -to_step place_design impl_1
```

See Also

- [create_run](#)
- [get_runs](#)
- [opt_design](#)
- [phys_opt_design](#)
- [place_design](#)
- [power_opt_design](#)
- [reset_run](#)
- [route_design](#)
- [set_property](#)
- [synth_design](#)
- [write_bitstream](#)

launch_sdk

Launch Xilinx Software Development Kit (SDK).

Syntax

```
launch_sdk [-bit arg] [-bmm arg] [-workspace arg] [-hwspec arg]  
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-bit]	Specify the bitstream file for FPGA programming
[-bmm]	Specify the BMM file for BRAM initialization
[-workspace]	Specify the workspace directory for SDK projects
[-hwspec]	Specify the hardware platform specification file (.xml)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[ToolLaunch](#), [XPS](#)

Description

Launch the Software Development Kit (SDK) to design the software for embedded processor sources in your project.

This command follows the **export_hardware** command, which exports the embedded processor hardware specification file (*system.xml*) for use by SDK. By default, the `export_hardware` command will write the hardware specification file (.xml) for the specified embedded processors to the *project_name.sdk/SDK/SDK_Export/hw* directory, to a file named after the embedded processor in the design, with the .XML extension.

The command returns a transcript of the SDK tool launch.

Arguments

-bit arg - (Optional) Specify the bitstream file for FPGA programming.

-bmm arg - (Optional) Specify the BMM file for BRAM initialization.

-workspace *arg* - (Optional) Specify the workspace directory for SDK projects. This is the folder in which your software projects are stored.

-hwspec *arg* - (Optional) The hardware platform specification file (.xml) for the Embedded Processor design. This is the file exported by the `export_hardware` command, or is the `system.xml` file found in the `sources/edk` directory of the project.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **`set_msg_config`** command.

Examples

The following example launches SDK, loading the specified hardware specification file for the project, and indicates the workspace to use:

```
launch_sdk -hwspec C:/Data/export_sdk/hw/robot.xml -workspace C:/Data/sdk_work/
```

See Also

[export_hardware](#)

launch_xsim

Launch simulation using XSim simulator.

Syntax

```
launch_xsim [-simset arg] [-noclean_dir] [-scripts_only] [-mode arg]  
[-type arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-simset]	Name of the simulation fileset
[-noclean_dir]	Do not remove simulation run directory files
[-scripts_only]	Only generate scripts
[-mode]	Simulation mode. Values: behavioral, post-synthesis, post-implementation Default: behavioral
[-type]	Netlist type. Values: functional, timing
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[ToolLaunch](#), [Simulation](#)

Description

Launch the integrated Vivado simulator in behavioral mode, or for functional or timing simulation of the post-synthesis or post-implementation nestlist.

Launching the Vivado simulator first runs xelab, the RTL elaborator, compiler, and linker used to create a simulation snapshot used by the Vivado simulator.

The Vivado simulator is then launched, using the xelab created snapshot.

The command returns the transcript of xelab and xsim.

Arguments

-simset *arg* - (Optional) Name of the simulation fileset containing the simulation test benches and sources to be used during simulation. If not specified, the current simulation fileset is used.

-noclean_dir - (Optional) Do not remove simulation run directory files prior to launching the simulator. However, some of the files generated for use by the simulator will be overwritten or updated by re-launching the simulator. The default is to remove the simulation run directory before launching the simulator.

-scripts_only - (Optional) Just generate the scripts for launching the Vivado simulator, rather than actually launching the tool. You can use the scripts to launch the simulator at a later time.

-mode [behavioral | post_synthesis | post_implementation] - (Optional) Simulation mode. Specifies either a behavioral simulation of the HDL design sources to verify syntax and confirm that the design performs as intended, a functional or timing simulation of the post-synthesis netlist, or a functional or timing simulation of the post implementation design to verify circuit operation after place and route. The default mode is **behavioral**.

-type [functional | timing] - (Optional) Cannot be used with **-mode behavioral**. Specifies functional simulation of just the netlist, or timing simulation of the netlist and SDF file. The default is **functional**. Post-synthesis timing simulation uses SDF component delays from the **synth_design** command. Post-implementation timing simulation uses SDF delays from the **place_design** and **route_design** commands.

Note Do not use **-type** with **-mode behavioral**, or the tool will return an error.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example launches the simulator in post implementation mode:

```
launch_xsim -mode post_implementation
```

See Also

- [close_sim](#)
- [current_sim](#)

limit_vcd

Limit the maximum size of the VCD file on disk (equivalent of \$dumplimit verilog task).

Syntax

```
limit_vcd [-quiet] [-verbose] filesize
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>filesize</i>	Specify the maximum size of the VCD file in bytes.

Categories

[Simulation](#)

Description

Specify the size limit, in bytes, of the Value Change Dump (VCD) file. This command operates like the Verilog **\$dumplimit** simulator directive.

When the specified file size limit has been reached, the dump process stops, and a comment is inserted into the VCD file to indicate that the file size limit has been reached.

Note: You must run the **open_vcd** command before using the **limit_vcd** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

filesize - (Required) Specify the file size limit of the open VCD file in bytes.

Examples

The following example limits the current VCD file:

```
limit_vcd 1000
```

See Also

- [checkpoint_vcd](#)
- [flush_vcd](#)
- [log_vcd](#)
- [open_vcd](#)

link_design

Open a netlist design.

Syntax

```
link_design [-name arg] [-part arg] [-constrset arg] [-top arg]  
[-mode arg] [-quiet] [-verbose]
```

Returns

Design object

Usage

Name	Description
[-name]	Design name
[-part]	Target part
[-constrset]	Constraint fileset to use
[-top]	Specify the top module name when the structural netlist is Verilog
[-mode]	The design mode. Values: default, out_of_context Default: default
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Tools](#)

Description

Opens a new or existing Netlist design, linking the netlists and constraints with the target part to create the design. This can also be accomplished with the **open_run** command.

The design_mode property for the current source fileset must be defined as GateLvl in order to open a Netlist design. If not, you will get the following error:

```
ERROR: The design mode of 'sources_1' must be GateLvl.
```

Arguments

-name arg - (Optional) The name of a new or existing Netlist design.

-part arg - (Optional) The Xilinx device to use when creating a new design. If the part is not specified the default part will be used.

-constrset *arg* - (Optional) The name of the constraint fileset to use when opening the design.

Note The **-constrset** argument must refer to a constraint fileset that exists. It cannot be used to create a new fileset. Use `create_fileset` for that purpose.

-top *arg* - (Optional) The top module of the design hierarchy of the netlist.

-mode [default | out_of_context] - (Optional) If you have synthesized a block, and disabled IO buffer insertion, you can load the resulting EDIF into the Vivado Design Suite using **-mode out_of_context**. This enables implementation of the module without IO buffers, prevents optimization due to unconnected inputs or outputs, and adjusts DRC rules appropriately for the design. Refer to the *Vivado Design Suite User Guide: Hierarchical Design (UG905)* for more information.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following creates a new Netlist design called Net1:

```
link_design -name Net1
```

Note The default source set, constraint set, and part will be used in this example.

The following example opens a Netlist design called Net1, and specifies the constraint set to be used:

```
link_design -name Net1 -constrset con1
```

See Also

[open_run](#)

list_features

List available features.

Syntax

```
list_features [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Tools](#)

Description

List the available features sets of the Vivado Design Suite that can be loaded with the `load_features` command.

Note If a feature has been previously loaded, it will not be listed as a feature available to load

This command returns a list of features, or an error message.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the list of features available to load into the Vivado Design Suite:

```
list_features
```

See Also

- [help](#)
- [load_features](#)

list_param

Get all parameter names.

Syntax

```
list_param [-quiet] [-verbose]
```

Returns

List

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[PropertyAndParameter](#)

Description

Gets a list of user-definable configuration parameters. These parameters configure a variety of settings and behaviors of the tool. For more information on a specific parameter use the **report_param** command, which returns a description of the parameter as well as its current value.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns a list of all user-definable parameters:

```
list_param
```

See Also

- [get_param](#)
- [report_param](#)
- [reset_param](#)
- [set_param](#)

list_property

List properties of object.

Syntax

```
list_property [-class arg] [-regexp] [-quiet] [-verbose] [object]  
[pattern]
```

Returns

List of property names

Usage

Name	Description
[-class]	Object type to query for properties. Ignored if object is specified.
[-regexp]	Pattern is treated as a regular expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>object</i>]	Object to query for properties
[<i>pattern</i>]	Pattern to match properties against Default: *

Categories

[Object](#), [PropertyAndParameter](#)

Description

Gets a list of all properties on a specified object or class.

Note report_property also returns a list of properties on an object, but includes the property type and property value.

Arguments

-class *arg* - (Optional) The class of object for which to list the properties.

Note When both **-class** and *object* are specified, the properties of the specific object are returned.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

object - (Optional) The single object on which to report properties.

Note If you specify multiple objects you will get an error.

Examples

The following example returns all properties of the specified object:

```
list_property [get_cells cpuEngine]
```

See Also

- [create_property](#)
- [get_cells](#)
- [get_property](#)
- [list_property_value](#)
- [report_property](#)
- [reset_property](#)
- [set_property](#)

list_property_value

List legal property values of object.

Syntax

```
list_property_value [-default] [-class arg] [-quiet] [-verbose] name  
[object]
```

Returns

List of property values

Usage

Name	Description
[-default]	Show only the default value.
[-class]	Object type to query for legal property values. Ignored if object is specified.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of property whose legal values is to be retrieved
[<i>object</i>]	Object to query for legal properties values

Categories

[Object](#), [PropertyAndParameter](#)

Description

Gets a list of valid values for an enumerated type property of either a class of objects or a specific object.

Note The command cannot be used to return valid values for properties other than enum properties. The **report_property** command will return the type of property to help you identify enum properties.

Arguments

-default - (Optional) Return the default value for the specified class of objects.

-class *arg* - (Optional) The class of object to query. The class of object can be used in place of an actual object.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the property to be queried. Only properties with an enumerated value, or a predefined value set, can be queried with this command. All valid values of the specified property will be returned.

object - (Optional) An object to query. An actual object can be used in place of the **-class** argument to specify the type of object to query.

Examples

The following example returns the list of valid values for the KEEP_HIERARCHY property from cell objects:

```
list_property_value KEEP_HIERARCHY -class cell
```

The following example returns the same result, but uses an actual cell object in place of the general cell class:

```
list_property_value KEEP_HIERARCHY [get_cells cpuEngine]
```

The following example returns the default value for the specified property by using the current design as a representative of the design class:

```
list_property_value -default BITSTREAM.GENERAL.COMPRESS [current_design]
```

See Also

- [create_property](#)
- [current_design](#)
- [get_cells](#)
- [get_property](#)
- [list_property](#)
- [report_property](#)
- [reset_property](#)
- [set_property](#)

list_targets

List applicable targets for the specified source.

Syntax

```
list_targets [-quiet] [-verbose] files
```

Returns

List of targets

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>files</i>	Source file for which the targets needs to be listed

Categories

[Project](#), [XPS](#)

Description

List the targets that are available for a specified IP core, DSP module, Embedded Processor source, or IP Subsystem. The following file types are accepted: .xci, .xco, .mdl, .xmp, .bd

Use the **generate_targets** command to generate the listed targets.

The command returns the list of available targets. If no targets are available for the specified file objects, nothing is returned.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Required) A files object that contains the list of source files to evaluate.

Note Use **get_files** to specify a files object, rather than specifying a file name.

Examples

The following example lists the available targets for any DSP modules in the design:

```
list_targets [get_files *.mdl]
```

See Also

- [create_bd_design](#)
- [create_sysgen](#)
- [create_xps](#)
- [generate_target](#)
- [get_files](#)
- [import_ip](#)
- [read_ip](#)

load_features

Load Tcl commands for a specified feature.

Syntax

```
load_features [-quiet] [-verbose] [features...]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[features]	Feature(s) to load, use list_features for a list of available features.

Categories

Tools

Description

Load the specified features of the Vivado Design Suite into memory.

To limit the memory footprint of the Vivado tool, some features of the application are only loaded into memory when a command from that feature set is run. For instance, the Vivado simulator feature is only partially loaded prior to actually launching the simulator using the **launch_xsim** command.

To access the complete list of Tcl commands associated with a feature of the Vivado Design Suite, and the help text for these commands, you must load the feature into the application memory using the **load_features** command.

You can list the features that are available to be loaded using the **list_features** command. The list of features is dynamic, and changes from release to release.

The command returns nothing if successful, or an error message if failed.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

features - List of features to load.

Examples

The following example loads the Vivado simulator feature:

```
load_features simulator
```

The following example loads all of the loadable feature sets of the Vivado Design Suite:

```
load_features [list_features]
```

See Also

- [help](#)
- [list_features](#)

lock_design

Locks or unlocks netlist, placement or routing of a design.

Syntax

```
lock_design [-level arg] [-unlock] [-export] [-quiet] [-verbose] [cell]
```

Returns

Nothing

Usage

Name	Description
[-level]	specify the locking level; Valid values are logical, placement, and routing. Default: placement
[-unlock]	Unlock cells, if cells are not specified, whole design is unlocked.
[-export]	mark that the constraints can be exported.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>cell</i>]	Lock cells, if cells are not specified, whole design is locked. Default: *

Categories

[Project](#)

log_saif

Log Switching Activity Interchange Format (SAIF) toggle for specified wire, signal, or reg.

Syntax

```
log_saif [-quiet] [-verbose] hdl_objects ...
```

Returns

Does not return any object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hdl_objects</i>	The hdl_objects to log

Categories

[Simulation](#)

Description

Writes the switching activity rates for the specified HDL signals during the current simulation.

The Switching Activity Interchange format (SAIF) file is an ASCII file containing header information, and toggle counts for the specified signals of the design. It also contains the timing attributes which specify time durations for signals at level 0, 1, X, or Z.

The **log_saif** command can only be used after the **open_saif** command has opened an SAIF file in the current simulation to capture switching activity rates.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

hdl_objects - Specifies the HDL signal names on which to capture code.

Examples

The following example logs switching activity for all signals in the current_scope:

```
log_saif [ get_objects ]
```

Log SAIF for only the internal signals starting with name c of the scope /tb/UUT:

```
log_saif [get_objects filter { type == internal_signal }/tb/UUT/c*]
```

See Also

- [close_saif](#)
- [get_objects](#)
- [open_saif](#)

log_vcd

Log Value Change Dump (VCD) simulation output for specified wire, signal, or reg.

Syntax

```
log_vcd [-level arg] [-quiet] [-verbose] [hdl_objects ...]
```

Returns

Does not return any object

Usage

Name	Description
<code>[-level]</code>	Number of levels to log (for HDL scopes) Default: 0
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[hdl_objects]</code>	Which HDL objects to log

Categories

[Simulation](#)

Description

Indicates which HDL objects to write into the Value Change Dump (VCD) file. In some designs the simulation results can become quite large; the **log_vcd** command lets you define the specific content of interest. This command models the behavior of the Verilog **\$dumpvars** system task.

HDL objects include HDL signals, variables, or constants as defined in the Verilog or VHDL testbench and source files. An HDL signal includes Verilog wire or reg entities, and VHDL signals. Examples of HDL variables include Verilog real, realtime, time, and event.

This command specifies which HDL objects and how many levels of design hierarchy to write into the VCD file. The actual values of the objects are written to the VCD file when you run the **checkpoint_vcd** or **flush_vcd** commands at a specific time during simulation.

Note: You must use the **open_vcd** command before using any other ***_vcd** commands.

Nothing is returned by this command.

Arguments

-level *arg* - (Optional) Specifies the number of levels of design hierarchy to traverse when locating HDL objects to write to the VCD file. The default value of 0 causes the tool to dump all values for the specified HDL objects at the level of hierarchy defined by *hdl_objects*, and all levels below that. A value of 1 indicates that only the level of hierarchy specified by *hdl_objects* should be written to the VCD file.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

hdl_objects - (Optional) Specifies the HDL objects to identify and write changing values into the VCD file. The level of hierarchy is also represented in the *hdl_objects* pattern. For instance */tb/UUT/** indicates all HDL objects within the */tb/UUT* level of the design.

Examples

Log value changes for all the ports from the scope */tb/UUT*:

```
log_vcd [get_objects -filter { type == port } /tb/UUT/* ]
```

Note Since **-levels** is not specified, all levels below the specified scope will be searched for ports matching the specified pattern as well

Log VCD for all the objects in the *current_scope*:

```
log_vcd *  
log_vcd [ get_objects *]
```

Log value changes for only internal signals with names starting with C, of the root scope */tb/UUT*:

```
log_vcd [get_objects -filter { type == internal_signal } ./C*]
```

See Also

- [checkpoint_vcd](#)
- [flush_vcd](#)
- [open_vcd](#)

log_wave

Log simulation output for specified wire, signal, or reg for viewing using Vivado Simulators waveform viewer. Unlike `add_wave`, this command does not add the waveform object to waveform viewer (i.e. Waveform Configuration). It simply enables logging of output to the Vivado Simulators Waveform Database (WDB).

Syntax

```
log_wave [-recursive] [-r] [-quiet] [-verbose] hdl_objects...
```

Returns

Nothing

Usage

Name	Description
<code>[-recursive]</code>	Searches recursively for objects
<code>[-r]</code>	Searches recursively for objects
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>hdl_objects</i>	Which hdl_objects to trace

Categories

[Simulation](#)

ltrace

Turns on or off printing of file name and line number of the hdl statement being simulated.

Syntax

```
ltrace [-quiet] [-verbose] value
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>value</i>	value: on, true, yes. Otherwise set to off, false, no

Categories

[Simulation](#)

make_diff_pair_ports

Make differential pair for 2 ports.

Syntax

```
make_diff_pair_ports [-quiet] [-verbose] ports...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>ports</i>	Ports to join

Categories

[PinPlanning](#)

Description

Joins two existing ports to create a differential pair.

The port directions, interfaces, and other properties must match in order for the specified ports to be joined as a differential pair. Otherwise an error will be returned.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

ports - (Required) Two port objects to join as a differential pair. The first port specified will be the positive side of the differential pair.

Examples

The following example joins the two specified ports to create a differential pair:

```
make_diff_pair_ports port_Pos1 port_Neg1
```

See Also

- [create_interface](#)
- [create_port](#)

make_wrapper

Generate HDL wrapper for the specified source.

Syntax

```
make_wrapper [-top] [-testbench] [-inst_template] [-fileset arg]
[-import] [-force] [-quiet] [-verbose] files
```

Returns

Nothing

Usage

Name	Description
[-top]	Create a top-level wrapper for the specified source
[-testbench]	Create a testbench for the specified source
[-inst_template]	Create an instantiation template for the specified source. The template will not be added to the project and will be generated for reference purposes only.
[-fileset]	Fileset name
[-import]	Import generated wrapper to the project
[-force]	Overwrite existing source(s)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>files</i>	Source file for which the wrapper needs to be generated

Categories

[Project](#), [XPS](#), [SysGen](#)

Description

Create a Verilog or VHDL wrapper for instantiating an IP core, DSP module, or Embedded Processor sub-design into a project.

Note The wrapper is generated in Verilog or VHDL according to the TARGET_LANGUAGE property on the project

The command returns information related to the creation of the wrappers.

Arguments

-top - (Optional) Create a top-level Verilog or VHDL wrapper for the specified source. The wrapper instantiates the DSP module or Embedded Processor sub-design as the top-level of the design hierarchy.

-testbench - (Optional) Create a simulation testbench for the specified DSP module or Embedded Processor sub-design. This includes the DUT module instantiation.

-inst_template - (Optional) Create an instantiation template for the specified source. The template will not be added to the project and will be generated for reference purposes only. The instantiation template can be cut and paste into another RTL file to create an instance of the module in the hierarchy.

-fileset - (Optional) Specify the fileset to add the wrapper to when importing into the project.

-import - (Optional) Import the wrapper file into the project, adding it to the appropriate fileset.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Required) Specify the files to generate wrappers for.

Examples

The following example create the instantiation template to integrate the specified Embedded Processor source into the design hierarchy:

```
make_wrapper -inst_template -fileset [get_filesets sources_1] \  
[get_files C:/Data/edk/xpsTest1/xpsTest1.xmp]
```

See Also

- [add_files](#)
- [create_sysgen](#)
- [create_xps](#)
- [generate_target](#)
- [import_ip](#)
- [list_targets](#)
- [read_ip](#)

mark_objects

Mark objects in GUI.

Syntax

```
mark_objects [-rgb args] [-color arg] [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-rgb]	RGB color index list
[-color]	Valid values are red green blue magenta yellow cyan and orange
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	Objects to mark

Categories

[GUIControl](#)

Description

Marks specified objects in GUI mode. This command places an iconic mark to aid in the location of the specified object or objects. The mark is displayed in a color as determined by one of the color options.

Objects can be unmarked with the **unmark_objects** command.

Note Use only one color option. If both color options are specified, **-rgb** takes precedence over **-color**

Arguments

-rgb args - (Optional) The color to use in the form of an RGB code specified as {R G B}. For instance, {255 255 0} specifies the color yellow, while {0 255 0} specifies green.

-color arg - (Optional) The color to use for marking the specified object or objects. Supported colors are: red, green, blue, magenta, yellow, cyan, and orange.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) One or more objects to be marked.

Examples

The following example adds a red icon to mark the currently selected objects:

```
mark_objects -color red [get_selected_objects]
```

See Also

- [get_selected_objects](#)
- [unmark_objects](#)

move_bd_cells

Move cells into a hierarchy cell. The connections between these cells are maintained; the connections between these cells and other cells are maintained through crossing hierarchy cell.

Syntax

```
move_bd_cells [-prefix arg] [-quiet] [-verbose] [parent_cell]  
[cells...]
```

Returns

0 if success

Usage

Name	Description
[-prefix]	Prefix name to add to cells
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>parent_cell</i>]	Parent cell
[<i>cells</i>]	Match engine names against cell names Default: *

Categories

[IPIntegrator](#)

move_files

Moves the files from one fileset to another while maintaining all of their original properties.

Syntax

```
move_files [-fileset arg] [-quiet] [-verbose] [files...]
```

Returns

List of files that were moved

Usage

Name	Description
[-fileset]	Destination fileset name
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>files</i>]	Name of the files to be moved

Categories

[Project](#), [Simulation](#)

open_bd_design

Open an existing IP subsystem design from disk file.

Syntax

```
open_bd_design [-quiet] [-verbose] name
```

Returns

The design object. Returns nothing if the command fails

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of IP subsystem design to open

Categories

[IPIntegrator](#)

Description

Open an IP subsystem design in the IP Integrator feature of the Vivado IDE. The IP subsystem must previously have been created using the `create_bd_design` command.

This command returns a message with the name of the opened IP subsystem design, or returns an error if the command fails.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - The path and file name of the IP subsystem design to open in the IP Integrator feature of the Vivado Design Suite. The name must include the file extension.

Examples

The following opens the specified IP subsystem design in the current project:

```
open_bd_design C:/Data/project1/project1.src/sources_1/bd/design_1/design_1.bd
```

See Also

- [close_bd_design](#)
- [create_bd_design](#)
- [current_bd_design](#)
- [save_bd_design](#)

open_example_project

Open the example project for the indicated IP.

Syntax

```
open_example_project [-dir arg] [-force] [-in_process] [-quiet]  
[-verbose] objects...
```

Returns

The Project that was opened

Usage

Name	Description
[-dir]	Path to directory where example project will be created
[-force]	Overwrite an example project if it exists
[-in_process]	Open the example project in the same process
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	The objects whose example projects will be opened

Categories

[Project](#), [IPFlow](#)

Description

Open an example project for the specified IP cores. The example project can be used to explore the features of the IP core in a stand-alone project, instead of integrated into the current project.

Arguments

-dir arg - (Optional) Specifies the path to the directory where the example project will be written.

-force - (Optional) Force the opening of a new example project, overwriting an existing example project at the specified path.

-in_process - (Optional) Open the example project in the same tool process as the current project. As a default, without this argument, a new process instance of the tool will be launched for the example project.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) The IP cores to open example projects for.

Examples

The following generates the target data for the example project, then opens the example project for the specified IP core:

```
generate_target {example} [get_ips blk_mem*]  
open_example_project -force [get_ips blk_mem*]
```

Note The Example target data must be generated prior to using the **open_example_project** command. This will create a Tcl script to open and configure the specified IP core

See Also

- [create_ip](#)
- [generate_target](#)
- [get_ips](#)
- [import_ip](#)

open_hw

Open the hardware tool.

Syntax

```
open_hw [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Hardware](#)

open_hw_target

Open a connection to a hardware target on the hardware server.

Syntax

```
open_hw_target [-quiet] [-verbose] [hw_target]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_target]	hardware target Default: current hardware target

Categories

[Hardware](#)

open_io_design

Open an IO design.

Syntax

```
open_io_design [-name arg] [-part arg] [-constrset arg] [-quiet]  
[-verbose]
```

Returns

Design object

Usage

Name	Description
<code>[-name]</code>	Design name
<code>[-part]</code>	Target part
<code>[-constrset]</code>	Constraint filesset to use
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Project](#)

Description

Opens a new or existing I/O Pin Planning design.

Note The design_mode property for the current source fileset must be defined as PinPlanning in order to open an I/O design. If not, you will get the following error:

```
ERROR: The design mode of 'sources_1' must be PinPlanning
```

Arguments

-name *arg* - (Optional) The name of a new or existing I/O Pin Planning design.

-part *arg* - (Optional) The Xilinx device to use when creating a new design. If the part is not specified the default part will be used.

-constrset *arg* - (Optional) The name of the constraint fileset to use when opening an I/O design.

Note The -constrset argument must refer to a constraint fileset that exists. It cannot be used to create a new fileset. Use **create_filesset** for that purpose.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following creates a new I/O design called myIO:

```
open_io_design -name myIO
```

Note The default source set, constraint set, and part will be used in this case.

The following example opens an existing I/O design called myIO, and specifies the constraint set to be used:

```
open_io_design -name myIO -constrset topCon
```

See Also

[create_project](#)

open_project

Open a Vivado project file (.xpr).

Syntax

```
open_project [-read_only] [-quiet] [-verbose] file
```

Returns

Opened project object

Usage

Name	Description
<code>[-read_only]</code>	Open the project in read-only mode
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>file</i>	Project file to be read

Categories

[Project](#)

Description

Opens a project file (.xpr) for editing the design source files and hierarchy, for performing I/O pin planning and floorplanning, and to synthesize and implement the device.

Arguments

-read_only - (Optional) Open the project in read only mode. You will not be able to save any modifications to the project unless you use the **save_project_as** command to save the project to a new editable project.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The project file to open. You must include both the path to the file and the .xpr file extension.

Examples

The following example opens the project named `my_project1` located in the Designs directory.

```
open_project C:/Designs/project1.xpr
```

Note The project must be specified with the `.xpr` extension for the tool to recognize it as a project file. The path to the file must be specified along with the project file name or the tool will return an error that it cannot find the specified file.

See Also

- [create_project](#)
- [current_project](#)

open_run

Open a run into a netlist or implementation design.

Syntax

```
open_run [-name arg] [-quiet] [-verbose] run
```

Returns

Design object

Usage

Name	Description
[-name]	Design name
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>run</i>	Run to open into the design

Categories

[Project](#)

Description

Opens the specified synthesis run into a Netlist Design or implementation run into an Implemented Design. The run properties defining the target part and constraint set are combined with the synthesis or implementation results to create the design view in the tool.

Arguments

-name - (Optional) The name of the design to open.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

run - (Required) Specifies the run name of the synthesis or implementation run to open. The run must have completed synthesis or implementation before it can be opened as a design.

Note If you attempt to open a run that has not been launched the tool will return an error.

Examples

The following command opens the specified synthesis run into a Netlist Design named synthPass1:

```
open_run -name synthPass1 synth_1
```

The following opens an Implemented Design for impl_1:

```
open_run impl_1
```

See Also

[launch_runs](#)

open_saif

Open file for storing signal switching rate for power estimation. The switching rate is written out in Switching Activity Interchange Format (SAIF) Only one SAIF is allowed to be open per simulation run.

Syntax

```
open_saif [-quiet] [-verbose] file_name
```

Returns

The SAIF object that was opened

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file_name</i>	The SAIF filename to store information

Categories

[Simulation](#)

Description

Create or open a Switching Activity Interchange Format (SAIF) file for storing signal switching rates in the current simulation for later use by the **report_power** command.

The Switching Activity Interchange format (SAIF) file is an ASCII file containing header information, and toggle counts for the specified signals of the design. It also contains the timing attributes which specify time durations for signals at level 0, 1, X, or Z.

The SAIF file is recommended for power analysis since it is smaller than the VCD file.

When an SAIF file has been opened, you can write the switching activity from the simulation into the SAIF file using **log_saif**.

Only one SAIF can be open at one time during simulation. To close the SAIF file, use the **close_saif** command.

This command returns the object ID of the opened SAIF file, or returns an error if the command failed.

Arguments

-quiet - Execute the command quietly, ignore any command line errors, and return no error messages if the command fails to execute.

-verbose - Suspends message limits during command execution.

file_name - Specifies the name of the SAIF file to open.

Examples

The following example opens the specified simulation:

```
open_saif myData.saif
```

open_vcd

Open a Value Change Dump (VCD) file for capturing simulation output. This Tcl command models behavior of \$dumpfile Verilog system task.

Syntax

```
open_vcd [-quiet] [-verbose] [file_name]
```

Returns

Returns a Vcd Object and sets the current vcd to this object so that current_vcd command will return this object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[file_name]	file name. Defaults to dump.vcd (This is LRM standard) Default: dump.vcd

Categories

[Simulation](#)

Description

Create or open a Value Change Dump (VCD) file to capture simulation output. This command operates like the Verilog **\$dumpfile** simulator directive.

VCD is an ASCII file containing header information, variable definitions, and value change details of a set of HDL signals. The VCD file can be used to view simulation result in a VCD viewer or to estimate the power consumption of the design.

When a VCD file has been opened, you can write the value changes from the simulation into the VCD file using **checkpoint_vcd**, **flush_vcd**, or **log_vcd**. In addition, you can pause and resume the collection of value change data with the **stop_vcd** and **start_vcd** commands.

You can limit the size of the VCD file by using the **limit_vcd** command.

To close the VCD file, use the close_vcd command.

Note: You must use the **open_vcd** command before using any other *_vcd commands. Only one VCD file can be open at any time.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file_name - (Optional) Is the name of the file into which to dump the current VCD information. When a filename is not specified, the default filename of `dump.vcd` is used. If the specified VCD file already exists, then **open_vcd** resets the VCD file to a new state, overwriting the current contents.

Examples

The following example opens the specified VCD file (`design1.vcd`) so that value changes can be written to it. The **log_vcd** command identifies all ports in the `/tb/UUT` scope, and only that level of the design hierarchy, to be written to the VCD file. The simulation is run for a specified period of time, and **flush_vcd** writes the current values of the HDL objects to the VCD file. Then **close_vcd** closes the open file.

```
open_vcd design1.vcd
log_vcd -level 1 [get_objects filter { type == port } /tb/UUT/* ]
run 1000
flush_vcd
close_vcd
```

See Also

- [checkpoint_vcd](#)
- [close_vcd](#)
- [flush_vcd](#)
- [limit_vcd](#)
- [log_vcd](#)
- [read_vcd](#)
- [start_vcd](#)
- [stop_vcd](#)

open_wave_config

Open a wave config.

Syntax

```
open_wave_config [-data_source arg] [-quiet] [-verbose] [filename]
```

Returns

The wave config opened

Usage

Name	Description
<code>[-data_source]</code>	<code>data_source</code> open connect: specifies whether to open the data source referenced in the WCFG file or connect to an already existing data source
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[filename]</code>	Loads a wave configuration object from the file <code>filename</code> . A new wave window showing that WCFG is also created and made the current wave window

Categories

[Waveform](#)

open_wave_database

Open Waveform Database (WDB) file produced by a prior simulation run and return a simulation object.

Syntax

```
open_wave_database [-quiet] [-verbose] wdb
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>wdb</i>	file name

Categories

[Simulation](#)

opt_design

Optimize the current netlist. This will perform the retarget, propconst, and sweep optimizations by default.

Syntax

```
opt_design [-retarget] [-propconst] [-sweep] [-bram_power_opt] [-remap]
[-resynth_area] [-directive arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-retarget]	Retarget
[-propconst]	Propagate constants across leaf-level instances
[-sweep]	Remove unconnected leaf-level instances
[-bram_power_opt]	Perform Block RAM power optimizations
[-remap]	Remap logic optimally in LUTs
[-resynth_area]	Resynthesis
[-directive]	Mode of behavior (directive) for this command. Please refer to Arguments section of this help for values for this option Default: Default
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Tools](#)

Description

Optimizes a design netlist for the target part. Optimization can provide improvements to synthesized netlists from third-party tools, or for netlists that may not have been optimized during synthesis. The command performs three optimizations by default: Retarget, Constant Propagation, Sweep.

Note Expressly specifying one optimization disables the other optimizations, unless they are also expressly specified.

To perform LUT Remapping, you must specify **-remap**.

To perform area-based re-synthesis, you must specify **-resynth_area**, or **-directive ExploreArea**. Run this command prior to implementation to optimize the design and simplify the netlist before placing and routing the design.

Arguments

-retarget - (Optional) Retarget one type of block to another when retargeting the design from one device family to another. For example, retarget instantiated MUXCY or XORCY components into a CARRY4 block; or retarget DCM to MMCM. The retarget optimization also absorbs inverters into downstream logic where possible.

-propconst - (Optional) Propagate constant inputs through the circuit, resulting in a simplified netlist. Propagation of constants can eliminate redundant combinational logic from the netlist.

-sweep - (Optional) Remove unnecessary logic, removing loadless cells and nets.

-bram_power_opt - (Optional) Enables power optimization on Block RAM cells. Changes the WRITE_MODE on unread ports of true dual-port RAMs to NO_CHANGE, and applies intelligent clock gating to Block RAM outputs.

-remap - (Optional) Remap the design to combine multiple LUTs into a single LUT to reduce the depth of the logic.

-resynth_area - (Optional) Perform re-synthesis in area mode to reduce the number of LUTs.

-directive arg - (Optional) Direct the mode of optimization with specific design objectives. Only one directive can be specified for a single **opt_design** command, and values are case-sensitive. Supported values include:

- **Explore** - Run multiple passes of optimization to improve results.
- **ExploreArea** - Run multiple passes of optimization, with an emphasis on reducing area.
- **AddRemap** - Run the default optimization, and include LUT remapping to reduce logic levels.
- **Default** - Run the default optimization.

Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on the effects of each directive.

Note The **-directive** option controls the overall optimization strategy, and is not compatible with any specific optimization options. It can only be used with **-quiet** and **-verbose**

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command. This option displays detailed information about the logic that is affected by each optimization.

Examples

The following example performs all three default optimizations (retarget, constant propagation, and sweep), and returns detailed results:

```
opt_design -verbose
```

The following example performs the sweep and retarget optimizations:

```
opt_design -sweep -retarget
```

Note Because **-sweep** and **-retarget** are expressly enabled in the prior example, **-propconst** optimization is disabled

The following example directs the **opt_design** command to use various algorithms to achieve potentially better results:

```
opt_design -directive Explore
```

The following example directs the **opt_design** command to use various algorithms to achieve potentially better results, while focusing on area reduction:

```
opt_design -directive ExploreArea
```

See Also

- [phys_opt_design](#)
- [place_design](#)
- [power_opt_design](#)
- [route_design](#)
- [synth_design](#)

phys_opt_design

Optimize the current placed netlist.

Syntax

```
phys_opt_design [-fanout_opt] [-placement_opt] [-rewire]
[-critical_cell_opt] [-dsp_register_opt] [-bram_register_opt]
[-hold_fix] [-retime] [-force_replication_on_nets args]
[-directive arg] [-pinswap] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-fanout_opt]</code>	Do cell-duplication based optimization on high-fanout timing critical nets
<code>[-placement_opt]</code>	Do placement based optimization on timing critical nets
<code>[-rewire]</code>	Do rewiring optimization
<code>[-critical_cell_opt]</code>	Do cell-duplication based optimization on timing critical nets
<code>[-dsp_register_opt]</code>	Do DSP register optimization
<code>[-bram_register_opt]</code>	Do BRAM register optimization
<code>[-hold_fix]</code>	Attempt to improve slack of high hold violators
<code>[-retime]</code>	Do retiming optimization
<code>[-force_replication_on_nets]</code>	Force replication optimization on nets
<code>[-directive]</code>	Mode of behavior (directive) for this command. Please refer to Arguments section of this help for values for this option Default: Default
<code>[-pinswap]</code>	Do pin swap optimization
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Tools](#)

Description

Performs timing-driven optimization on the negative-slack paths of a design. A path should have a negative slack near the worst negative slack (WNS) to be considered for optimization. This optional command should be run after **place_design** and before **route_design**.

The command performs the following optimizations by default: high-fanout optimization, placement-based optimization of critical paths, rewire, critical-cell optimization, DSP register optimization, BRAM register optimization, and a final fanout optimization.

Note Expressly specifying one optimization disables the other optimizations, unless they are also expressly specified.

Physical optimizations involve replication, re-timing, hold fixing, and placement improvement. The **phys_opt_design** command automatically performs all necessary netlist and placement changes.

To perform hold fixing you must specify the **-hold_fix** option, or the **-directive Explore** option.

If the **phys_opt_design** command is used iteratively, the subsequent run optimizes the results of the prior run.

The command reports each net processed, a summary of any optimizations performed, and the WNS before and after optimization. Replicated objects are named by appending `_replica` to the original object name, followed by the replicated object count.

Arguments

-fanout_opt - (Optional) Performs delay-driven optimization on high-fanout timing critical nets, by replicating drivers to reduce delay.

-placement_opt - (Optional) Move cells to reduce delay on timing-critical nets.

-rewire - (Optional) Refactor logic cones to reduce logic levels and reduce delay on critical signals.

-critical_cell_opt - (Optional) Replicate cells on timing critical nets to reduce delays.

-dsp_register_opt - (Optional) Improve critical path delay by moving registers from slices to DSP blocks, or from DSP blocks to slices.

-bram_register_opt - (Optional) Improve critical path delay by moving registers from slices to block RAMs, or from block RAMs to slices.

-hold_fix - (Optional) Performs optimizations to insert data path delay to fix hold time violations.

-retime - (Optional) Re-time registers forward and backward through combinational logic to balance path delays.

-directive *arg* - (Optional) Direct the mode of physical optimization with specific design objectives. Only one directive can be specified for a single **phys_opt_design** command, and values are case-sensitive. Supported values include:

- **Explore** - Run different algorithms in multiple passes of optimization, including hold violation fixing and replication for very high fanout nets.
- **AggressiveExplore** - Similar to Explore but with different optimization algorithms and more aggressive goals.
- **AlternateReplication** - Use different algorithms for performing critical cell replication.
- **AggressiveFanoutOpt** - Uses different algorithms for fanout-related optimizations with more aggressive goals.
- **AlternateDelayModeling** - Performs all optimizations using alternate algorithms for estimating net delays.
- **AddRetime** - Performs the default **phys_opt_design** flow and adds register re-timing.
- **Default** - Run **phys_opt_design** with default settings.

Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on the effects of each directive.

Note The **-directive** option controls the overall optimization strategy, and is not compatible with any specific optimization options. It can only be used with **-quiet** and **-verbose**

-force_replication_on_nets *args* - (Optional) Force the drivers of the specified nets to be replicated, regardless of timing slack. Replication is based on load placements and requires manual analysis to determine if replication is sufficient. If further replication is required, nets can be replicated repeatedly by successive commands.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example performs a physical optimization of the open design:

```
phys_opt_design
```

This example performs register re-timing, and optimization of registers across DSP blocks and block RAMs:

```
phys_opt_design -retime -dsp_register_opt -bram_register_opt
```

This example directs **phys_opt_design** to run more iterations to achieve potentially better results:

```
phys_opt_design -directive Explore
```

This example directs **phys_opt_design** to consider more nets for replication:

```
phys_opt_design -directive AggressiveFanoutOpt
```

See Also

- [opt_design](#)
- [place_design](#)
- [power_opt_design](#)
- [route_design](#)

place_cell

Move or place one or more instances to new locations. Sites and cells are required to be listed in the right order and there should be same number of sites as number of cells.

Syntax

```
place_cell [-quiet] [-verbose] cell_site_list ...
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>cell_site_list</code>	a list of cells and sites in the interleaved order

Categories

[Floorplan](#)

Description

Places cells onto device resources of the target part. Cells can be placed onto specific BEL sites (e.g. **SLICE_X49Y60/A6LUT**), or into available SLICE resources (e.g. **SLICE_X49Y60**). If you specify the SLICE but not the BEL the tool will determine an appropriate BEL within the specified SLICE if one is available.

When placing a cell onto a specified site, the site must not be currently occupied, or an error will be returned: "Cannot set site and bel property of instances. Site SLICE_X49Y61 is already occupied."

You can test if a site is occupied by querying the IS_OCCUPIED property of a BEL site:

```
get_property IS_OCCUPIED [get_bels SLICE_X48Y60/D6LUT]
```

Note The IS_OCCUPIED property of a SLICE only tells you if some of the BELs within the SLICE are occupied; not whether or not the SLICE is fully occupied.

This command can be used to place cells, or to move placed cells from one site on the device to another site. The command syntax is the same for placing an unplaced cell, or moving a placed cell.

When moving a placed cell, if you specify only the SLICE for the site, the tool will attempt to place the cell onto the same BEL site in the new SLICE as it currently is placed. For instance moving a cell from the B6LUT, by specifying a new SLICE, will cause the tool to attempt to place the cell onto the B6LUT in the new SLICE. If this BEL site is currently occupied, an error is returned.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

cell_site_list - (Required) Specifies a list of cells and sites as {*cell_name site*}. The cell name is listed first, followed the BEL site or SLICE to place the cell onto. If the site is specified as a SLICE, the tool will select an available BEL within the SLICE. Multiple cells can be placed onto multiple sites by repeating the cell/site pair multiple times as needed:

{*cell_name1 site1 cell_name2 site2 cell_name3 site3 ... cell_nameN siteN*}.

Examples

The following example places the specified cell onto the specified BEL site:

```
place_cell div_cntr_reg_inferredi_4810_15889 SLICE_X49Y60/D6LUT
```

The following example places the specified cell into the specified SLICE:

```
place_cell div_cntr_reg_inferredi_4810_15889 SLICE_X49Y61
```

Note The tool will select an appropriate BEL site if one is available. If no BEL is available, and error will be returned

The following example places multiple cells onto multiple sites:

```
place_cell { \
cpuEngine/cpu_iwb_adr_o/buffer_fifo/i_4810_17734 SLICE_X49Y60/A6LUT \
cpuEngine/or1200_cpu/or1200_mult_mac/i_4775_15857 SLICE_X49Y60/B6LUT \
cpuEngine/cpu_iwb_adr_o/buffer_fifo/xlnx_opt_LUT_i_4810_18807_2 SLICE_X49Y60/C6LUT }
```

See Also

- [create_cell](#)
- [remove_cell](#)
- [unplace_cell](#)

place_design

Automatically place ports and leaf-level instances.

Syntax

```
place_design [-directive arg] [-no_timing_driven] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-directive]</code>	Mode of behavior (directive) for this command. Please refer to Arguments section of this help for values for this option. Default: Default
<code>[-no_timing_driven]</code>	Do not run in timing driven mode
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Tools](#)

Description

Place the ports and logic instances in the current design onto device resources on the target part. The tool optimizes placement to minimize negative timing slack and reduce overall wire length, while also attempting to spread out placement to reduce routing congestion.

Placement is one step of the complete design implementation process, which can be run automatically through the use of the **launch_runs** command when running the Vivado tools in Project Mode.

In Non-Project Mode, the implementation process must be run manually with the individual commands: **opt_design**, **place_design**, **phys_opt_design**, **power_opt_design**, and **route_design**. Refer to the *Vivado Design Suite User Guide: Design Flows Overview (UG892)* for a complete description of Project Mode and Non-Project Mode.

Both placement and routing can be completed incrementally, based on prior results stored in a Design Checkpoint file (DCP), using the incremental compilation flow. Refer to the **read_checkpoint** command, or to *Vivado Design Suite User Guide: Implementation (UG904)* for more information on incremental place and route.

You can also manually place some elements of the design using **place_ports**, or by setting LOC properties on the cell, and then automatically place the remainder of the design using **place_design**.

This command requires an open synthesized design, and it is recommended that you run the **opt_design** command prior to running **place_design** to avoid placing a suboptimal netlist.

Arguments

-directive *arg* - (Optional) Direct placement to achieve specific design objectives. Only one directive can be specified for a single **place_design** command, and values are case-sensitive. Supported values include:

- **Explore** - Increased placer effort in detail placement and post-placement optimization.
- **WLDDrivenBlockPlacement** - Wirelength-driven placement of RAM and DSP blocks. Override timing-driven placement by directing the Vivado placer to minimize the distance of connections to and from blocks.
- **LateBlockPlacement** - Defer detailed placement of RAMB and DSP blocks to the final stages of placement. Normally blocks are committed to valid sites early in the placement process. Instead, the placer uses coarse block placements that may not align with proper columns, then places blocks at valid sites during detail placement.
- **ExtraNetDelay_high** - Increases estimated delay of high fanout and long-distance nets. Three levels of pessimism are supported: high, medium, and low. **ExtraNetDelay_high** applies the highest level of pessimism.
- **ExtraNetDelay_medium** - Increases estimated delay of high fanout and long-distance nets. Three levels of pessimism are supported: high, medium, and low. **ExtraNetDelay_medium** applies the default level of pessimism.
- **ExtraNetDelay_low** - Increases estimated delay of high fanout and long-distance nets. Three levels of pessimism are supported: high, medium, and low. **ExtraNetDelay_low** applies the lowest level of pessimism.
- **SpreadLogic_high** - Distribute logic across the device. Three levels are supported: high, medium, and low. **SpreadLogic_high** achieves the highest level of distribution.
- **SpreadLogic_medium** - Distribute logic across the device. Three levels are supported: high, medium, and low. **SpreadLogic_medium** achieves a nominal level of distribution.
- **SpreadLogic_low** - Distribute logic across the device. Three levels are supported: high, medium, and low. **SpreadLogic_low** achieves a minimal level of distribution.
- **ExtraPostPlacementOpt** - Increased placer effort in post-placement optimization.
- **SSI_ExtraTimingOpt** - Use an alternate algorithm for timing-driven partitioning across SLRs.
- **SSI_SpreadSLLs** - Partition across SLRs and allocate extra area for regions of higher connectivity.
- **SSI_BalanceSLLs** - Partition across SLRs while attempting to balance SLLs between SLRs.
- **SSI_BalanceSLRs** - Partition across SLRs to balance number of cells between SLRs.
- **SSI_HighUtilSLRs** - Direct the placer to attempt to place logic closer together in each SLR.
- **RuntimeOptimized** - Run fewest iterations, trade higher design performance for faster runtime.
- **Quick** - Absolute, fastest runtime, non-timing-driven, performs the minimum required placement for a legal design.
- **Default** - Run **place_design** with default settings.

Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on placement strategies and the **-directive** option.

Note The **-directive** option controls the overall placement strategy, and is not compatible with any specific **place_design** options. It can only be used with **-quiet** and **-verbose**. In addition, the **-directive** option is ignored if the design is using the incremental compilation flow as defined by **read_checkpoint -incremental**.

-no_timing_driven - (Optional) Disables the default timing driven placement algorithm. This results in a faster placement based on wire lengths, but ignores any timing constraints during the placement process.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example places the current design:

```
place_design
```

The following example directs the Vivado placer to try different placement algorithms to achieve a better placement result:

```
place_design -directive Explore
```

See Also

- [launch_runs](#)
- [opt_design](#)
- [place_ports](#)
- [phys_opt_design](#)
- [power_opt_design](#)
- [read_checkpoint](#)
- [route_design](#)
- [set_property](#)

place_pblocks

Run the pblocks Placer.

Syntax

```
place_pblocks [-effort arg] [-utilization arg] [-quiet]  
[-verbose] pblocks...
```

Returns

Nothing

Usage

Name	Description
<code>[-effort]</code>	Placer effort level (per pblock) Values: LOW, MEDIUM, HIGH Default: HIGH
<code>[-utilization]</code>	Placer utilization (per pblock)
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>pblocks</code>	List of pblocks to place

Categories

[Floorplan](#)

Description

Places Pblocks onto the fabric of the FPGA. Pblocks must be created using the `create_pblock` command, and should be populated with assigned logic using the `add_cells_to_pblock` command.

Note An empty Pblock will be placed as directed, but results in a Pblock covering a single CLB tile (two SLICES).

Arguments

-effort *arg* - (Optional) Effort level that the Pblock placer should use in placing each Pblock onto the fabric. Valid values are LOW, MEDIUM, HIGH, with the default being HIGH.

-utilization *arg* - (Optional) Percentage of device resources that should be consumed by the logic elements assigned to a Pblock when it is placed onto the FPGA. For instance, a utilization rate of 50% means that half of the resources should be allocated to the logic in the Pblock, and half should be left for other design elements to be intermingled. A high utilization rate makes the Pblock smaller but more difficult to place, while a smaller utilization makes the Pblock larger.

Note Pblock utilization is post-synthesis estimation. Actual results may be different, and may require you to resize the Pblock using the **resize_pblock** command.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pblocks - (Required) One or more Pblocks to be placed onto the fabric of the FPGA.

Examples

The following example places the specified Pblocks with a utilization of 75%:

```
place_pblocks -effort LOW -utilization 75 block1 block2 block3 block4 block5
```

See Also

- [add_cells_to_pblock](#)
- [create_pblock](#)
- [resize_pblock](#)

place_ports

Automatically place a set of ports.

Syntax

```
place_ports [-skip_unconnected_ports] [-check_only] [-iobank args]
[-quiet] [-verbose] [ports...]
```

Returns

Nothing

Usage

Name	Description
<code>[-skip_unconnected_ports]</code>	Do not place unconnected ports
<code>[-check_only]</code>	Only check IO/Clock placement DRCs
<code>[-iobank]</code>	Limit placement to the following banks
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[ports]</code>	Ports to place (if omitted, all ports will be placed)

Categories

[PinPlanning](#)

Description

Automatically places ports on an available I/O or clocking site, or into the specified I/O banks.

The `place_ports` command will not replace ports that are currently placed by the user, or placed and fixed,

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-skip_unconnected_ports - (Optional) Do not place unconnected ports.

-check_only - (Optional) Run the clock placer DRCs, which are also available PLCK checks in the `report_drc` command. This option does not result in ports being placed, only checked for valid placement.

-iobank *args* - (Optional) Place the specified ports into the listed IO bank objects. IO bank objects are returned by the **get_iobanks** command.

Note Limiting port placement to specific IO banks will result in a placement error if there are insufficient placement sites for the number of ports being placed.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

ports - (Optional) The names of the ports to be placed. If no *ports* are specified, all ports will be placed.

Note If previously placed ports are specified, or included in the list of ports to place, the Vivado tool will not replace or move those ports.

Examples

The following example places the port objects returned by the **get_ports** command, onto I/O bank 13 of the device, as returned by **get_iobanks**:

```
place_ports -iobank [get_iobanks 13] [get_ports DataOut_pad_1_o]
```

The following example places all input ports onto I/O banks 12, 13, 14 and 15 of the device:

```
place_ports -iobank [get_iobanks {12 13 14 15}] [all_inputs]
```

See Also

- [create_port](#)
- [get_iobanks](#)
- [make_diff_pair_ports](#)
- [remove_port](#)

power_opt_design

Optimize dynamic power using intelligent clock gating.

Syntax

```
power_opt_design [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Power](#)

Description

Optimizes the dynamic power consumption of the design by changing clock gating to take advantage of clock enable on a flop. Clock gating optimizations are automatically performed on the entire design to improve power consumption while making no changes to the existing logic or the clocks that would alter the behavior of the design.

You can configure the power optimization to include or exclude specific cells using the **set_power_opt** command.

Run power optimization after synthesis, or after placement. When run before placement, this command optimizes the design to save power. When run after placement, this command optimizes the design to save power while preserving timing. Running after placement limits the optimizations available to the **power_opt_design** command. To achieve the best results, the command should be run prior to placement.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example performs power optimization of the open design:

```
power_opt_design
```

See Also

- [report_power](#)
- [report_power_opt](#)
- [set_power_opt](#)

pr_verify

Verify the static logics in two DCP files.

Syntax

```
pr_verify [-quiet] [-verbose] file1 file2
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file1</i>	DCP file one
<i>file2</i>	DCP file two

Categories

[FileIO](#)

program_hw_devices

Program hardware devices.

Syntax

```
program_hw_devices [-quiet] [-verbose] [hw_device ...]
```

Returns

Hardware devices

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_device]	list of hardware devices Default: current hardware device

Categories

[Hardware](#)

ptrace

Turns on or off printing of name of the hdl process being simulated.

Syntax

```
ptrace [-quiet] [-verbose] value
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>value</i>	value: on, true, yes. Otherwise set to off, false, no

Categories

[Simulation](#)

read_checkpoint

Read a design checkpoint.

Syntax

```
read_checkpoint [-incremental] [-part arg] [-quiet] [-verbose] file
```

Returns

Nothing

Usage

Name	Description
[-incremental]	Input design checkpoint file to be used for re-using implementation.
[-part]	Override the checkpoint part. Note that this may cause errors if the checkpoint contains xdef. Ignored with -cell
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Design checkpoint file

Categories

[FileIO](#)

Description

Reads a design checkpoint file (DCP) that contains the netlist, constraints, and may optionally have the placement and routing information of an implemented design. You can save and restore design checkpoints at any stage in the design.

When reading a checkpoint, there is no need to create a project first. The **read_checkpoint** command reads the design data into memory, opening the design in Non-Project Mode. Refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) for more information on Project Mode and Non-Project Mode.

You can also import a design checkpoint, and retarget the design to a new Xilinx device by specifying a new target part.

Note When multiple design checkpoints are open in the Vivado tool, you must use the **current_project** command to switch between the open designs. You can use **current_design** to check which checkpoint is the active design

Arguments

-incremental *arg* - (Optional) Load a checkpoint file into an already open design to enable the incremental compilation design flow, where *arg* specifies the path and filename of the incremental design checkpoint (DCP) file. In the incremental compilation flow, the cell placement and net routing from the incremental design is matched with existing design objects and applied to the in-memory database. After loading an incremental design checkpoint, you can use the **report_incremental_reuse** command to determine the percentage of physical data reused from the incremental checkpoint, in the current design. The **place_design** and **route_design** commands will run incremental place and route, preserving reused placement and routing information and incorporating it into the design solution.

Note Reading a design checkpoint with **-incremental**, loads the physical data into the current in-memory design. To clear out the incremental design data, you must either reload the synthesized design, using **open_run**, or load a new incremental design to overwrite the one previously loaded. Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on incremental place and route.

-part *arg* - (Optional) A target part for the imported checkpoint design. This is useful when importing a checkpoint file that has a netlist and constraints. However, you can also use the **-part** argument to change the target part for a checkpoint file with placement and routing information for an implemented design. This will usually result in an error after the netlist and constraints have been read.

A subsequent `read_checkpoint -incremental` command will replace physical data reuse from the previous incremental checkpoint. If it is not desired to run the incremental place and route flow after loading an incremental checkpoint, then the original checkpoint must be reloaded.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The path and filename of the checkpoint file.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example imports the specified checkpoint file, and specifies the target part for the design:

```
read_checkpoint C:/Data/state1/checkpoint.dcp -part xc7k325tffg900-2
```

See Also

- [current_design](#)
- [current_project](#)
- [write_checkpoint](#)

read_csv

Import package pin and port placement information.

Syntax

```
read_csv [-quiet_diff_pairs] [-quiet] [-verbose] file
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet_diff_pairs]</code>	Suppress warnings about differential pair inference when importing I/O ports
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>file</code>	Pin Planning CSV file

Categories

[FileIO](#)

Description

Imports port definition and package pin placement information from a comma separated value (CSV) file.

The port definitions in a CSV file can be imported into an I/O Pin Planning project. In a Pin Planning project, importing a CSV file replaces the current port definitions. Any ports in the design that are not found in the imported CSV file will be removed.

In all other projects the port definitions are defined in the source design data, however package pin assignments and port attributes can be read from the specified CSV file.

The ports read from the CSV file can not have spaces in the name, or the tool will return an error. The specific format and requirements of the CSV file are described in the *Vivado Design Suite User Guide: I/O and Clock Planning (UG899)*.

Arguments

-quiet_diff_pairs - (Optional) The tool transcripts messages related to pins that may be inferred as differential pairs when importing the CSV file. This option suppresses messages related to inferring differential pairs.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The file name of the CSV file to be imported.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example imports a CSV file into an open project:

```
read_csv C:/Data/pinList.csv
```

The following example sets up a new IO Pin Planning project, and then imports the specified CSV file into it, and infers any differential pairs in the CSV file:

```
create_project myPinPlan C:/Data/myPinPlan -part xc7v285tffg1157-1
set_property design_mode PinPlanning [current_fileset]
open_io_design -name io_1
read_csv C:/Data/import.csv
infer_diff_pairs -filetype csv C:/Data/import.csv
```

Note The design_mode property on the source fileset is what determines the nature of the project.

See Also

- [create_project](#)
- [infer_diff_pairs](#)
- [open_io_design](#)
- [set_property](#)
- [write_csv](#)

read_edif

Read one or more EDIF or NGC files.

Syntax

```
read_edif [-quiet] [-verbose] files
```

Returns

List of file objects that were added

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>files</i>	EDIF or NGC file name(s)

Categories

[FileIO](#)

Description

Imports an EDIF or NGC netlist file into the Design Source fileset of the current project.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Required) The name of the EDIF or NGC files to be imported.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example imports an EDIF file into the open project:

```
read_edif C/Data/bft_top.edf
```

See Also

[write_edif](#)

read_hw_ila_data

Read hardware ILA data from a file.

Syntax

```
read_hw_ila_data [-quiet] [-verbose] file
```

Returns

Hardware ILA data object

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>file</i>	hardware ILA data file name

Categories

[Hardware](#)

read_hw_sio_scan

Read hardware SIO scan data from a file. A hardware SIO scan object will be created if not provided.

Syntax

```
read_hw_sio_scan [-quiet] [-verbose] file [hw_sio_scan]
```

Returns

Hardware SIO scan object

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	hardware SIO scan file name
<i>[hw_sio_scan]</i>	hardware SIO scan data object Default: None

Categories

[Hardware](#)

read_ip

Read one or more IP files.

Syntax

```
read_ip [-quiet] [-verbose] files
```

Returns

List of IP file objects that were added

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>files</i>	IP file name(s)

Categories

[FileIO](#), [IPFlow](#)

Description

Read the specified list of IP files and add them to the design and the current fileset.

Files are added by reference into the current project, just as in the **add_files** command.

You can use this command to read the contents of source files into the in-memory design, when running the Vivado tool in Non Project mode, in which there is no project file to maintain and manage the various project source files. Refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) for more information on Non Project mode.

Use the **import_ip** command to add the IP cores and import the files into the local project directory.

The command returns the list of files read.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Required) The list of IP files to read into the current project. Both XCI and XCO file formats are supported. An XCI file is an IP-XACT format file that contains information about the IP parameterization. An XCO file is a CORE Generator log file that records all the customization parameters used to create the IP core and the project options in effect when the core was generated.

Examples

The following example reads the specified IP files:

```
read_ip C:/test_ip/char_fifo.xci
```

See Also

- [add_files](#)
- [import_ip](#)

read_saif

Import simulation data in saif format.

Syntax

```
read_saif [-strip_path arg] [-no_strip] [-out_file arg] [-quiet]  
[-verbose] file
```

Returns

Nothing

Usage

Name	Description
<code>[-strip_path]</code>	Specifies the name of the instance of the current design as it appears in the SAIF file
<code>[-no_strip]</code>	Do not strip first two levels of hierarchy from SAIF file
<code>[-out_file]</code>	Specifies the name of the output file that contains nets that could not be matched
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>file</i>	Specifies the name of the SAIF file to be read

Categories

[FileIO](#), [Power](#), [Simulation](#)

Description

Reads a Switching Activity Interchange Format (SAIF) file for use during power analysis by the **report_power** command. Running **report_power** after reading the SAIF file will use the activity rates from the specified file.

You can also read switching activity in the form of a VCD file using the **read_vcd** command.

Arguments

-strip_path *arg* - (Optional) Strip the specified instance path prefix from elements in the SAIF file to allow them to be mapped properly to instances in the current design.

-no_strip - (Optional) Do not strip first two levels of hierarchy from the SAIF file.

-out_file *arg* - (Optional) The name of an output file where unmatched nets and other messages are reported. This file is created during the import of the SAIF file. If the **-out_file** option is not specified, the information is not saved to a file.

Note If the path is not specified as part of the file name, the tool will write the specified file to the current working directory, or the directory from which the tool was launched.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The name of the SAIF file to read.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example:

```
read_saif -strip_path /design//top/F1 C:/Data/design1.saif
```

See Also

- [read_vcd](#)
- [report_power](#)

read_twx

Read timing results from Trace STA tool.

Syntax

```
read_twx [-cell arg] [-pblock arg] [-quiet] [-verbose] name file
```

Returns

Nothing

Usage

Name	Description
[-cell]	Interpret names in the report file as relative to the specified cell
[-pblock]	Interpret names in the report file as relative to the specified pblock
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name for the set of results
<i>file</i>	Name of the Trace import file

Categories

[FileIO](#)

Description

Imports timing results in the TWX format timing report files generated by the Xilinx Timing Reporter And Circuit Evaluator (TRACE) tool. The TWX file can be imported at the top-level, which is the default, or at a specific cell-level or relative to a specific Pblock.

After the TWX files are imported, the timing results display in the Timing Results view in GUI mode.

Arguments

-cell *arg* - (Optional) Specify The name of a hierarchical cell in the current design to import the TWX file into. The timing paths will be applied to the specified cell.

-pblock *arg* - (Optional) The name of a Pblock in the current design. The timing paths will be imported relative to the specified block.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the Timing Results view to create when importing the timing paths in the TWX file.

Note Both *name* and *file* are required positional arguments. The *name* argument must be provided first.

file - (Required) The file name of the TWX file to be imported.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example reads the specified TWX file into the top-level of the design:

```
read_twx C:/Data/timing_files/bft.twx
```

See Also

[report_timing](#)

read_vcd

Import simulation data in vcd format.

Syntax

```
read_vcd [-strip_path arg] [-no_strip] [-out_file arg] [-quiet]  
[-verbose] file
```

Returns

Nothing

Usage

Name	Description
<code>[-strip_path]</code>	Specifies the name of the instance of the current design as it appears in the VCD file
<code>[-no_strip]</code>	Do not strip first two levels of hierarchy from VCD file
<code>[-out_file]</code>	Specifies the name of the output file that contains nets that could not be matched
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>file</code>	Specifies the name of the VCD file to be read

Categories

[FileIO](#), [Power](#), [Simulation](#)

Description

Read a Value Change Dump (VCD) file for use during power analysis by the **report_power** command. The **read_vcd** command will annotate the design nodes with activity from the VCD file. Running **report_power** after reading the VCD file will use the activity rates from the specified file.

Note The VCD file can be written by the Vivado simulator using the **open_vcd**, **log_vcd**, and **flush_vcd** commands. However, the VCD file cannot be read into the Vivado simulator using the **read_vcd** command

You can also read switching activity in the form of an SAIF file using the **read_saif** command.

Arguments

-strip_path *arg* - (Optional) Strip the specified instance path prefix from elements in the VCD file to allow them to be mapped properly to instances in the current design.

-no_strip - (Optional) Do not strip first two levels of hierarchy from the VCD file.

-out_file *arg* - (Optional) Specify the name of an output file where unmatched nets and other messages are reported. This file is created during the import of the VCD file. If the **-out_file** option is not specified, the information is not saved to a file.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The name of the VCD file to read.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example reads a VCD file named design1:

```
read_vcd C:/Data/design1.vcd
```

See Also

- [flush_vcd](#)
- [log_vcd](#)
- [open_vcd](#)
- [read_saif](#)
- [report_power](#)

read_verilog

Read one or more Verilog files.

Syntax

```
read_verilog [-library arg] [-sv] [-quiet] [-verbose] files...
```

Returns

List of file objects that were added

Usage

Name	Description
[-library]	Library name Default: work
[-sv]	Enable system verilog compilation
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>files</i>	Verilog file name(s)

Categories

[FileIO](#)

Description

Reads Verilog or SystemVerilog source files. This command is similar to the **add_files** command. The Verilog file is added to the source fileset as it is read. If the **-library** argument is specified, the file is added with the Library property defined appropriately.

You can use this command to read the contents of source files into the in-memory design, when running the Vivado tool in Non Project mode, in which there is no project file to maintain and manage the various project source files. Refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) for more information on Non Project mode.

Because SystemVerilog is a superset of the Verilog language, the **read_verilog** command can read both file types. However, for SystemVerilog files, the **-sv** option needs to be specified for **read_verilog** to enable compilation in the SystemVerilog mode. In this mode, the tool recognizes and honors the SystemVerilog keywords and constructs.

You can have a mixture of both Verilog files (.v files), and SystemVerilog files (.sv files), as well as VHDL (using **read_vhdl**). When the tool compiles these files for synthesis, it creates separate "compilation units" for each file type. All files of the same type are compiled together.

Arguments

-library *arg* - (Optional) The library the Verilog file should reference. The default Verilog library is work.

-sv - (Optional) Read the files as a SystemVerilog compilation group.

Note Since Verilog is a subset of SystemVerilog, unless a Verilog source has user-defined names that collide with reserved SystemVerilog keywords, reading Verilog files with the **-sv** switch enables SystemVerilog compilation mode for those files. However, adding a SystemVerilog file in a Verilog compilation unit (without **-sv**) will not work.

files - (Required) The name of one or more Verilog files to be read.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reads the specified Verilog file and adds it to the source files:

```
read_verilog C:/Data/FPGA_Design/new_module.v
```

The following example creates two compilation units, one for SystemVerilog files and one for Verilog files:

```
read_verilog -sv { file1.sv file2.sv file3.sv }  
read_verilog { file1.v file2.v file3.v }
```

See Also

- [add_files](#)
- [read_vhdl](#)

read_vhdl

Read one or more VHDL files.

Syntax

```
read_vhdl [-library arg] [-quiet] [-verbose] files
```

Returns

List of file objects that were added

Usage

Name	Description
[-library]	VHDL library Default: work
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>files</i>	VHDL file name(s)

Categories

[FileIO](#)

Description

Reads a VHDL source file. This command is similar to the **add_files** command. The VHDL file is added to the source fileset as it is read. If the **-library** argument is specified, the file is added with the Library property defined appropriately.

You can use this command to read the contents of source files into the in-memory design, when running the Vivado tool in Non Project mode, in which there is no project file to maintain and manage the various project source files. Refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) for more information on Non Project mode.

Arguments

-library *arg* - (Optional) The library the VHDL file should reference. The default VHDL library is work.

file - (Required) Filename of the VHDL file to be read.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reads the specified VHDL file and adds it to the source fileset:

```
read_vhdl C:/Data/FPGA_Design/new_module.vhdl
```

See Also

[add_files](#)

read_xdc

Read physical and timing constraints from one or more files.

Syntax

```
read_xdc [-cells args] [-ref arg] [-quiet_diff_pairs] [-quiet]
[-verbose] files
```

Returns

List of files

Usage

Name	Description
<code>[-cells]</code>	Import constraints for these cells
<code>[-ref]</code>	Import constraints for this ref
<code>[-quiet_diff_pairs]</code>	Suppress warnings about differential pair inference when importing I/O ports
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>files</code>	Input file(s) to read

Categories

FileIO

Description

Imports physical constraints from a Xilinx Design Constraints file (XDC). The XDC can be imported at the top-level, which is the default, or applied to specific cells, or to instances of a specific cell. When imported at the top-level, the specified XDC file is added to the active constraint fileset.

Note Constraints from the XDC file will overwrite any current constraints of the same name. Therefore, exercise some caution when reading a XDC file to be sure you will not overwrite important constraints.

This command is similar to the **add_files** command in that the XDC file is added by reference rather than imported into the local project directory.

You can use this command to read the contents of source files into the in-memory design, when running the Vivado tool in Non Project mode, in which there is no project file to maintain and manage the various project source files. Refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) for more information on Non Project mode.

Arguments

-ref *arg* - (Optional) Read the constraints from the XDC file and apply them to ALL instances of the referenced module, wherever they happen to be instantiated in the current design.

-cells *args* - (Optional) Apply the constraints from the XDC file to the specified instance names. The constraints will be applied ONLY to the specified cell instances, and the XDC file will not be added to the active constraint fileset.

Note A design must be open when specifying the -cells option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The filename of the XDC file to be imported.

Note If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

Examples

The following example reads the XDC file and applies it to the current design:

```
read_xdc file_1.xdc
```

The following example reads the XDC file and applies it ALL instances of the referenced module found in the current design:

```
read_xdc -ref hex2led file_2.xdc
```

The following example reads the XDC file and applies it ONLY to the specified instance within the referenced module:

```
read_xdc -ref sixty -cells lsbcount file_3.xdc
```

The following example reads the XDC file and applies it to the specified instances in the current design, even though they are instances of different modules:

```
read_xdc -cells one_decode sixty/msbcount file_4.xdc
```

See Also

- [add_files](#)
- [infer_diff_pairs](#)
- [write_xdc](#)

redo

Re-do previous command.

Syntax

```
redo [-list] [-quiet] [-verbose]
```

Returns

With -list, the list of redoable tasks

Usage

Name	Description
<code>[-list]</code>	Show a list of redoable tasks
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[GUIControl](#)

Description

Redo a command that has been previously undone. This command can be used repeatedly to redo a series of commands.

If a command group has been created using the **startgroup** and **endgroup** commands, the redo command will redo the group of commands as a sequence.

Arguments

-list - (Optional) Get the list of commands that can be redone. When you use the **undo** command, the tool will step backward through a list of commands. The **redo** command can then be used to redo those commands.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns a list of commands that can be redone:

```
redo -list
```

See Also

- [undo](#)
- [startgroup](#)
- [endgroup](#)

refresh_design

Refresh the current design.

Syntax

```
refresh_design [-part arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-part]	Target part
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Project

Description

Reloads the current design from the project data on the hard drive. This overwrites the in-memory view of the design to undo any recent design changes.

Arguments

-part *arg* - (Optional) The new target part for the design when it is reloaded. This overrides the constraint file part specified in the project data on the hard drive.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following command reloads the current design from the project data on hard disk. This will overwrite the unsaved changes of the design which are in memory.

```
refresh_design
```

Note You can use the command to undo a series of changes to the design and revert to the previously saved design.

The following example refreshes the current design using the specified V6 part as the target device. The second command is required to make the selected part the target device for the active implementation run.

```
refresh_design -part xc6vcx75tff784-1  
set_property part xc6vcx75tff784-1 [get_runs impl_6]
```

Note The second command is not required if the target part is not changed.

See Also

[set_property](#)

refresh_hw_device

Refresh a hardware device. Read device and core information from device.

Syntax

```
refresh_hw_device [-update_hw_probes arg] [-quiet] [-verbose]  
[hw_device]
```

Returns

Nothing

Usage

Name	Description
<code>[-update_hw_probes]</code>	Update hardware probe information, read from probes file
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[hw_device]</code>	hardware device Default: current hardware device

Categories

[Hardware](#)

refresh_hw_server

Refresh a connection to a hardware server.

Syntax

```
refresh_hw_server [-quiet] [-verbose] [hw_server]
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[hw_server]</code>	hardware server

Categories

[Hardware](#)

refresh_hw_sio

Refresh the status of the specified hardware objects. Inputs can be any hardware object. At least one object is required. If properties are specified that do not exist in the object, that property will not be refreshed.

Syntax

```
refresh_hw_sio [-regexp] [-properties args] [-quiet]  
[-verbose] hw_objects
```

Returns

Nothing

Usage

Name	Description
[-regexp]	Properties list contains full regular expressions
[-properties]	List of properties to refresh Default: All properties in object
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_objects</i>	hardware objects

Categories

[Hardware](#)

refresh_hw_target

Refresh a hardware target.

Syntax

```
refresh_hw_target [-quiet] [-verbose] [hw_target]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_target]	hardware target

Categories

[Hardware](#)

refresh_hw_vio

Update hardware probe INPUT_VALUE and ACTIVITY_VALUE properties with values read from hardware VIO core(s).

Syntax

```
refresh_hw_vio [-update_output_values arg] [-quiet] [-verbose]  
[hw_vios...]
```

Returns

Nothing

Usage

Name	Description
<code>[-update_output_values]</code>	Update hardware probe OUTPUT_VALUE property with values read from VIO core(s). Default is false.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[hw_vios]</code>	List of hardware VIO objects.

Categories

[Hardware](#)

regenerate_bd_layout

Regenerate layout.

Syntax

```
regenerate_bd_layout [-routing] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-routing]	Preserve placement of blocks and regenerate routing
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[IPIntegrator](#)

reimport_files

Reimport files when they are found out-of-date.

Syntax

```
reimport_files [-force] [-quiet] [-verbose] [files...]
```

Returns

List of file objects that were imported

Usage

Name	Description
[-force]	Force a reimport to happen even when the local files may be newer
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[files]</i>	List of files to reimport. If no files are specified, all files in the project that are out-of-date, will be reimported

Categories

[Project](#)

Description

Reimports project files. This updates the local project files from the original referenced source files.

Arguments

-force - (Optional) Reimport files even when the local project files may be newer than their referenced source files.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Optional) List of files to reimport. If no files are specified, all files in the project that are out-of-date, will be reimported. If you use **-force** and specify no files, all files in the project will be reimported.

Examples

The following example reimports all project files regardless of whether they are out of date, or the local files are newer than the referenced source file:

```
reimport_files -force
```

Note No warnings will be issued for newer local files that will be overwritten.

The following example reimports the specified files to the project, but only if the original source file is newer than the local project file:

```
reimport_files C:/Data/FPGA_Design/source1.v C:/Data/FPGA_Design/source2.vhdl
```

See Also

- [add_files](#)
- [import_files](#)

remove_bps

Remove breakpoints from a simulation.

Syntax

```
remove_bps [-all] [-file arg] [-line arg] [-quiet] [-verbose]  
[BreakPointObjsOrIds ...]
```

Returns

Nothing

Usage

Name	Description
[-all]	Remove all breakpoints
[-file]	The specific file to remove the breakpoint from given a line number
[-line]	The specific line number to remove the breakpoint given a filename Default: -1
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[BreakPointObjsOrIds]</i>	A list of one or more breakpoint objects and/or breakpoint object ID's to be removed

Categories

[Simulation](#)

remove_cell

Remove cells from the current design.

Syntax

```
remove_cell [-quiet] [-verbose] cells...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>cells</i>	List of cells to remove

Categories

Netlist

Description

Remove cells from the current netlist in either an open Synthesized or Implemented design.

Note You cannot remove cells from library macros, also called macro-primitives

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

cells - (Required) List of cells to remove. The instance name can be specified as a hierarchical name, from the top-level of the design. In this case, you must use the hierarchy separator character in the hierarchical instance name. You can determine the current hierarchy separator with the **get_hierarchy_separator** command.

Examples

The following example removes the fftEngine from the in-memory netlist of the current design:

```
remove_cell fftEngine  
remove_cell usbEngine0/usb_out
```

See Also

- [create_cell](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

remove_cells_from_pblock

Remove cells from a Pblock.

Syntax

```
remove_cells_from_pblock [-quiet] [-verbose] pblock cells...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>pblock</i>	Pblock to remove cells from
<i>cells</i>	Cells to remove

Categories

Floorplan, XDC

Description

Removes the specified logic instances from a Pblock. Cells are added to a Pblock with the **add_cells_to_pblock** command.

Note Cells that have been placed will not be unplaced as they are removed from a Pblock. Any current LOC assignments are left intact.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pblock - (Required) The name of the Pblock from which to remove the specified instances.

cells - (Required) One or more cell objects to remove from the specified Pblock.

Examples

The following example removes the specified cells from the pb_cpuEngine Pblock:

```
remove_cells_from_pblock pb_cpuEngine [get_cells cpuEngine/cpu_dwb_dat_o/*]
```

See Also

[add_cells_to_pblock](#)

remove_conditions

Remove conditions from a simulation. The names can be specified as Tcl glob pattern.

Syntax

```
remove_conditions [-all] [-quiet] [-verbose] [ConditionObjs]
```

Returns

Nothing

Usage

Name	Description
[-all]	Remove all conditions
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[ConditionObjs]</i>	ConditionObjs, id's or names

Categories

[Simulation](#)

remove_drc_checks

Remove drc rule check objects from a user rule deck.

Syntax

```
remove_drc_checks [-of_objects args] [-regexp] [-nocase] [-filter arg]
-ruledeck arg [-quiet] [-verbose] [patterns]
```

Returns

Drc_check

Usage

Name	Description
[-of_objects]	Get 'drc_rule' objects of these types: 'drc_ruledeck'.
[-regexp]	Patterns are full regular expressions
[-nocase]	Perform case-insensitive matching. (valid only when -regexp specified)
[-filter]	Filter list with expression
-ruledeck	DRC rule deck to modify
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[patterns]	Match the 'drc_rule' objects against patterns. Default: *

Categories

[DRC](#), [Object](#)

Description

Remove the specified design rule checks from a drc_ruledeck object.

A rule deck is a collection of design rule checks grouped for convenience, to be run with the **report_drc** command at different stages of the FPGA design flow, such as during I/O planning or placement. The tool comes with a set of factory defined rule decks, but you can also create new user-defined rule decks with the **create_drc_ruledeck** command.

Checks are added to a rule deck using the **add_drc_checks** command.

The DRC rule check object features the **IS_ENABLED** property that can be set to true or false using the **set_property** command. When a new rule check is created, the **IS_ENABLED** property is set to true as a default. Set the **IS_ENABLED** property to false to disable the rule check from being used by **report_drc** without having to remove the rule from the rule deck.

Tip Use the **reset_drc_check** command to restore the DRC rule, and its properties, to the default settings.

This command returns the list of design rule checks that were removed from the specified rule deck.

Arguments

-of_objects *arg* - (Optional) Remove the rule checks of the specified *drc_ruledeck* object from the specified rule deck. This has the effect of removing all the rules in one rule deck from the target rule deck.

Note **-of_objects** cannot be used with a search *pattern*

-regexp - (Optional) Specifies that the search *patterns* are written as regular expressions. Both search *patterns* and **-filter** expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add *.** to the beginning or end of a search string to widen the search. See http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm for help with regular expression syntax.

Note The Tcl built-in command **regexp** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexp.htm>.

-nocase - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

-filter *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by the search pattern, based on specified property values. You can find the properties on an object with the **report_property** or **list_property** commands.

The specific operators that can be used in filter expressions are "equals" and "not-equals" (== and !=), and "contains" and "not-contains" (=~ and !~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following example returns input ports that do NOT end with the "clk" substring:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*clk"}
```

Boolean (**bool**) type properties can be directly evaluated in filter expressions as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

-ruledck *arg* - (Required) The name of the rule deck to remove the specified design rule checks from.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

patterns - (Optional) Remove the design rule checks that match the specified patterns from the rule deck. The default pattern is the wildcard '*' which removes all rule checks from the specified rule deck. More than one pattern can be specified to remove multiple rule checks based on different search criteria.

Note You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples

The following example removes the rule checks matching the specified filter pattern from the my_rules rule deck:

```
remove_drc_checks -filter {GROUP == AVAL} -ruledock my_rules
```

The following example disables the specified DRC check without removing it from the rule deck:

```
set_property IS_ENABLED FALSE [get_drc_checks RAMW-1]
```

The following example removes all rule checks from the specified rule deck:

```
remove_drc_checks -ruledock my_rules
```

See Also

- [add_drc_checks](#)
- [create_drc_check](#)
- [create_drc_ruledock](#)
- [get_drc_checks](#)
- [get_drc_ruledocks](#)
- [list_property](#)
- [report_drc](#)
- [report_property](#)
- [reset_drc_check](#)

remove_files

Remove files or directories from a fileset.

Syntax

```
remove_files [-fileset arg] [-quiet] [-verbose] [files...]
```

Returns

List of files that were removed

Usage

Name	Description
[-fileset]	Fileset name
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>files</i>]	Name of the file(s) to be removed

Categories

[Project](#), [Simulation](#)

Description

Removes the specified file objects from the current or specified fileset. The file is removed from the current project, but is not removed from the disk.

Files can be specified as file name strings, or as file objects returned by the **get_files** command. When specified as strings, the file is looked for in the current or specified fileset. When the file object is specified by **get_files**, the fileset is defined by the object, and **-fileset** is ignored.

When successful, this command returns nothing. If the specified file is not found, an error is returned.

Arguments

-fileset *arg* - (Optional) The name of the fileset to locate the specified files. As a default, the files will be removed from the current fileset as defined by the **current_fileset** command.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Required) The name of the files to remove from the project.

Note If no files are specified, no files are removed.

Examples

The following example removes the file named C:/Design/top.xdc from the constraint set `constrs_1`:

```
remove_files -fileset constrs_1 C:/Design/top.xdc
```

Multiple files can be specified as follows:

```
remove_files -fileset sim_1 top_tb1.vhdl top_tb2.vhdl
```

The following example gets all the file objects in the current project, and removes them:

```
remove_files [get_files]
```

Important! This will remove ALL files from your design.

See Also

- [add_files](#)
- [current_fileset](#)
- [get_files](#)

remove_forces

Release force on signal, wire, or reg applied using 'add_force' command.

Syntax

```
remove_forces [-all] [-quiet] [-verbose] [ForceObj...]
```

Returns

Nothing

Usage

Name	Description
[-all]	Remove all forces
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[ForceObj]</i>	ForceObj or id's

Categories

[Simulation](#)

Description

Remove the specified force objects, or force IDs from the current simulation.

Forces are applied to specific HDL objects using the **add_force** command. This command removes those forces from the current simulation.

Important! If there are **force/release** statements on an HDL object in the test bench or module, these statements are overridden by the **add_force** command. When the **remove_force** command releases these objects to resume their normal operation, the Verilog **force/release** statements resume their effect

This command returns nothing if successful, or returns an error if it fails.

Arguments

-all - (Optional) Remove all forces from the current simulation.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

ForceObj - (Optional) Remove only the specified force object or objects. The force ID is returned by the `add_force` command when the force is created.

Examples

The following example creates a force object using the `add_force` command, and captures the force ID in a Tcl variable, then removes that force object:

```
set f10 [ add_force reset 1 300 ]  
remove_forces $f10
```

The following example removes all force objects from the current simulation:

```
remove_forces -all
```

See Also

- [get_objects](#)
- [add_force](#)

remove_hw_sio_link

Remove an existing hardware SIO link.

Syntax

```
remove_hw_sio_link [-quiet] [-verbose] hw_sio_links
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_sio_links</i>	hardware SIO links

Categories

[Hardware](#)

remove_hw_sio_linkgroup

Remove an existing hardware SIO link group.

Syntax

```
remove_hw_sio_linkgroup [-quiet] [-verbose] hw_sio_linkgroups
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_sio_linkgroups</i>	hardware SIO linkgroups

Categories

[Hardware](#)

remove_hw_sio_scan

Remove an existing hardware SIO scan.

Syntax

```
remove_hw_sio_scan [-quiet] [-verbose] hw_sio_scans
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_sio_scans</i>	hardware SIO scans

Categories

[Hardware](#)

remove_net

Remove nets from the current design.

Syntax

```
remove_net [-quiet] [-verbose] nets...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>nets</i>	List of nets to remove

Categories

[Netlist](#)

Description

Remove the specified net from the netlist of an open Synthesized or Implemented Design.

Note You cannot remove nets from library macros, also called macro-primitives

To remove a bus, you must specify the primary bus name, and not specify a bus index. This ensures that the entire bus is removed, and not just a portion of the bits associated with the bus.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

nets - (Required) The list of nets to remove from the netlist of the current design.

Example

The following example illustrates the warning returned when trying to remove one bit of a bus net, and then removes the entire bus by specifying the root name:

```
remove_net DataIn_pad_1_i[0]
WARNING: [Coretc1-82] No nets matched 'DataIn_pad_1_i[0]'.
remove_net DataIn_pad_1_i
```

See Also

- [create_net](#)
- [disconnect_net](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

remove_pin

Remove pins from the current design.

Syntax

```
remove_pin [-quiet] [-verbose] pins...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>pins</i>	List of pins to remove

Categories

Netlist

Description

Remove pins from the current netlist in either an open Synthesized or Implemented design.

Note You cannot remove pins from library macros, or macro-primitives.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pins - (Required) List of pins to remove from the netlist. The pins must be specified hierarchically by the cell instance the pin is found on.

Examples

The following example removes the fftEngine from the in-memory netlist of the current design:

```
remove_cell fftEngine
```

See Also

- [create_cell](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

remove_port

Remove the given list of top ports from the netlist.

Syntax

```
remove_port [-quiet] [-verbose] ports...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>ports</i>	Ports and/or bus ports to remove

Categories

[PinPlanning](#)

Description

Removes the specified ports or buses.

The **remove_port** command will remove ports that have been added with the **create_port** command, but cannot delete ports that are defined in the RTL or netlist design.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

ports - One or more names of ports to remove.

Examples

The following example deletes the specified port:

```
remove_port PORT0
```

The following example deletes the two specified ports of a bus:

```
remove_port BUS[1] BUS[2]
```

The following example deletes both the N and P sides of a differential pair port:

```
remove_port D_BUS_P[0]
```

Note Deleting either the N or the P side of a differential pair will also delete the other side of the pair.

See Also

- [create_port](#)
- [create_interface](#)
- [place_ports](#)

rename_ref

Rename a cell ref.

Syntax

```
rename_ref [-ref arg] [-to arg] [-prefix_all arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-ref]	Cell ref to rename
[-to]	New name
[-prefix_all]	Rename all eligible hierarchical cell refs in the current design. Construct the new name using the given prefix plus the original name
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Netlist

Description

Rename the reference name of a single non-primitive cell, or apply a reference prefix to all non-primitive cells in the current synthesized or implemented design.

This command provides a mechanism to change the non-primitive reference names in the current design so that they do not collide with the reference names in another design. This lets two modules or designs be synthesized or simulated together, while avoiding any name collisions between the two designs.

This command returns nothing when renaming the reference a single cell, and returns the number of cells renamed when used with **-prefix_all**. If the command fails, an error is returned.

Arguments

-ref *arg* - (Optional) Specify the current reference name of a non-primitive cell.

-to *arg* - (Optional) Change the reference name to the specified value.

-prefix_all *arg* - (Optional) Apply the specified prefix to the reference names of all non-primitive cells in the current design, except the top module.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example changes the specified reference name to the value indicated:

```
rename_ref -ref usbf_top -to MOD1_usbf_top
```

The following example applies the specified reference name prefix to all non-primitive cells in the current design:

```
rename_ref -prefix_all MOD1_
```

See Also

- [launch_xsim](#)
- [synth_design](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

reorder_files

Change the order of source files in the active fileset.

Syntax

```
reorder_files [-fileset arg] [-before arg] [-after arg] [-front]
[-back] [-auto] [-disable_unused] [-quiet] [-verbose] files...
```

Returns

Nothing

Usage

Name	Description
[-fileset]	Fileset to reorder
[-before]	Move the listed files before this file
[-after]	Move the listed files after this file
[-front]	Move the listed files to the front (default)
[-back]	Move the listed files to the back
[-auto]	Automatically re-orders the given fileset
[-disable_unused]	Disables all files not associated with the TOP design unit
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>files</i>	Files to move

Categories

[Project](#)

Description

Reorders source files in the specified fileset. Takes the files indicated and places them at the front of, the back of, or before or after other files within the fileset. This command also has an auto reorder feature that reorders the files based on the requirements of the current top module in the design.

Arguments

-fileset *arg* - (Optional) The fileset in which to reorder files. The default is the sources_1 source fileset.

- before** *arg* - (Optional) Place the specified files before this file in the fileset. The file must be specified with the full path name in the fileset.
- after** *arg* - (Optional) Place the specified files after this file in the fileset. The file must be specified with the full path name in the fileset.
- front** - (Optional) Place the specified files at the front of the list of files in the fileset.
- back** - (Optional) Place the specified files at the back of the list of files in the fileset.
- auto** - (Optional) Enable automatic reordering based on the hierarchy requirements of the current top-module in the project. Often used after changing the top module with the **"set_property top"** command.
- disable_unused** - (Optional) Disable any files not currently used by the hierarchy based on the top-module. Often used after changing the top module with the **"set_property top"** command.
- quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.
- verbose** - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

files - (Required) One or more files to relocate in the fileset. Files must be specified by their full path name in the fileset, and are reordered in the order they are specified.

Examples

The following example takes the specified files and moves them to the front of the source filesset:

```
reorder_files -front {C:/Data/FPGA/file1.vhdl C:/Data/FPGA/file2.vhdl}
```

Note The default source filesset is used in the preceding example since the **-filesset** argument is not specified.

The following example sets a new top_module in the design, and then automatically reorders and disables unused files based on the hierarchy of the new top-module:

```
set_property top block1 [current_filesset]  
reorder_files -auto -disable_unused
```

See Also

- [add_files](#)
- [create_filesset](#)
- [current_filesset](#)
- [remove_files](#)

report_bps

Print details of the given breakpoint objects.

Syntax

```
report_bps [-quiet] [-verbose] [BreakPointObjs ...]
```

Returns

Print the breakpoints id, file_name and line_number to the console in textual format

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[BreakPointObjs]</i>	List of breakpoint objects to report

Categories

[Simulation](#)

report_carry_chains

Report carry chains.

Syntax

```
report_carry_chains [-file arg] [-append] [-return_string]  
[-max_chains arg] [-quiet] [-verbose]
```

Returns

Report

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append to existing file
[-return_string]	return report as string
[-max_chains]	Number of chains for which report is to be generated Default: 1
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

Description

Report the details of the carry chains used by the current open design. The report includes the average depth of all carry chains, as well as the specific depth of each carry chain reported.

By default, the longest carry chain is reported, but the number of chains reported can be specified.

The command returns the carry chain report.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-max_chains *arg* - (Optional) Number of chains to report. By default the longest carry chain is reported.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the 10 longest carry chains in the design:

```
report_carry_chains -max_chains 10
```


report_clock_interaction

Report on inter clock timing paths and unlocked registers.

Syntax

```
report_clock_interaction [-delay_type arg] [-setup] [-hold]
[-significant_digits arg] [-file arg] [-append] [-name arg]
[-return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-delay_type]</code>	Type of path delay: Values: max, min, min_max Default: max
<code>[-setup]</code>	Consider max delay timing paths (equivalent to -delay_type max)
<code>[-hold]</code>	Consider min delay timing paths (equivalent to -delay_type min)
<code>[-significant_digits]</code>	Number of digits to display: Range: 0 to 3 Default: 2
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-name]</code>	Output the results to GUI panel with this name
<code>[-return_string]</code>	Return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Report](#)

Description

Reports clock interactions and signals that cross clock domains to identify potential problems such a metastability or data loss or incoherency some visibility into the paths that cross clock domains is beneficial. This command requires an open synthesized or implemented design.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-delay_type *arg* - (Optional) Specifies the type of delay to analyze when running the clock interaction report. The valid values are min, max, and min_max. The default setting for **-delay_type** is max.

-setup - (Optional) Check for setup violations. This is the same as specifying **-delay_type max**.

-hold - (Optional) Check for hold violations. This is the same as specifying **-delay_type min**.

Note **-setup** and **-hold** can be specified together, which is the same as specifying **-delay_type min_max**

-significant_digits *arg* - (Optional) The number of significant digits in the output results. The valid range is 0 to 3. The default setting is 2 significant digits.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-name *arg* - (Optional) The name of the Clock Interaction Report view to display in the tool GUI mode. If the name has already been used in an open Report view, that view will be closed and a new report opened.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example sets the model for interconnect delay, selects a device speed grade, and then runs **report_clock_interaction**:

```
set_delay_model -interconnect none
set_speed_grade -3
report_clock_interaction -delay_type min_max -significant_digits 3 -name "results_1"
```

The following example returns the clock interactions, writing the report to the GUI, to the specified file, and returns a string which is assigned to the specified variable:

```
set clk_int [report_clock_interaction -file clk_int.txt -name clk_int1 \
-return_string]
```

See Also

- [create_clock](#)
- [create_generated_clock](#)
- [report_clocks](#)
- [set_delay_model](#)
- [set_speed_grade](#)

report_clock_networks

Report clock networks.

Syntax

```
report_clock_networks [-file arg] [-append] [-name arg]  
[-return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append the results to file, don't overwrite the results file
[-name]	Output the results to GUI panel with this name
[-return_string]	return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Reports the network fanout of each clock net in the design. The graphical form of the report, returned when the **-name** argument is specified, provides a hierarchical tree view of the clock network.

The default report that is returned to the standard output, to a file, or to a string, simply specifies the clock net names and the instance pins that is the clock start point.

The report is returned to the standard output by default, unless the **-file**, **-return_string**, or **-name** arguments are specified.

Arguments

-file *arg* - (Optional) Write the clock network report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-name *arg* - (Optional) Specifies the name of the results to output to the GUI.

-return_string - (Optional) Directs the output to a Tcl string. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the clock network names and startpoints to the specified file:

```
report_clock_networks -file C:/Data/ClkNets.txt
```

report_clock_utilization

Report information about clock nets in design.

Syntax

```
report_clock_utilization [-file arg] [-append] [-write_xdc arg]  
[-return_string] [-quiet] [-verbose]
```

Returns

Report

Usage

Name	Description
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append to existing file
<code>[-write_xdc]</code>	file to output clock constraint. If not specified the clock constraint will be appended to clock report.
<code>[-return_string]</code>	return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

Report

Description

Returns information related to clock nets in the design and clock resource utilization on the target device.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-write_xdc *arg* - (Optional) Output XDC location constraints for the various clock resources to the specified filename. If the path is not specified as part of the file name the file will be created in the current working directory, or the directory from which the tool was launched.

Note The XDC constraints in the clock utilization report begin with the "Location of..." statement. If the **write_xdc** option is specified, these lines will be output to the specified file rather than to the standard output

-return_string - (Optional) Direct the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns information about the clock nets in the design and the clock resources utilized on the target device, and writes it to the specified file:

```
report_clock_utilization -file C:/Data/FPGA_Design/clock_util.txt
```

The following example reports the clock nets and clock resource utilization to the standard output, but writes the XDC location constraints to the specified file:

```
report_clock_utilization -write_xdc clock_util_xdc.txt
```

Note Because the path is not specified as part of the XDC file name, the file will be created in the current working directory, or the directory from which the tool was launched.

See Also

- [create_clock](#)
- [create_generated_clock](#)

report_clocks

Report clocks.

Syntax

```
report_clocks [-file arg] [-append] [-return_string] [-quiet]  
[-verbose] [clocks]
```

Returns

Nothing

Usage

Name	Description
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-return_string]</code>	return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[<i>clocks</i>]</code>	List of clocks Default: *

Categories

[Report](#)

Description

Returns a table showing all the clocks in a design, including propagated clocks, generated and auto-generated clocks, virtual clocks, and inverted clocks in the current synthesized or implemented design. More detailed information about each clock net can be obtained with the **report_clock_utilization** command.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

clocks - (Optional) The clocks to match against the specified patterns. The default pattern is the wildcard '*' which returns all clocks in the design. More than one pattern can be specified to find multiple clocks based on different search criteria.

Examples

The following example returns the name, period, waveform, and sources of the clocks in the current design:

```
report_clocks -file C:/Data/FPGA_Design/clock_out.txt
```

The following example reports the clocks in the design with "Clock" in the name:

```
report_clocks *Clock*
```

See Also

- [create_clock](#)
- [create_generated_clock](#)
- [report_clock_utilization](#)

report_compile_order

Report the compile order by analyzing files and constructing a hierarchy.

Syntax

```
report_compile_order [-fileset arg] [-missing_instances] [-constraints]  
[-file arg] [-append] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-fileset]	Fileset to parse to determine compile order
[-missing_instances]	Report missing instances in the design hierarchy
[-constraints]	Report the constraint compile order
[-file]	Filename to output results to.
[-append]	Append output to existing file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Project](#)

Description

Report the compilation order of files in the various active filesets: constraints, design sources, and simulation sources.

This command returns the order of file processing for synthesis, implementation, and simulation. The report can be limited by specifying the fileset of interest with **-fileset**, or using the **-constraints** option to report the order of constraints processing in the active constraint set.

By default the report is returned to the Tcl console, or standard output, but it can also be written to a file.

Arguments

-fileset *arg* - (Optional) Limit the report to the specified fileset.

-missing_instances - (Optional) Return the list of cells that are missing source files in the current or specified fileset.

-constraints - (Optional) Report the compilation order of the constraint files in the current design, including constraints for any IP in the design.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the compilation order of the active filesets in the current design:

```
report_compile_order
```

The following returns a list of cells with missing source files in the current design, and appends the report to the specified file:

```
report_compile_order - missing_instances -file C:/Data/report1.txt -append
```

The following command lists the compile order of the files in the active constraint set:

```
report_compile_order -constraints
```

See Also

- [current_fileset](#)
- [update_compile_order](#)

report_conditions

Print details of the given condition objects.

Syntax

```
report_conditions [-quiet] [-verbose] [ConditionObjs ...]
```

Returns

Prints name, id, condition_expression and commands of each condition object on the console

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>ConditionObjs</i>]	ConditionObjs, id's or names

Categories

[Simulation](#)

report_config_timing

Report settings affecting timing analysis.

Syntax

```
report_config_timing [-file arg] [-append] [-name arg] [-return_string]
[-all] [-no_header] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-file]</code>	Output the results to file
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-name]</code>	Output the results to GUI panel with this name
<code>[-return_string]</code>	return report as string
<code>[-all]</code>	report all configuration settings (by default, only the typically important settings are reported)
<code>[-no_header]</code>	do not generate a report header
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Report](#), [Timing](#)

Description

Report the configuration of timing constraints of the current design.

By default the report is abbreviated, containing only a few key timing constraints. Use the **-all** argument to return all timing related configuration.

Arguments

-file *arg* - (Optional) Write the timing constraints configuration report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-name *arg* - (Optional) The name of the results to output to the GUI.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-all - (Optional) Reports the state of all timing related attributes and constraints in the design. By default, only a limited set of important timing attributes is reported.

-no_header - (Optional) Disables the report header. By default the report includes a header that lists:

- Report Type - timer_configuration.
- Design - The top module of the design.
- Part - The device, package, and speed grade of the target part.
- Version - The version of software used to create the report
- Date - The date of the report.
- Command - The command options used to create the report.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the current timing configuration, returns the information as a string, and sets that string into the specified Tcl variable:

```
set timeConfig [report_config_timing -all -no_header -return_string]
puts $timeConfig
```

See Also

[delete_timing_results](#)

report_control_sets

Report the unique control sets in design.

Syntax

```
report_control_sets [-file arg] [-append] [-sort_by args]
[-cells args] [-return_string] [-quiet] [-verbose]
```

Returns

Report

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append to existing file
[-sort_by]	Sort criterion: can be used only when -verbose is used. Options are clk, clkEn, set. Ex: report_control_sets -verbose -sort_by {clk clkEn}
[-cells]	Cells/bel_instances for which to report control sets
[-return_string]	return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Report the control sets of the current design.

Control sets are the list of control signals (Clock, CE, SR) for SLICE registers and LUTs. Registers must belong to the same control set in order to be packed into the same device resource. Registers without a control signal cannot be packed into devices with registers having control signals. A high number of control sets can cause difficulty fitting the device and can cause routing congestion and timing issues.

By default the **report_control_sets** command returns an abbreviated report indicating only the number of unique control sets. However, the **-verbose** arguments returns a detailed report of all control sets, for either the whole design or for the specified cells.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-sort_by *args* - (Optional) Sort the detailed report generated by the **-verbose** argument according to the specified criteria. Valid sort criteria are: **clk**, **clkEn**, and **set**.

Note The **-sort_by** argument is used with **-verbose**

-cells *args* - (Optional) Report control sets for the specified cell objects.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the control sets of the current design, sorted by the clk and clkEn signals:

```
report_control_sets -verbose -sort_by {clk clkEn}
```

The following example reports the control sets of the specified cells, sorted by clk and set:

```
report_control_sets -verbose -sort_by {clk set} -cells [get_cells usb*]
```

report_datasheet

Report data sheet.

Syntax

```
report_datasheet [-significant_digits arg] [-file arg] [-append]  
[-return_string] [-sort_by arg] [-name arg] [-show_all_corners]  
[-group args] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-significant_digits]	Number of digits to display: Range: 0 to 3 Default: 3
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append the results to file, don't overwrite the results file
[-return_string]	return report as string
[-sort_by]	Sorting order: Values: clock, port Default: clock
[-name]	Output the results to GUI panel with this name
[-show_all_corners]	provide all corners
[-group]	List of output ports for skew calculation
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Create a "datasheet" report for the current design. The datasheet has the timing characteristics of a design at the I/O pads, similar to what is reported in the .twr file.

For example setup and hold times of input I/Os in relation to clocks, max/min delays from clocks to output pads, skews of input/ output buses.

Arguments

-significant_digits *arg* - (Optional) Number of digits to display from 0 to 3. The default is 3.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-sort_by [port | clock] - (Optional) Sorting order. Valid values are clock or port. The default is to sort the report by clocks.

-show_all_corners - (Optional) Report all process corners.

-group [get_ports {xxx1 xxx2 ... xxxN}] - (Optional) Allows you to define your own custom group of ports for analysis. This option requires a list of port objects as returned by the **get_ports** command. The first port in the list will be used as the reference for skew calculation. In most cases, this will be a clock port of a source synchronous output interface. Multiple groups can be specified, each with its own reference clock port. When **-group** is not specified the timer automatically finds the group of output ports based on the launching clock, and reports skew based on that clock.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns the datasheet sorted by ports, for all process corners:

```
report_datasheet -sort_by port -show_all_corners
```

The following example reports the datasheet with the skew calculation for two groups of ports, with the first port of each group providing the reference for the skew calculation for that group. In this example, CLK0OUT is the forwarded clock for DATA0-4 and CLK1OUT is forwarded clock for DATA4-7:

```
report_datasheet -file ds.txt -group [get_ports {CLK0OUT DATA0 DATA1 DATA2 DATA3}] \
-group [get_ports {CLK1OUT DATA4 DATA5 DATA6 DATA7}]
```

See Also

[get_ports](#)

report_debug_core

Report details on debug cores.

Syntax

```
report_debug_core [-file arg] [-append] [-return_string] [-quiet]  
[-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-return_string]</code>	Return report as a string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Report](#)

Description

Writes a report of the various ILA debug cores in the current project, and the parameters of those cores. Debug cores can be added to a project using the **create_debug_core** command.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-return_string - (Optional) Return report as a string. This argument can not be used with the **-file** argument.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example writes the debug core report to the specified file name at the specified location:

```
report_debug_core -file C:/Data/FPGA_Design/project_1_cores.txt
```

See Also

- [create_debug_core](#)
- [write_chipscope_cdc](#)

report_default_switching_activity

Get default switching activity on specified default types.

Syntax

```
report_default_switching_activity [-static_probability] [-toggle_rate]
-type args [-file arg] [-return_string] [-append] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-static_probability]	report static probability
[-toggle_rate]	report toggle rate
-type	Reports the default seed values of specified types for vector-less propagation engine. List of valid default type values: input, input_set, input_reset, input_enable, register, dsp, bram_read_enable, bram_write_enable, output_enable, clock, all
[-file]	Filename to output results to. (send output to console if -file is not used)
[-return_string]	return default switching activity as string
[-append]	append default switching activity to end of file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Displays the default switching activity currently configured for the specified element type.

The reported values are defined using the **set_default_switching_activity** command.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-static_probability - (Optional) Include static probability in the report but do not include toggle rate.

-toggle_rate - (Optional) Include toggle rate in the report but do not include static probability.

Note Both toggle rate and probability will be reported unless either **-toggle_rate** or **-static_probability** is specified to limit the results

-type <types> - (Required) The component types that are reported. Valid values are: input, input_set, input_reset, input_enable, register, dsp, bram_read_enable, bram_write_enable, output_enable, clock, all.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-return_string - (Optional) Return the report as a string rather than a data set.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the default switching attributes for dsp:

```
report_default_switching_activity -type dsp
Default Dsp Probability = 0.50
Default Dsp Toggle Rate (%) = 12.50
```

The following example reports the default switching attributes for all types, and stores it into a Tcl variable `swa1`:

```
set swa1 [report_default_switching_activity -type all -return_string]
```

Note Without the **-return_string** argument, the command will perform correctly, but the `$swa` variable will not be assigned the reported information.

See Also

- [power_opt_design](#)
- [report_power](#)
- [report_switching_activity](#)
- [reset_default_switching_activity](#)
- [reset_switching_activity](#)
- [set_default_switching_activity](#)
- [set_switching_activity](#)

report_disable_timing

Report disabled timing arcs.

Syntax

```
report_disable_timing [-file arg] [-append] [-return_string] [-quiet]  
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append the results to file, don't overwrite the results file
[-return_string]	return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#), [Timing](#)

Description

Displays a report of timing paths that will be excluded from timing analysis in the current synthesized or implemented design.

The format of the report is organized into columns for "Cell or Port" to define the object associated with the timing path, "From" and "To" to define the timing path, the condition, and the reason for excluding the path from timing. The various reasons for exclusion are as follows:

- constraint - **set_disable_timing** constraint is specified
- constant - Logic constant
- loop - Breaks a logic loop
- bidirect instance path - Feedback path through bidirectional instances
- bidirect net path - Feedback path on nets with bidirectional pins

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports all timing paths that will not be included in timing analysis:

```
report_disable_timing
```

The following example outputs the disable timing report as a string, stores it in a variable, and then puts it to the display:

```
set bad_time [report_disable_timing -return_string]
puts $bad_time
```

See Also

- [report_timing](#)
- [set_disable_timing](#)

report_drc

Run DRC.

Syntax

```
report_drc [-name arg] [-rules args] [-ruledECK arg] [-file arg]  
[-format arg] [-append] [-return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-name]	Output the results to GUI panel with this name
[-rules]	DRC rules (see get_drc_checks for available rules)
[-ruledECK]	Container of DRC rule checks Default: report_drc
[-file]	Filename to output results to. (send output to console if -file is not used)
[-format]	Specifies how to format the report. Default is 'text', another option is 'xml'. Only applies if -file is used. If xml output is used, -append is not allowed. Default: text
[-append]	Append the results to file, do not overwrite the results file
[-return_string]	return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[DRC](#), [Report](#)

Description

Check the current design against a specified set of design rule checks, or a rule deck, and report any errors or violations that are found.

The tool includes a large number of predefined design rule checks to be used by the **report_drc** command. Use the **get_drc_checks** command to list the currently defined design rule checks. You can also create new custom design rule checks using the **create_drc_check** command.

A rule deck is a collection of design rule checks grouped for convenience, to be run at different stages of the FPGA design flow, such as during I/O planning or placement. The tool comes with a set of factory defined rule decks, but you can also create new user-defined rule decks with the **create_drc_ruledeck** command. Use the **get_drc_ruledecks** command to return a list of the currently defined rule decks available for use in the **report_drc** command.

The **report_drc** command runs a default rule deck when the **-rules** or **-ruledeck** options are not specified. Creating a user-defined DRC automatically adds the new design rule check to the default rule deck.

DRC rules can be enabled or disabled using the **IS_ENABLED** property on the rule check object. If a rule **IS_ENABLED** false, the rule will not be run by the **report_drc** command, whether it is specified directly using **-rules**, or indirectly with **-ruledeck**.

Tip You can reset the properties of a DRC rule to the factory default settings using the **reset_drc_check** command.

This command requires an open design to check the design rules against. The command returns a report with the results of violations found by the design rule checks. Violations can be listed with the **get_drc_vios** command.

You can reset the current results of the **report_drc** command, clearing any found violations, using the **reset_drc** command.

Arguments

-name arg - (Optional) The name to assign to the results when run in GUI mode.

-ruledeck arg - (Optional) The name of a DRC rule deck. A rule deck is a list of DRC rule check names. You can provide the name of a factory DRC rule deck or a user-defined rule deck. The **report_drc** command checks the design against the rules that are added to the given rule deck. Custom rule decks can be defined using the **create_drc_ruledeck** command. Use the **get_drc_ruledecks** command to list the currently defined rule decks.

-rules args - (Optional) A list of rules to run the DRC report against. All specified rules will be checked against the current design. Rules are listed by their group name or full key. Using the **-rules** option creates a temporary user-defined rule deck, with the specified design rule checks, and uses the temporary rule deck for the run.

Note **-ruledeck** and **-rules** cannot be used together

-file arg - (Optional) Write the DRC report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example lists the available rule decks. The results include all factory rule decks and all user-defined rule decks.

```
get_drc_ruledecks
```

The following example returns the list of DRC rules defined in the specified rule deck:

```
get_drc_checks -of_objects [get_drc_ruledecks placer_checks]
```

The following examples run the specified DRC rule deck and rules against the current design, and writes the results to the specified file:

```
report_drc -ruledck placer_checks -file C:/Data/DRC_Rpt1.txt  
report_drc -rules {IOCNT-1 IOPCPR-1 IOPCMGT-1 IOCTMGT-1 IODIR-1} \  
-file C:/Data/DRC_Rpt1.txt -append
```

Note The **-append** option adds the result of the second **report_drc** command to the specified file

See Also

- [create_drc_check](#)
- [create_drc_ruledck](#)
- [create_drc_violation](#)
- [get_drc_checks](#)
- [get_drc_ruledcks](#)
- [get_drc_vios](#)
- [reset_drc](#)
- [reset_drc_check](#)

report_drivers

Print drivers along with current driving values for an HDL wire or signal object.

Syntax

```
report_drivers [-quiet] [-verbose] hdl_object
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hdl_object</i>	Which hdl_object to report

Categories

[Simulation](#)

report_environment

Report system information.

Syntax

```
report_environment [-file arg] [-format arg] [-append] [-return_string]
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-file]	Write system information to specified file.
[-format]	Specifies how to format the report. Default is 'text', another option is 'xml'. Only applies if -file is used. If xml output is used, -append is not allowed. Default: text
[-append]	Append report to existing file
[-return_string]	Return report content as a string value
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

Description

Report the details of the system environment that the tool is running under. The details of the environment report include: operating system version, CPU, memory, available disk space, and specific settings of various environment variables.

The default is to write the report to the standard output. However, the report can be written to a file instead.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the current environment to the specified file:

```
report_environment -file C:/Data/toolEnv.txt
```

report_exceptions

Report timing exceptions.

Syntax

```
report_exceptions [-from args] [-rise_from args] [-fall_from args]
[-to args] [-rise_to args] [-fall_to args] [-through args]
[-rise_through args] [-fall_through args] [-ignored] [-no_header]
[-file arg] [-append] [-return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-from]</code>	From pins, ports, cells or clocks
<code>[-rise_from]</code>	Rising from pins, ports, cells or clocks
<code>[-fall_from]</code>	Falling from pins, ports, cells or clocks
<code>[-to]</code>	To pins, ports, cells or clocks
<code>[-rise_to]</code>	Rising to pins, ports, cells or clocks
<code>[-fall_to]</code>	Falling to pins, ports, cells or clocks
<code>[-through]</code>	Through pins, ports, cells or nets
<code>[-rise_through]</code>	Rising through pins, ports, cells or nets
<code>[-fall_through]</code>	Falling through pins, ports, cells or nets
<code>[-ignored]</code>	only report exceptions which are ignored
<code>[-no_header]</code>	Do not generate a report header
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-return_string]</code>	return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

Report, Timing

Description

Report all timing exceptions applied to setup and hold checks defined by timing constraints in the current design, or report the exceptions on the specified timing paths. Timing exceptions can be defined by timing constraints such as **set_false_path** or **set_multicycle_path** that change the default assumptions for timing paths in the design.

The exceptions are reported to the standard output by default, but can be redirected to a file or to a Tcl string variable.

Arguments

-from *args* - (Optional) A list of start points on the timing path to report exceptions on.

-rise_from *args* - (Optional) A list of the start points on the timing path to report exceptions on the rising-edge of the path.

-fall_from *args* - (Optional) A list of the start points on the timing path to report exceptions on the falling-edge of the path.

-to *args* - (Optional) A list of the end points for the timing path to report exceptions on.

-rise_to *args* - (Optional) A list of the end points on the timing path to report exceptions on the rising-edge of the path.

-fall_to *args* - (Optional) A list of the end points on the timing path to report exceptions on the falling-edge of the path.

-through *args* - (Optional) A list of pins, cell, or nets through which the timing path passes.

-rise_through *args* - (Optional) A list of pins, cell, or nets through which the rising-edge timing path passes.

-fall_through *args* - (Optional) Specifies the list of pins, cell, or nets through which the falling-edge timing path passes.

-ignored - (Optional) Report only the timing exceptions which are totally ignored. For example, if a **set_max_delay** is applied from a pin A to a pin B, and if there is no timing path between these two pins, the **set_max_delay** will be totally ignored by the timing engine, and reported with this option.

-file *arg* - (Optional) Write the report into the specified file. By default the timing exceptions are reported to the standard output, or the Tcl console. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

This example reports all timing exceptions in the current design:

```
report_exceptions
```

See Also

- [create_clock](#)
- [create_generated_clock](#)
- [report_timing](#)
- [report_timing_summary](#)
- [set_false_path](#)
- [set_input_delay](#)
- [set_max_delay](#)
- [set_min_delay](#)
- [set_multicycle_path](#)
- [set_output_delay](#)

report_high_fanout_nets

Report high fanout nets.

Syntax

```
report_high_fanout_nets [-file arg] [-append] [-ascending]
[-load_types] [-clock_regions] [-slr] [-max_nets arg] [-min_fanout arg]
[-max_fanout arg] [-cells args] [-return_string] [-quiet] [-verbose]
```

Returns

Report

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append to existing file
[-ascending]	Report nets in ascending order
[-load_types]	Report load details
[-clock_regions]	Report clock region wise load distribution
[-slr]	Report SLR wise load distribution
[-max_nets]	Number of nets for which report is to be generated Default: 10
[-min_fanout]	Report nets that have fanout greater than or equal to the specified integer Default: 1
[-max_fanout]	Report nets that have fanout less than or equal to the specified integer Default: INT_MAX
[-cells]	Cells/bel_instances for which to report nets
[-return_string]	return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Report the fanout of nets in the design, starting with the highest fanout nets, and working down. Options allow you to control various aspects of the report.

This command can be run on an implemented design, or on the synthesized netlist. However, the results will be more complete on the implemented design.

The command returns the fanout report of nets in the design.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-ascending - (Optional) Report nets in ascending order.

-load_types - (Optional) Reports the various load types on the net sorted in two different ways: by load types (data/clock/set/reset...) and by device resource at which loads are placed (Slices/IO...). When report_high_fanout_nets is run on the unplaced synthesized design only the load type is reported.

-clock_regions - (Optional) Report the load distribution across clock regions. This option reports the clock regions the various loads on the net are located in, after placement.

-slr - (Optional) Report the load distribution across SLRs. This option reports the SLRs the various loads on the net are located in, after placement.

-max_nets *arg* - (Optional) Number of nets to report. Default: 10

-min_fanout *arg* - (Optional) Report nets that have fanout greater than or equal to the specified integer. This allows you to report nets with specific fanout loads of interest. Default: 1.

-max_fanout *arg* - (Optional) Report nets that have fanout less than or equal to the specified integer. This allows you to report nets with specific fanout loads of interest. There is no maximum value specified by default.

-cells *args* - (Optional) Cells/bel_instances for which to report nets. Report the nets attached to the specified cells or bels in the design.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the top 100 nets with fanouts greater than 50 loads:

```
report_high_fanout_nets -min_fanout 50 -max_nets 100
```

report_incremental_reuse

Compute achievable incremental reuse for the given design-checkpoint and report.

Syntax

```
report_incremental_reuse [-file arg] [-append] [-cells args]  
[-hierarchical] [-hierarchical_depth arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append to existing file
[-cells]	Report incremental reuse of given list of cells
[-hierarchical]	Generates text-based hierarchical incremental reuse report.
[-hierarchical_depth]	Specifies the depth level for textual hierarchical incremental reuse report Default: 0
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

For use with the incremental compilation flow, this command reports on the amount of design overlap between the current design and an incremental checkpoint loaded using the **read_checkpoint -incremental** command.

This report analyzes the loaded incremental checkpoint against the current design to see if the two are sufficiently correlated to drive incremental placement and routing. A low correlation between the current design and the checkpoint should discourage using the checkpoint as a basis for incremental place and route. Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on incremental place and route.

If there is a low correlation of reuse between the current design and the loaded incremental checkpoint, you will need to restore the original design using **open_run** or **read_checkpoint**. Alternatively, you can overload the incremental checkpoint in the current design by issuing the **read_checkpoint -incremental** command again to specify a new incremental checkpoint.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified. By default, the report will be written to the Tcl console.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-cells *args* - (Optional) Specifies the cells to use from the DCP file.

-hierarchical - (Optional) Generate a text-based hierarchical incremental reuse report.

-hierarchical_depth *arg* - (Optional) Specifies the depth level for the text-based hierarchical incremental reuse report. The default is 0.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example loads an incremental checkpoint into the current design, and then reports the correlation of the loaded incremental checkpoint to the current design:

```
read_checkpoint -incremental C:/Data/reuse_checkpoint1.dcp
report_incremental_reuse
```

See Also

- [open_run](#)
- [place_design](#)
- [read_checkpoint](#)
- [route_design](#)

report_io

Display information about all the IO sites on the device.

Syntax

```
report_io [-file arg] [-append] [-return_string] [-quiet] [-verbose]
```

Returns

Report

Usage

Name	Description
[-file]	Filename to output results to. Send output to console if -file is not used.
[-append]	Append to existing file
[-return_string]	return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Report details of the IO banks of the current design. Details include device specific information such as target part, package, and speed grade, and also provides information related to each pin on the device.

Arguments

-file *arg* - (Optional) Write the report into the specified file.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the IO blocks of the current design:

```
report_io
```

See Also

[report_route_status](#)

report_ip_status

Report on the status of the IP instances in the project.

Syntax

```
report_ip_status [-name arg] [-file arg] [-append] [-return_string]  
[-quiet] [-verbose]
```

Returns

True for success

Usage

Name	Description
[-name]	Output the results to GUI panel with this name Values: The name of the GUI dialog
[-file]	Filename to output results to (send output to console if -file is not used) Values: The report filename
[-append]	Append to existing file
[-return_string]	Return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[IPFlow](#)

report_objects

Print details of the given hdl objects (variable, signal, wire, or reg).

Syntax

```
report_objects [-quiet] [-verbose] [hdl_objects...]
```

Returns

Print name, type, data_type of the HDL objects on console in textual format

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hdl_objects]	The hdl_objects to report. Default is report_objects [get_objects *]

Categories

[Simulation](#)

report_operating_conditions

Get operating conditions values for power estimation.

Syntax

```
report_operating_conditions [-voltage args] [-grade] [-process]
[-junction_temp] [-ambient_temp] [-thetaja] [-thetasa] [-airflow]
[-heatsink] [-thetajb] [-board] [-board_temp] [-board_layers] [-all]
[-file arg] [-return_string] [-append] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-voltage]	Gets voltage value. Supported voltage supplies vary by family.
[-grade]	Temperature grade. Supported values vary by family.
[-process]	Gets process
[-junction_temp]	Junction Temperature (C): auto degC
[-ambient_temp]	Ambient Temperature (C): default degC
[-thetaja]	ThetaJA (C/W): auto degC/W
[-thetasa]	Gets ThetaSA
[-airflow]	Airflow (LFM): 0 to 750
[-heatsink]	Gets dimensions of heatsink
[-thetajb]	Gets ThetaJB
[-board]	Board type: jedec, small, medium, large, custom
[-board_temp]	Board Temperature degC
[-board_layers]	Board layers: 4to7, 8to11, 12to15, 16+
[-all]	Gets all operating conditions listed in this help message
[-file]	Filename to output results to. (send output to console if -file is not used)
[-return_string]	return operating conditions as string
[-append]	append operating conditions to end of file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report, SDC

Description

Displays the real-world operating conditions that are used when performing analysis of the design.

The environmental operating conditions of the device are used for power analysis when running the **report_power** command, but are not used during timing analysis. The values of operating conditions can be defined by the **set_operating_conditions** command.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

- voltage** - (Optional) Report the list of voltage pairs. Supported voltage supplies vary by family.
- grade** - (Optional) Report the temperature grade of the target device
- process** - (Optional) Report the manufacturing process characteristics to be assumed.
- junction_temp** - (Optional) Report the device junction temperature used for modeling
- ambient_temp** - (Optional) Reports the environment ambient temperature
- thetaja** - (Optional) Report the Theta-JA thermal resistance used for modeling
- thetasa** - (Optional) Report the Theta-SA thermal resistance used for modeling
- airflow** - (Optional) Report the Linear Feet Per Minute (LFM) airflow to be used for modeling.
- heatsink** - (Optional) Report the heatsink type to be used for modeling.
- thetajb** - (Optional) Report the Theta-JB thermal resistance used for modeling
- board** - (Optional) Report the board size to be used for modeling.
- board_temp** - (Optional) Report the board temperature in degrees Centigrade to be used for modeling.
- board_layers** - (Optional) Report the number of board layers to be used for modeling
- all** - (Optional) Report the current values of all operating conditions. Use this to avoid having to report each condition separately.
- file *arg*** - (Optional) Write the report into the specified file.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

Specify an industrial temperature grade device with an ambient temperature of 75 degrees C and then write those settings to a file on disk.

```
set_operating_conditions -grade industrial -junction_temp 75
report_operating_conditions -grade -junction_temp -return_string -file \
~/conditions.txt
```

See Also

[set_operating_conditions](#)

report_param

Get information about all parameters.

Syntax

```
report_param [-file arg] [-append] [-non_default] [-return_string]  
[-quiet] [-verbose] [pattern]
```

Returns

Param report

Usage

Name	Description
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-non_default]</code>	Report only params that are set to a non default value
<code>[-return_string]</code>	Return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[<i>pattern</i>]</code>	Display params matching pattern Default: *

Categories

[PropertyAndParameter](#), [Report](#)

Description

Gets a list of all user-definable parameters, the current value, and a description of what the parameter configures or controls.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pattern - (Optional) Match parameters against the specified pattern. The default pattern is the wildcard '*' which gets all user-definable parameters.

Examples

The following example returns the name, value, and description of all user-definable parameters:

```
report_param
```

The following example returns the name, value, and description of user-definable parameters that match the specified search pattern:

```
report_param *coll*
```

See Also

- [get_param](#)
- [list_param](#)
- [reset_param](#)
- [set_param](#)

report_phys_opt

Report details of Physical Synthesis transformations.

Syntax

```
report_phys_opt [-file arg] [-append] [-return_string] [-quiet]  
[-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-file]</code>	Output file
<code>[-append]</code>	Append the results to file
<code>[-return_string]</code>	return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Report](#)

Description

Reports the results of the fanout driver replication and load redistribution optimizations performed by the **phys_opt_design** command.

Arguments

-file *arg* - (Optional) Write the physical optimization report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-return_string - (Optional) Directs the output of the report to a Tcl string. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the physical optimizations performed in the current design by the **phys_opt_design** command:

```
report_phys_opt -file C:/Data/physOpt_Report.txt
```

See Also

[phys_opt_design](#)

report_power

Run power estimation and display report.

Syntax

```
report_power [-no_propagation] [-hier arg] [-vid] [-file arg]
[-name arg] [-format arg] [-xpe arg] [-l arg] [-return_string]
[-append] [-fileset arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-no_propagation]	Disables the propagation engine to estimate the switching activity of nets.
[-hier]	Hierarchy report style (logic, power, or all) Default: logic
[-vid]	Voltage ID (VID) of device is used
[-file]	Filename to output results to. (send output to console if -file is not used)
[-name]	Output the results to GUI panel with this name
[-format]	Format for the power estimation report: text, xml Default: text
[-xpe]	Output the results to XML file for importing into XPE
[-l]	Maximum number of lines to report in detailed reports (l >= 0) Default: 10
[-return_string]	return report as string
[-append]	append power report to end of file
[-fileset]	Fileset to parse and get .xpe file for PS7 block
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report, Power

Description

Run power analysis on the current design, and report details of power consumption based on the current operating conditions of the device, and the switching rates of the design. The operating conditions can be set using the **set_operating_conditions** command. The switching activity can be defined using the **set_default_switching_activity** command.

Power analysis requires a synthesized netlist, or a placed and routed design.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-no_propagation - (Optional) For all undefined nodes power analysis uses a vector-less propagation engine to estimate activity. This argument disables the propagation engine for a faster analysis of the design.

-hier [logic | power | all] - (Optional) Defines the details (logic, or power) to include in the Detailed Report section of the output. The default is **logic**.

-vid - (Optional) Use the Voltage ID bit of the target device. Voltage identification is a form of adaptive voltage scaling (AVS) that enables certain devices in the Virtex-7 family to be operated at a reduced voltage of 0.9V while delivering the same specified performance of a device operating at the nominal supply voltage of 1.0V. Voltage identification capable devices consume approximately 30% lower worst case (maximum) static power and correspondingly dissipate less heat.

-file arg - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-name arg - (Optional) Specifies the name of the results set to report the results to.

-xpe arg - (Optional) Output the results to an XML file for importing into XPower Estimator or XPower Analyzer.

-l arg - (Optional) Maximum number of lines to report in the Detailed Reports section. Valid values are greater than or equal to 0.

Note This options also triggers additional levels of detail in the Detailed Report section that are not reported when **-l** is not specified.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example performs power analysis, without net propagation, and writes the results to an XML file for use in XPE:

```
report_power -no_propagation -xpe C:/Data/design1.xpe
```

See Also

- [read_saif](#)
- [read_vcd](#)
- [set_default_switching_activity](#)
- [set_operating_conditions](#)

report_power_opt

Report power optimizations.

Syntax

```
report_power_opt [-cell arg] [-file arg] [-format arg] [-name arg]  
[-append] [-return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-cell]	top level cell
[-file]	output file
[-format]	Specifies how to format the report. Default is 'text', another option is 'xml'. Only applies if -file is used. If xml output is used, -append is not allowed. Default: text
[-name]	Output the results to GUI panel with this name
[-append]	append if existing file. Otherwise overwrite existing file.
[-return_string]	return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Power

Description

Report power optimizations that have been performed on the design with the **power_opt_design** command.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-cell *arg* - (Optional) Report power optimization for a specific cell instance. By default, the power optimizations performed on the whole design are reported.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the power optimizations performed on the current design:

```
report_power_opt
```

See Also

- [power_opt_design](#)
- [report_power](#)

report_property

Report properties of object.

Syntax

```
report_property [-all] [-class arg] [-return_string] [-file arg]
[-append] [-regexp] [-quiet] [-verbose] [object] [pattern]
```

Returns

Property report

Usage

Name	Description
<code>[-all]</code>	Report all properties of object even if not set
<code>[-class]</code>	Object type to query for properties. Not valid with
<code>[-return_string]</code>	Set the result of running report_property in the Tcl interpreter's result variable
<code>[-file]</code>	Filename to output result to. Send output to console if -file is not used
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-regexp]</code>	Pattern is treated as a regular expression
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[<i>object</i>]</code>	Object to query for properties
<code>[<i>pattern</i>]</code>	Pattern to match properties against Default: *

Categories

[Object](#), [PropertyAndParameter](#), [Report](#)

Description

Gets the property name, property type, and property value for all of the properties on a specified object.

Note `list_property` also returns a list of properties on an object, but does not include the property type or value.

Arguments

-all - (Optional) Return all properties of an object, even if the property value is not defined.

-return_string - (Optional) Directs the output to a Tcl string. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

object - (Optional) A single object on which to report properties.

Note If you specify multiple objects you will get an error.

Examples

The following example returns all properties of the specified object:

```
report_property -all [get_cells cpuEngine]
```

To determine which properties are available for the different design objects supported by the tool, you can use multiple **report_property** commands in sequence. The following example returns all properties of the specified current objects:

```
report_property -all [current_project]
report_property -all [current_fileset]
report_property -all [current_design]
report_property -all [current_run]
```

See Also

- [create_property](#)
- [get_cells](#)
- [get_property](#)
- [list_property](#)
- [list_property_value](#)
- [reset_property](#)
- [set_property](#)

report_pulse_width

Report pulse width check.

Syntax

```
report_pulse_width [-file arg] [-append] [-name arg] [-return_string]
[-all_violators] [-significant_digits arg] [-limit arg] [-min_period]
[-max_period] [-low_pulse] [-high_pulse] [-max_skew] [-clocks args]
[-quiet] [-verbose] [objects]
```

Returns

Nothing

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append the results to file, don't overwrite the results file
[-name]	Results name in which to store output
[-return_string]	return report as string
[-all_violators]	Only report pins/ports where check violations occur
[-significant_digits]	Number of digits to display: Range: 0 to 3 Default: 2
[-limit]	Number of checks of a particular type to report per clock: Default is 1 Default: 1
[-min_period]	Only report min period checks
[-max_period]	Only report max period checks
[-low_pulse]	Only report min low pulse width checks
[-high_pulse]	Only report min high pulse width checks
[-max_skew]	Only report max skew checks
[-clocks]	List of clocks for which to report min pulse width/min period checks
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>objects</i>]	List of objects to check min pulse width with

Categories

[Report](#)

Description

Reports the pulse width of the specified clock signals in the clock network and upon reaching the flip-flop. This command also performs high pulse width checking, using maximum delay for the rising edge and minimum delay for the falling edge of the clock. Performs low pulse width checking using minimum delay for the rising edge, and maximum delay for the falling edge. This results in a worst case analysis for the current Synthesis or Implemented Design because it assumes worst-case delays for both rising and falling edges. This command also reports the maximum skew, or maximum timing separation allowed between clock signals.

The report includes minimum pulse width, maximum pulse width, low pulse width, high pulse width, and max skew checks by default. However, selecting a specific check will disable the other checks unless they are also specified.

The default report is returned to the standard output, but can be redirected to a file, or to a Tcl string variable for further processing. The report is returned to the standard output by default, unless the **-file**, **-return_string**, or **-name** arguments are specified.

Arguments

-file *arg* - (Optional) Write the report into the specified file. If the specified file already exists, it will be overwritten by the new report, unless the **-append** option is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-name *arg* - (Optional) Specifies the name of the results set for the GUI. Pulse Width reports in the GUI can be deleted by the **delete_timing_results** command.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-all_violators - (Optional) Report only the *objects* where violations are found.

-significant_digits *arg* - (Optional) The number of significant digits in the output results. The valid range is 0 to 3. The default setting is 2 significant digits.

-limit *arg* - (Optional) The number of checks of a particular type to report per clock. This is a value ≥ 1 , and the default is 1.

-min_period - (Optional) Report minimum period checks.

-max_period - (Optional) Report maximum period checks.

-low_pulse - (Optional) Report minimum low pulse width checks.

-high_pulse - (Optional) Report minimum high pulse width checks.

-max_skew - (Optional) Check the skew constraints between two clock pins.

Note The default of the **report_pulse_width** command is to report **min_period**, **max_period**, **low_pulse**, **high_pulse**, and **max_skew**. Specifying one or more of these options configures the command to only report the specified checks.

-clocks arg - (Optional) Clocks to report pulse width and period checks. All clocks are checked if the **-clocks** option is not specified.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) The pin objects to report the pulse width from. All pins are checked if no *objects* are specified.

Examples

The following example performs the minimum period and low pulse width check, returning the results to a named results set in the GUI:

```
report_pulse_width -min_period -low_pulse -name timing_1
```

See Also

[delete_timing_results](#)

report_route_status

Report on status of the routing.

Syntax

```
report_route_status [-return_string] [-file arg] [-append]
[-of_objects args] [-route_type arg] [-list_all_nets] [-show_all]
[-dump_routes] [-has_routing] [-ignore_cache] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-return_string]	Set the result of running the report in the Tcl interpreter's result variable
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append the results to file, don't overwrite the results file
[-of_objects]	Report detailed routing for these routes
[-route_type]	Only show routes with the given status: UNPLACED NOLOADS NODRIVER UNROUTED ANTENNAS CONFLICTS PARTIAL INTRASITE HIERPORT ROUTED(ignored if -of_objects is used)
[-list_all_nets]	list full route information for every net in the design (ignored if -of_objects is used)
[-show_all]	list all relevant pins for routes marked as UNPLACED or PARTIAL routes and list all relevant nodes for routes marked as ANTENNAS or CONFLICTS routes (by default only the first 15 pins or nodes are listed for a route)
[-dump_routes]	show the full routing tree for every routed net in the design. This is VERY VERBOSE.
[-has_routing]	returns 0 if there is no routing currently stored for this design and 1 if there is. All other options are ignored.
[-ignore_cache]	throw away all cached information and recalculate the route status for the entire design (slow)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

Description

Reports the state of routing in the current design.

The route status report can include a wide range of information, from a simple 1 if the design has routing, to a complete route tree for each net in the design.

Arguments

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-of_objects *args* - (Optional) Report the full routing tree for the specified route, net, or xdef_net objects.

-route_type *arg* - (Optional) Only show routes with the specified route status. Valid route states are: UNPLACED, NOLOADS, NODRIVER, UNROUTED, ANTENNAS, CONFLICTS, PARTIAL, INTRASITE, HIERPORT, ROUTED.

Note This option is ignored if **-of_objects** is specified

-list_all_nets - (Optional) Report summary route status for every net in the design.

Note This option is ignored if **-of_objects** is specified

-show_all - (Optional) Report all relevant pins for routes marked as UNPLACED or PARTIAL routes and list all relevant nodes for routes marked as ANTENNAS or CONFLICTS routes. As a default only the first 15 pins or nodes are listed for a given route.

-dump_routes - (Optional) Report the full routing tree for every routed net in the design.

Note This is a very long report, and can take some time to generate.

-has_routing - (Optional) Returns false (0) if the design is unrouted, and returns true (1) if the design has routing. All other options are ignored when **-has_routing** is specified.

Note Has routing does not mean fully routed.

-ignore_cache - (Optional) By default the **report_route_status** command is iterative, and only updates the route information for new nets and routes as the design is implemented. This argument will cause the command to ignore the cached information and regenerate the report for the entire design.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the route status for the specified nets:

```
report_route_status -of_objects [get_nets u4*]
```

report_scopes

Print names of the children scopes (declarative regions) of given scope(s) or the current scope.

Syntax

```
report_scopes [-quiet] [-verbose] [hdl_scopes ...]
```

Returns

Report_scopes prints a subset of properties of the HDL scope on console in textual format

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hdl_scopes]	The hdl_objects to report. Default is report_scopes [get_scopes *]

Categories

[Simulation](#)

report_simlib_info

Report info of simulation libraries.

Syntax

```
report_simlib_info [-file arg] [-append] [-quiet] [-verbose] [path]
```

Returns

Nothing

Usage

Name	Description
[-file]	Output file Default: report_simlib_info.log
[-append]	Append mode
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>path</i>]	Report Pre-Compiled library information

Categories

[Simulation](#)

report_ssn

Run SSN analysis on the current package and pinout.

Syntax

```
report_ssn [-name arg] [-return_string] [-format arg] [-file arg]  
[-append] [-phase] [-quiet] [-verbose]
```

Returns

Ssn report

Usage

Name	Description
[-name]	Output the results to GUI panel with this name
[-return_string]	Return report as string
[-format]	Report format. Valid arguments are CSV, HTML, TXT Default: csv
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append the report to the specified file
[-phase]	Account for multi-clock phase in the analysis
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Perform a simultaneous switching noise (SSN) analysis of the current design. The SSN analysis is an accurate method for predicting how output switching affects interface noise margins. The calculation and estimates are based on a range of variables intended to identify potential noise-related issues in your design and should not be used as final design "sign off" criteria.

SSN analysis provides estimates of the disruption that simultaneously switching outputs can cause on other output ports in the I/O bank, as well as input ports in the case of Spartan-6 devices. The SSN predictor incorporates I/O bank-specific electrical characteristics into the prediction to better model package effects on SSN.

By default, **report_ssn** assumes that every port toggles asynchronously. This results in a worst-case noise analysis, which may be overly pessimistic. The **-phase** option lets you consider clocking information available in the design to more accurately report SSN noise. Clocks must be defined using the **create_clock** and **create_generated_clock** commands. The period, phase shift and duty cycle of the generated clocks have significant impact on SSN analysis.

The **report_ssn** command provides a detailed SSN analysis for Spartan-6, Virtex-6, Virtex-7, Kintex-7, and Artix-7 devices. For the Spartan-3, Virtex-4, and Virtex-5 devices, the **report_sso** command provides an approximation of switching noises.

The report is returned to the standard output, unless the **-file**, **-return_string**, or **-name** arguments are specified.

Arguments

-name arg - (Optional) Specifies the name of the results to output to the GUI.

-return_string - (Optional) Directs the output to a Tcl string. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-format [csv | html] - (Optional) Specifies the format of the output as either comma-separated values (CSV), or HTML. The default output is CSV.

Note The format applies when **-file** is specified, but is otherwise ignored.

-file arg - (Optional) Write the SSN report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-phase - (Optional) Consider clock switching cycles in SSN analysis to provide a more accurate result.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example performs an SSN analysis on the current design, formats the output as HTML, and writes the output to the specified file:

```
report_ssn -format html -file C:/Data/devSSN.html
```

The following example performs an SSN analysis, with phase analysis, and returns the output to a string which is stored in the specified variable:

```
set devSSN [report_ssn -phase -format html -return_string]
```

Note The **-format** argument in the preceding example is ignored in the absence of **-file**.

See Also

- [create_clock](#)
- [create_generated_clock](#)
- [reset_ssn](#)

report_switching_activity

Get switching activity on specified objects.

Syntax

```
report_switching_activity [-static_probability] [-signal_rate]  
[-file arg] [-return_string] [-append] [-quiet] [-verbose]  
[objects...]
```

Returns

Nothing

Usage

Name	Description
<code>[-static_probability]</code>	report static probability
<code>[-signal_rate]</code>	report signal rate
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-return_string]</code>	return switching activity as string
<code>[-append]</code>	append switching activity to end of file
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[objects]</code>	objects

Categories

[Report](#)

Description

This command is used to report different kinds of switching activity on design nets, ports, pins, and cells. These include simple signal rate and simple static probability on nets, ports, and pins; and state dependent static probabilities on cells.

The reported values are defined using the **set_switching_activity** command.

Note This command returns the switching activity for the specified objects.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-static_probability - (Optional) Specifies that the command returns static probability as part of the report.

-signal_rate - (Optional) Specifies that the command returns the signal rate as part of the report.

-file *filename* - (Optional) Write the report to the specified path and file.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-return_string - (Optional) Returns the data as a text string for assignment to a Tcl variable.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) Specifies the list of net, port, or pin objects to return switching activity information on.

Examples

The following example reports the signal_rate and static probability value on all output ports in the design:

```
report_switching_activity -signal_rate -static_probability [all_outputs]
```

See Also

- [power_opt_design](#)
- [report_default_switching_activity](#)
- [report_power](#)
- [reset_default_switching_activity](#)
- [reset_switching_activity](#)
- [set_default_switching_activity](#)
- [set_switching_activity](#)

report_timing

Report timing paths.

Syntax

```
report_timing [-from args] [-rise_from args] [-fall_from args]
[-to args] [-rise_to args] [-fall_to args] [-through args]
[-rise_through args] [-fall_through args] [-delay_type arg]
[-setup] [-hold] [-max_paths arg] [-nworst arg] [-unique_pins]
[-path_type arg] [-input_pins] [-label_reused] [-slack_lesser_than arg]
[-slack_greater_than arg] [-group args] [-sort_by arg]
[-no_report_unconstrained] [-user_ignored] [-match_style arg]
[-of_objects args] [-significant_digits arg] [-file arg] [-append]
[-name arg] [-return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-from]</code>	From pins, ports, cells or clocks
<code>[-rise_from]</code>	Rising from pins, ports, cells or clocks
<code>[-fall_from]</code>	Falling from pins, ports, cells or clocks
<code>[-to]</code>	To pins, ports, cells or clocks
<code>[-rise_to]</code>	Rising to pins, ports, cells or clocks
<code>[-fall_to]</code>	Falling to pins, ports, cells or clocks
<code>[-through]</code>	Through pins, ports, cells or nets
<code>[-rise_through]</code>	Rising through pins, ports, cells or nets
<code>[-fall_through]</code>	Falling through pins, ports, cells or nets
<code>[-delay_type]</code>	Type of path delay: Values: max, min, min_max, max_rise, max_fall, min_rise, min_fall Default: max
<code>[-setup]</code>	Report max delay timing paths (equivalent to -delay_type max)
<code>[-hold]</code>	Report min delay timing paths (equivalent to -delay_type min)
<code>[-max_paths]</code>	Maximum number of paths to output when sorted by slack, or per path group when sorted by group: Value >=1 Default: 1
<code>[-nworst]</code>	List up to N worst paths to endpoint: Value >=1 Default: 1
<code>[-unique_pins]</code>	for each unique set of pins, show at most 1 path per path group

Name	Description
<code>[-path_type]</code>	Format for path report: Values: end, summary, short, full, full_clock, full_clock_expanded Default: full_clock_expanded
<code>[-input_pins]</code>	Show input pins in path
<code>[-label_reused]</code>	Label reuse status on pins in the report
<code>[-slack_lesser_than]</code>	Display paths with slack less than this Default: 1e+30
<code>[-slack_greater_than]</code>	Display paths with slack greater than this Default: -1e+30
<code>[-group]</code>	Limit report to paths in this group(s)
<code>[-sort_by]</code>	Sorting order of paths: Values: group, slack Default: slack
<code>[-no_report_unconstrained]</code>	Do not report infinite slack paths
<code>[-user_ignored]</code>	Only report paths which have infinite slack because of set_false_path or set_clock_groups timing constraints
<code>[-match_style]</code>	Style of pattern matching, valid values are ucf, sdc Default: ucf
<code>[-of_objects]</code>	Report timing for these paths
<code>[-significant_digits]</code>	Number of digits to display: Range: 0 to 3 Default: 3
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-name]</code>	Output the results to GUI panel with this name
<code>[-return_string]</code>	return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Report](#), [Timing](#)

Description

This command performs timing analysis on the specified timing paths of the current Synthesized or Implemented Design. By default the tool reports the timing path with the worst calculated slack within each path group. However, you can optionally increase the number of paths and delays reported with the use of the **-nworst** or **-max_paths** arguments.

The timing engine runs in "quad" timing mode, analyzing min and max delays for both slow and fast corners. You can configure the type of analysis performed by the **config_timing_corners** command. However, it is not recommended to change the default because this reduces the timing analysis coverage.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to the GUI, to a file, or returned as a string if desired.

Arguments

-from *args* - (Optional) The starting points of the timing paths to be analyzed. Ports, pins, or cells can be specified as timing path startpoints. You can also specify a clock object, and all startpoints clocked by the named clock will be analyzed.

-rise_from *args* - (Optional) Similar to the **-from** option, but only the rising edge of signals coming from the startpoints are considered for timing analysis. If a clock object is specified, only the paths launched by the rising edge of the clock are considered as startpoints.

-fall_from *args* - (Optional) Similar to the **-from** option, but only the falling edge of signals coming from the startpoints are considered for timing analysis. If a clock object is specified, only the paths launched by the falling edge of the clock are considered as startpoints.

-to *args* - (Optional) The endpoints, or destination objects of timing paths to be analyzed. Ports, pins, and cell objects can be specified as endpoints. A clock object can also be specified, in which case endpoints clocked by the named clock are analyzed.

-rise_to *args* - (Optional) Similar to the **-to** option, but only the rising edge of signals going to the endpoints is considered for timing analysis. If a clock object is specified, only the paths captured by the rising edge of the named clock are considered as endpoints.

-fall_to *args* - (Optional) Similar to the **-to** option, but only the falling edge of signals going to the endpoints is considered for timing analysis. If a clock object is specified, only the paths captured by the falling edge of the named clock are considered as endpoints.

-through *args* - (Optional) Consider only paths through the specified pins, cell instance, or nets during timing analysis. You can specify individual **-through** (or **-rise_through** and **-fall_through**) points in sequence to define a specific path through the design for analysis. The order of the specified through points is important to define a specific path. You can also specify through points with multiple objects, in which case the timing path can pass through any of the specified through objects.

-rise_through *args* - (Optional) Similar to the **-through** option, but timing analysis is only performed on paths with a rising transition at the specified objects.

-fall_through *args* - (Optional) Similar to the **-through** option, but timing analysis is only performed on paths with a falling transition at the specified objects.

-delay_type *arg* - (Optional) Specifies the type of delay to analyze when running the timing report. The valid values are min, max, min_max, max_rise, max_fall, min_rise, min_fall. The default setting for **-delay_type** is max.

-setup - (Optional) Check for setup violations. This is the same as specifying **-delay_type max**.

-hold - (Optional) Check for hold violations. This is the same as specifying **-delay_type min**.

Note **-setup** and **-hold** can be specified together, which is the same as specifying **-delay_type min_max**

-max_paths *arg* - (Optional) The maximum number of paths to output when sorted by slack; or maximum number of paths per path group when sorted by group, as specified by **-sort_by**. This is specified as a value greater than or equal to 1. By default the **report_timing** command will report the single worst timing path, or the worst path per path group.

-nworst *arg* - (Optional) The number of timing paths to output in the timing report. The timing report will return the *N* worst paths based on the specified value, greater than or equal to 1. The default setting is 1.

-unique_pins - (Optional) Show only one timing path for each unique set of pins.

-path_type *arg* - (Optional) Specify the path data to output in the timing report. The default format is `full_clock_expanded`. The valid path types are:

- `end` - Shows the endpoint of the path only, with calculated timing values.
- `summary` - Displays the startpoints and endpoints with slack calculation.
- `short` - Displays the startpoints and endpoints with calculated timing values.
- `full` - Displays the full timing path, including startpoints, through points, and endpoints.
- `full_clock` - Displays full clock paths in addition to the full timing path.
- `full_clock_expanded` - Displays full clock paths between a master clock and generated clocks in addition to the `full_clock` timing path. This is the default setting.

-input_pins - (Optional) Show input pins in the timing path report. For use with **-path_type** `full`, `full_clock`, and `full_clock_expanded`.

-label_reused - (Optional) For designs using incremental place and route (**read_checkpoint -incremental**), label pins with information related to the physical data reused from the specified incremental checkpoint. Reuse labels include:

- `R` : Cell placement and routing to this pin are reused.
- `PNR` : Cell placement is reused but routing to this pin is not reused.
- `NR` : Neither cell placement nor routing to this pin is reused.
- `N` : The cell, net, or pin is new. It does not exist in the incremental checkpoint.

-slack_lesser_than *arg* - (Optional) Report timing on paths with a calculated slack value less than the specified value. Used with **-slack_greater_than** to provide a range of slack values of specific interest.

-slack_greater_than *arg* - (Optional) Report timing on paths with a calculated slack value greater than the specified value. Used with **-slack_lesser_than** to provide a range of slack values of specific interest.

-group *args* - (Optional) Report timing for paths in the specified path groups. Currently defined path groups can be determined with the **get_path_groups** command.

Note Each clock creates a path group. Path groups can also be defined with the **group_path** command.

-sort_by [slack | group] - (Optional) Sort timing paths in the report by slack values, or by path group. Valid values are `slack` or `group`. By default, the **report_timing** command reports the worst, or **-nworst**, timing paths in the design. However, with **-sort_by group**, the **report_timing** command returns the worst, or **-nworst**, paths of each path group.

-no_report_unconstrained - (Optional) Do not report timing on unconstrained paths. As a default, **report_timing** will include unconstrained paths which will have infinite slack.

-user_ignored - (Optional) Report only the paths that are usually ignored by timing due to presence of **set_false_path** or **set_clock_groups** constraints.

Note The **-user_ignored** and **-no_report_unconstrained** options are mutually exclusive and cannot be specified together

-match_style [sdc | ucf] - (Optional) Indicates that the patterns for objects matches UCF constraints or SDC constraints. The default is UCF.

-of_objects *args* - (Optional) Report timing on the specified timing path objects. Used with the **get_timing_paths** command.

Note The **-of_objects** option cannot be used with the various forms of **-from**, **-to**, or **-through** options which are also used to identify timing paths to report

-significant_digits *arg* - (Optional) The number of significant digits in the output results. The valid range is 0 to 3. The default setting is 3 significant digits.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-name *arg* - (Optional) Specifies the name of the results set for the GUI.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the timing for the 5 worst paths in the design, reporting the full timing path, including input pins, with timing values:

```
report_timing -nworst 5 -path_type full -input_pins
```

The following example shows the use of the multiple through points to define both a specific path (through state_reg1) and alternate paths (through count_3 or count_4), and writes the timing results to the specified file:

```
report_timing -from go -through {state_reg1} -through { count_3 count_4 } \
-to done -path_type summary -file C:/Data/timing1.txt
```

See Also

- [get_path_groups](#)
- [get_timing_paths](#)
- [group_path](#)
- [place_design](#)
- [report_timing_summary](#)
- [route_design](#)
- [set_clock_groups](#)
- [set_false_path](#)
- [set_msg_limit](#)

report_timing_summary

Report timing summary.

Syntax

```
report_timing_summary [-check_timing_verbose] [-delay_type arg]
[-no_detailed_paths] [-setup] [-hold] [-max_paths arg] [-nworst arg]
[-path_type arg] [-label_reused] [-input_pins] [-slack_lesser_than arg]
[-report_unconstrained] [-significant_digits arg] [-no_header]
[-file arg] [-append] [-name arg] [-return_string] [-datasheet]
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-check_timing_verbose]</code>	produce a verbose report when checking the design for potential timing problems
<code>[-delay_type]</code>	Type of path delay: Values: max, min, min_max Default: min_max
<code>[-no_detailed_paths]</code>	do not report timing paths for each clock and path group analyzed
<code>[-setup]</code>	Report max delay timing paths (equivalent to -delay_type max)
<code>[-hold]</code>	Report min delay timing paths (equivalent to -delay_type min)
<code>[-max_paths]</code>	Maximum number of paths to report per clock or path group: Value >=1 Default: 1
<code>[-nworst]</code>	List up to N worst paths to endpoint: Value >=1 Default: 1
<code>[-path_type]</code>	Format for path report: Values: end summary short full full_clock full_clock_expanded Default: full_clock_expanded
<code>[-label_reused]</code>	Label reuse status on pins in the report
<code>[-input_pins]</code>	Show input pins in path
<code>[-slack_lesser_than]</code>	Display paths with slack less than this Default: 1e+30
<code>[-report_unconstrained]</code>	report unconstrained paths
<code>[-significant_digits]</code>	Number of digits to display: Range: 0 to 3 Default: 3
<code>[-no_header]</code>	do not generate a report header
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)

Name	Description
[-append]	Append the results to file, don't overwrite the results file
[-name]	Output the results to GUI panel with this name
[-return_string]	return report as string
[-datasheet]	Include data sheet report
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#), [Timing](#)

Description

Generate a timing summary to help understand if the design has met timing requirements. The timing summary can be run on an open Synthesized or Implemented Design.

The timing summary report includes the following information:

- Timer Settings - Details the timing engine settings used to generate the timing information in the report.
- Check Timing - Contains the same information that is produced by the **check_timing** command, which summarizes potential timing issues.
- Design Timing Summary - Provides a summary of the timing of the design, including values for worst and total negative slack (WNS/TNS), worst and total hold slack (WHS/THS), and component switching limits (CSL).
- Clock Definitions - Contains the same information that is produced by the **report_clocks** command, showing all the clocks that were created for the design, either by **create_clock**, **create_generated_clock**, or automatically by the tool.
- Intra-Clock Table - Summarizes timing paths with the same source and destination clocks.
- Inter-Clock Table - Summarizes timing paths with different source and destination clocks.
- Path Group Table - Shows default path groups and user-defined path groups created by the **group_path** command.
- Timing Details - Contains detailed timing paths, both max delay and min delay, as well as component switching limits for each clock defined, similar to the **report_timing** command.
- Data sheet - Contains the same information that is produced by the **report_datasheet** command. It contains the timing characteristics of a design at the I/O ports. The data sheet information is added to the summary report only when the **-datasheet** option is specified.

This command is automatically run during implementation as part of the **launch_runs** command.

Note By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

Arguments

-check_timing_verbose - (Optional) Output a verbose timing summary report.

-delay_type *arg* - (Optional) Specifies the type of delay to analyze when running the timing report. The valid values are min, max, min_max. The default setting for **-delay_type** is min_max.

-no_detailed_paths - (Optional) Do not report the full timing path for each clock or path group analyzed.

-setup - (Optional) Check for setup violations. This is the same as specifying **-delay_type max**.

-hold - (Optional) Check for hold violations. This is the same as specifying **-delay_type min**.

Note **-setup** and **-hold** can be specified together, which is the same as specifying **-delay_type min_max**

-max_paths *arg* - (Optional) The maximum number of paths to report per clock or per path group. This is specified as a value greater than or equal to 1. By default the **report_timing_summary** command will report the single worst timing path, or the worst path per path group.

-nworst *arg* - (Optional) The number of timing paths to output in the timing report. The timing report will return the *N* worst paths to endpoints based on the specified value, greater than or equal to 1. The default setting is 1.

-path_type *arg* - (Optional) Specify the path data to output in the timing summary report. The default format is full_clock_expanded. The valid path types are:

- **end** - Shows the endpoint of the path only, with calculated timing values.
- **summary** - Displays the startpoints and endpoints with slack calculation.
- **short** - Displays the startpoints and endpoints with calculated timing values.
- **full** - Displays the full timing path, including startpoints, through points, and endpoints.
- **full_clock** - Displays full clock paths in addition to the full timing path.
- **full_clock_expanded** - Displays full clock paths between a master clock and generated clocks in addition to the full_clock timing path. This is the default setting.

-label_reused - (Optional) For designs using incremental place and route (**read_checkpoint -incremental**), label pins with information related to the physical data reused from the specified incremental checkpoint. Reuse labels include:

- R : Cell placement and routing to this pin are reused.
- PNR : Cell placement is reused but routing to this pin is not reused.
- NR : Neither cell placement nor routing to this pin is reused.
- N : The cell, net, or pin is new. It does not exist in the incremental checkpoint.

-input_pins - (Optional) Show input pins in the timing path report. For use with **-path_type** full, full_clock, and full_clock_expanded.

-slack_less_than *arg* - (Optional) Report timing on paths with a calculated slack value less than the specified value. Used with **-slack_greater_than** to provide a range of slack values of specific interest.

-report_unconstrained - (Optional) Report timing on unconstrained paths in the current design. As a default, the **report_timing_summary** command will not include unconstrained timing paths.

-significant_digits *arg* - (Optional) The number of significant digits in the output results. The valid range is 0 to 3. The default setting is 3 significant digits.

-no_header - (Optional) Do not add header information to the report. This can be useful when returning the timing summary report to a string for further processing.

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-name *arg* - (Optional) Specifies the name of the results set for the GUI. Timing summary reports in the GUI can be deleted by the **delete_timing_results** command.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-datasheet - (Optional) Generate data sheet information to add to the summary report.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the timing summary of the current design:

```
report_timing_summary
```

The following example reports the hold timing summary of the current design, including unconstrained paths, with the specified options:

```
report_timing_summary -delay_type min -path_type full_clock_expanded \
    -report_unconstrained -max_paths 2 -nworst 1 -significant_digits 2 \
    -input_pins -name {timing_6}
```


See Also

- [check_timing](#)
- [create_clock](#)
- [create_generated_clock](#)
- [delete_timing_results](#)
- [get_path_groups](#)
- [get_timing_paths](#)
- [group_path](#)
- [report_clocks](#)
- [report_timing](#)
- [set_msg_limit](#)
- [report_datasheet](#)

report_transformed_primitives

Report details of Unisim primitive transformations.

Syntax

```
report_transformed_primitives [-file arg] [-append] [-return_string]  
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-file]</code>	Output file
<code>[-append]</code>	Append the results to file
<code>[-return_string]</code>	return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Report](#)

Description

Report the transformed primitives in the current design.

As part of the process of opening the Synthesized design, and loading it into memory, the tool will transform legacy netlist primitives to the supported subset of Unisim primitives.

As a default this report will be written to the standard output. However, the report can also be written to a file or returned to a Tcl string variable for further processing.

Arguments

-file *arg* - (Optional) Write the transformed primitives report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-return_string - (Optional) Directs the output to a Tcl string. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the transformed primitives in the current design, and returns the result to the specified Tcl variable:

```
set transPrim [ report_transformed_primitives -return_string ]
```

report_utilization

Compute utilization of device and display report.

Syntax

```
report_utilization [-file arg] [-append] [-pblocks args] [-cells args]
[-return_string] [-packthru] [-name arg] [-no_primitives] [-omit_locs]
[-hierarchical] [-hierarchical_depth arg] [-quiet] [-verbose]
```

Returns

Report

Usage

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append the results to file, don't overwrite the results file
[-pblocks]	Report utilization of given list of pblocks
[-cells]	Report utilization of given list of cells
[-return_string]	Return report as string
[-packthru]	Reports LUTs used exclusively as pack-thru
[-name]	Output the results to GUI panel with this name
[-no_primitives]	Removes "Primitives Section" from report_utilization o/p.
[-omit_locs]	Removes "Loced" column from report_utilization o/p.
[-hierarchical]	Generates text-based hierarchical report.
[-hierarchical_depth]	Specifies the depth level for textual hierarchical report Default: 0
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

Description

Report the utilization of resources on the target part by the current synthesized or implemented design. The report is returned to the standard output, unless the **-file**, **-return_string**, or **-name** arguments are specified.

Though resource utilization can be reported early in the design process, the report will be more accurate as the design progresses from synthesis through implementation.

Arguments

-file *arg* - (Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless **-append** is also specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append - (Optional) Append the output of the command to the specified file rather than overwriting it.

Note The **-append** option can only be used with the **-file** option

-pblocks *arg* - (Optional) Report the resources utilized by one or more Pblocks in the design.

-cells *arg* - (Optional) Report the resources utilized by on or more hierarchical cells in the current design.

-return_string - (Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note This argument cannot be used with the **-file** option.

-packthru - (Optional) Reports LUTs used for route through purposes. This appears in the utilization report as "LUTs used exclusively as route-thrus".

-name *arg* - (Optional) Specifies the name of the results to output to the GUI.

-no_primitives - (Optional) Remove the Primitives section from the report. The Primitives section reports the number and type of logic primitives used on the device.

-omit_locs - (Optional) Omit the LOCed column from the report. The LOCed column reports the quantity of logic elements that have been placed onto the fabric of the device.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reports the resources utilized by the Pblocks in the design, and writes the results to the specified file:

```
report_utilization -pblocks [get_pblocks] -file C:/Data/FPGA_Design/pb_util.txt
```

See Also

[delete_utilization_results](#)

report_values

Print current simulated value of given HDL objects (variable, signal, wire, or reg).

Syntax

```
report_values [-radix arg] [-quiet] [-verbose] [hdl_objects ...]
```

Returns

Print name and value of HDL objects on the console in textual format

Usage

Name	Description
<code>[-radix]</code>	The radix specifies the radix to use for printing the values of the hdl_objects. Allowed values are: default, dec, bin, oct, hex, unsigned, ascii.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[hdl_objects]</code>	The hdl_objects to report. Default is report_objects [get_objects *]

Categories

[Simulation](#)

Description

Report the values of the specified HDL objects at the current simulation run time.

HDL objects include HDL signals, variables, or constants as defined in the Verilog or VHDL testbench and source files. An HDL signal includes Verilog wire or reg entities, and VHDL signals. Examples of HDL variables include Verilog real, realtime, time, and event.

HDL constants include Verilog parameters and localparams, and VHDL generic and constants. The HDL scope, or scope, is defined by a declarative region in the HDL code such as a module, function, task, process, or begin-end blocks in Verilog. VHDL scopes include entity/architecture definitions, block, function, procedure, and process blocks.

Arguments

-radix *arg* - (Optional) Specifies the radix to use when returning the value of the specified objects. Allowed values are: **default**, **dec**, **bin**, **oct**, **hex**, **unsigned**, and **ascii**.

Note The radix **dec** indicates a signed decimal. Specify the radix **unsigned** when dealing with unsigned data

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

hdl_objects - (Required) Specifies one or more HDL objects to return the values of. The object can be specified by name, or can be returned as an object from the **get_objects** command.

Examples

The following example reports the value of all objects at the current time:

```
report_values [get_objects]
```

This example shows the difference between the **bin**, **dec**, and **unsigned** radix on the value returned from the specified bus:

```
report_values -radix bin /test/bench_VStatus_pad_0_i[7:0]
  Declared: {/test/bench_VStatus_pad_0_i[7:0]} Verilog 10100101
report_values -radix unsigned /test/bench_VStatus_pad_0_i[7:0]
  Declared: {/test/bench_VStatus_pad_0_i[7:0]} Verilog 165
report_values -radix dec /test/bench_VStatus_pad_0_i[7:0]
  Declared: {/test/bench_VStatus_pad_0_i[7:0]} Verilog -91
```

See Also

- [current_time](#)
- [get_objects](#)
- [get_value](#)
- [set_value](#)

reset_default_switching_activity

Reset switching activity on default types.

Syntax

```
reset_default_switching_activity [-static_probability] [-toggle_rate]
[-quiet] [-verbose] type...
```

Returns

Nothing

Usage

Name	Description
<code>[-static_probability]</code>	Reset static probability
<code>[-toggle_rate]</code>	Reset toggle rate
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>type</i>	Resets the default seed values to tool defaults on specified types for vector-less propagation engine. List of valid default type values: input, input_set, input_reset, input_enable, register, dsp, bram_read_enable, bram_write_enable, output_enable, clock, all

Categories

Power

Description

Reset the attributes of the default switching activity on nets, ports, pins, and cells in the design.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-static_probability - (Optional) Reset the static probability of the specified type.

-toggle_rate - (Optional) Reset the toggle rate of the specified type.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

type - The type to reset. List of valid values: input, input_set, input_reset, input_enable, register, dsp, bram_read_enable, bram_write_enable, output_enable, clock, all.

Examples

The following example resets the toggle rate and static probability value on all design output ports:

```
reset_default_switching_activity -toggle_rate -static_probability all
```

See Also

- [power_opt_design](#)
- [report_default_switching_activity](#)
- [report_power](#)
- [report_default_switching_activity](#)
- [report_switching_activity](#)
- [reset_switching_activity](#)
- [set_default_switching_activity](#)
- [set_switching_activity](#)

reset_drc

Remove DRC report.

Syntax

```
reset_drc [-name arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-name]	DRC result name
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[DRC](#), [Report](#)

Description

Clear the DRC results from the specified named result set.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Optional) Specifies the name of the DRC results to be cleared. The name is established by the **-name** argument in the **report_drc** command.

Examples

The following example clears the specified results set from memory and the GUI:

```
reset_drc DRC1
```

See Also

[report_drc](#)

reset_drc_check

Reset one or more drc checks to factory defaults.

Syntax

```
reset_drc_check [-quiet] [-verbose] [rules...]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[rules]	The list of checks to reset.

Categories

DRC, Object

Description

Reset the specified DRC checks to the defaults provided by the Vivado Design Suite. This will restore the DRC check to its default configuration, including any changes to the IS_ENABLED or SEVERITY properties.

The IS_ENABLED property can be modified on a specific DRC check to disable the rule from being checked, even when it is specified either directly in the **report_drc** command, or as part of a ruledeck.

The SEVERITY property is a string property that can be modified to change the severity associated with a specific DRC rule when a violation is found during the **report_drc** command. The supported values are: FATAL, ERROR, "CRITICAL WARNING", WARNING, ADVISORY

```
reset_drc_check [-quiet] [-verbose] [<rules>...]
```

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

rules - (Required) The list of one or more DRC rules to reset to the tool defaults.

Examples

The following example modifies the IS_ENABLED property for the ROAS-1 rule, modifies the SEVERITY property for the RFFC-1 rule, and then restores the default settings for all checks:

```
set_property IS_ENABLED false [get_drc_checks ROAS-1]
set_property SEVERITY "Critical Warning" [get_drc_checks RFFC-1]
reset_drc_check [get_drc_checks]
```

See Also

- [add_drc_checks](#)
- [get_drc_checks](#)
- [report_drc](#)
- [set_property](#)

reset_hw_ila

Reset hardware ILA control properties to default values.

Syntax

```
reset_hw_ila [-reset_compare_values arg] [-quiet] [-verbose]  
[hw_ilas...]
```

Returns

Nothing

Usage

Name	Description
<code>[-reset_compare_values]</code>	Reset associated hardware probe compare values.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[hw_ilas]</code>	List of hardware ILA objects. Default: Current hardware ILA

Categories

[Hardware](#)

reset_hw_vio_activity

Reset hardware VIO ACTIVITY_VALUE properties, for hardware probes associated with specified hardware VIO objects.

Syntax

```
reset_hw_vio_activity [-quiet] [-verbose] [hw_vios...]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_vios]	List of hardware VIO objects.

Categories

[Hardware](#)

reset_hw_vio_outputs

Reset hardware VIO core outputs to initial values.

Syntax

```
reset_hw_vio_outputs [-quiet] [-verbose] [hw_vios...]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_vios]	List of hardware VIO objects.

Categories

[Hardware](#)

reset_msg_config

Resets or removes a message control rule previously defined by the set_msg_config command.

Syntax

```
reset_msg_config [-string args] [-id arg] [-severity arg] [-limit]  
[-suppress] [-count] [-default_severity] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-string]	A qualifier, apply the selected operation only to messages that contain the given strings Default: empty
[-id]	A qualifier, the message id to match. If not specified, all message ids will be matched
[-severity]	A qualifier, apply the selected operation only to messages at the given severity level
[-limit]	reset the limit values for message controls that match the given qualifiers for the current project
[-suppress]	stop suppressing messages that match the given qualifiers for the current project
[-count]	reset the count of messages for all message controls that match the given qualifiers for the current project. This will prevent messages from being suppressed by a -limit control until the message count once again exceeds the specified limit.
[-default_severity]	reset the message severity of all messages controls for the current project that match the given qualifiers to their default value
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

This command restores the default settings of the message limits or severity for messages returned by the Vivado tool, or can unsuppress previously suppressed messages, as configured by the **set_msg_config** command.

You can only perform one reset action for each **reset_msg_config** command. An error is returned if more than one action is attempted in a single **reset_msg_config** command.

Message qualifiers of string, ID, and severity are used to determine which messages are reset by the **reset_msg_config** command. Multiple qualifiers have an AND relationship; the configuration rule will be applied only to messages matching all qualifiers.

Note You must supply at least one message qualifier to identify a message or group of messages to apply the command to, or an error is returned.

To report the current rule configurations for messages, use the **get_msg_config** command.

Arguments

-string args - (Optional) Apply the selected operation only to messages that contain the given list of strings. Strings must be enclosed in braces, and multiple strings can be specified separated by spaces:

```
{{Vivado} {All Programmable}}
```

Note Strings are case sensitive.

-id arg - (Optional) Reset messages matching the specified message ID. The message ID is included in all returned messages. For example, "Common 17-54" and "Netlist 29-28".

Note A wildcard * indicates all message IDs should be reset

-severity arg - Reset messages with the specified message severity. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

Note Since this is a two word value, it must be enclosed in {}.

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

-limit - (Optional) Reset the message limit for messages matching the string, ID, or severity qualifiers.

-suppress - (Optional) Reset, or unsuppress messages matching the string, ID, or severity qualifiers.

-count - (Optional) Reset the message count for messages matching the string, ID, or severity qualifiers.

-default_severity - (Optional) Restore the default message severity for messages matching the string, ID, or severity qualifiers.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example resets the specified INFO message to its default severity:

```
reset_msg_config -id "Common 17-81" -default_severity
```

The following example unsuppresses messages with the specified message ID:

```
reset_msg_config -id {HDL 9-1654} -suppress
```

See Also

- [get_msg_config](#)
- [set_msg_config](#)

reset_msg_count

Reset message count.

Syntax

```
reset_msg_count [-quiet] [-verbose] id
```

Returns

New message count

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>id</i>	Unique message Id to be reset, e.g. "Common 17-99". "reset_msg_count -id *" reset all counters

Categories

[Report](#)

Description

Reset the message count for the specified message ID to 0. This restarts the message counter toward the specified message limit. This can be used to reset the count of specific messages that may be reaching the limit, or reset the count of all messages returned by the tool.

Every message delivered by the tool has a unique global message ID that consists of an application sub-system code and a message identifier. This results in a message ID that looks like the following:

```
"Common 17-54"  
"Netlist 29-28"  
"Synth 8-3295"
```

You can get the current message count for a specific message ID using the **get_msg_count** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

id - (Required) Specifies the message ID to reset the count to 0. Specify * to reset the count of all messages to 0.

Examples

The following example resets the message count for all messages:

```
reset_msg_count *
```

See Also

- [get_msg_count](#)
- [set_msg_config](#)

reset_msg_limit

Reset message limit.

Syntax

```
reset_msg_limit [-severity arg] [-id arg] [-quiet] [-verbose]
```

Returns

New message limit

Usage

Name	Description
[-severity]	Message severity to be reset (not valid with -id,) e.g. "ERROR" or "CRITICAL WARNING" Default: ALL
[-id]	Unique message Id to be reset (not valid with -severity,) e.g "Common 17-99"
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Restores the default message limit. The command can be used to restore the default limit for a specific message ID, for a specific message severity, or for all messages returned.

The current default limit for all messages returned is 4,294,967,295.

Arguments

-id value - (Optional) The ID of a specific message for which to change the message limit. Each message returned contains its own ID. For instance, "Common 17-54" and "Netlist 29-28".

-severity value - (Optional) The severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

Note Since this is a two word value, it must be enclosed in {} or "".

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example resets the message limit of the specified message ID:

```
reset_msg_limit -id "Netlist 29-28"
```

See Also

- [get_msg_config](#)
- [get_msg_count](#)
- [get_msg_limit](#)
- [reset_msg_config](#)
- [set_msg_config](#)
- [set_msg_limit](#)

reset_msg_severity

Reset Message Severity by ID.

Syntax

```
reset_msg_severity [-quiet] [-verbose] id
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>id</i>	Unique message id to be set, e.g. ""Common 17-99""

Categories

[Report](#)

Description

Restores the specified message ID to its default severity.

Use this command after **set_msg_severity** to restore a specific message ID to its original severity level.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

id - (Required) The ID of the message for which the severity should be reset.

Examples

The following example restores the severity of message ID **common-99** to its original severity:

```
reset_msg_severity common-99
```

The following example restores the severity of message ID **Netlist-1129** to its original severity:

```
reset_msg_severity Netlist-1129
```

See Also

[set_msg_severity](#)

reset_operating_conditions

Reset operating conditions to tool default for power estimation.

Syntax

```
reset_operating_conditions [-voltage args] [-grade] [-process]
[-junction_temp] [-ambient_temp] [-thetaja] [-thetasa] [-airflow]
[-heatsink] [-thetajb] [-board] [-board_temp] [-board_layers] [-quiet]
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[-voltage]	Resets voltage value. Supported voltage supplies vary by family.
[-grade]	Resets temperature grade
[-process]	Resets process
[-junction_temp]	Resets Junction Temperature
[-ambient_temp]	Resets Ambient Temperature
[-thetaja]	Resets ThetaJA
[-thetasa]	Resets ThetaSA
[-airflow]	Resets Airflow
[-heatsink]	Resets dimensions of heatsink
[-thetajb]	Resets ThetaJB
[-board]	Resets Board type
[-board_temp]	Resets Board Temperature
[-board_layers]	Resets Board layers
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

SDC, Power

Description

Resets the specified operating conditions to their default values. If no operating conditions are specified, all operating conditions are reset to their default values.

Operating conditions can be set using the **set_operating_conditions** command. The current values can be determined using the **report_operating_conditions** command. The environmental operating conditions of the device are used for power analysis when running the **report_power** command, but are not used during timing analysis.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-voltage *args* - (Optional) Reset the voltage supply to the default value. The voltage supply and its default depend on the device family.

-grade - (Optional)) Reset the temperature grade of the selected device. The default value is "commercial".

-process - (Optional) Reset the manufacturing process for the target device. The default process is "typical".

-junction_temp - (Optional) Reset the junction temperature for the target device. The default value is "auto".

-ambient_temp - (Optional) Reset the ambient temperature of the design. The default setting is "default".

-thetaja - (Optional) Reset the Theta-JA thermal resistance. The default setting is "auto".

-thetasa - (Optional) Reset the Theta-SA thermal resistance. The default setting is "auto".

-airflow - (Optional) Reset the Linear Feet Per Minute (LFM) airflow. The default setting varies by device family.

-heatsink - (Optional) Reset the heatsink profile. The default setting is "medium".

-thetajb - (Optional) Reset the Theta-JB thermal resistance. The default setting is "auto".

-board - (Optional) Reset the board size to be used for modeling. The default value is "medium".

-board_temp *arg* - (Optional) Reset the board temperature to the default setting.

-board_layers - (Optional) Reset the number of board layers to be used for modeling to the default setting of "12to15".

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example resets all the operating conditions for the design to their default setting:

```
reset_operating_conditions
```

The following example resets the junction, ambient, and board temperature for the design to their default settings:

```
reset_operating_conditions -junction_temp -ambient_temp -board_temp
```

The following example resets the voltage supply Vccint to its default value:

```
reset_operating_conditions -voltage Vccint
```

See Also

- [report_operating_conditions](#)
- [report_power](#)
- [set_operating_conditions](#)

reset_param

Reset a parameter.

Syntax

```
reset_param [-quiet] [-verbose] name
```

Returns

Original value

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Parameter name

Categories

[PropertyAndParameter](#)

Description

Restores a user-definable configuration parameter that has been changed with the **set_param** command to its default value.

You can use the **report_param** command to see which parameters are currently defined.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of a parameter to reset. You can only reset one parameter at a time.

Examples

The following example restores the `tcl.statsThreshold` parameter to its default value:

```
reset_param tcl.statsThreshold
```

See Also

- [get_param](#)
- [list_param](#)
- [report_param](#)
- [set_param](#)

reset_project

Reset current project.

Syntax

```
reset_project [-exclude_runs] [-exclude_ips] [-exclude_sim_runs]  
[-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-exclude_runs]</code>	Do not reset runs
<code>[-exclude_ips]</code>	Do not reset ips
<code>[-exclude_sim_runs]</code>	Do not reset simulation runs
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Project](#)

Description

Reset the current project to its starting condition, with source and constraint files, by cleaning out the various output files created during synthesis, simulation, implementation, and write_bitstream. Also resets the state of the project to the start of the design flow.

Arguments

-exclude_runs - (Optional) Exclude the `<project>.runs` folder from the reset process. In this case, the runs folder will be preserved, while the rest of the project data will be removed.

-exclude_ips - (Optional) Exclude the `<project>.srcs/sources_1/ip` folder from the reset process. In this case, the IP folder, containing the IP cores and generated targets, will be preserved, while the rest of the project data will be removed.

-exclude_sim_runs - (Optional) Exclude the `<project>.sim` folder from the reset process. In this case, the simulation folder will be preserved, while the rest of the project data will be removed.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Example

The following example resets the current project, while preserving the simulation run data, and returning all messages regardless of message limits:

```
reset_project -exclude_sim_runs -verbose
```

See Also

- [create_project](#)
- [current_project](#)

reset_property

Reset property on object(s).

Syntax

```
reset_property [-quiet] [-verbose] property_name objects...
```

Returns

The value that was set if success, "" if failure

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>property_name</i>	Name of property to reset
<i>objects</i>	Objects to set properties

Categories

[Object](#), [PropertyAndParameter](#)

Description

Restores the specified property to its default value on the specified object or objects. If no default is defined for the property, the property is unassigned on the specified object.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

property_name - (Required) The name of the property to be reset.

objects - (Required) One or more objects on which the property will be restored to its default value.

Examples

The following example resets the ALL_PROPS property on all cells:

```
reset_property ALL_PROPS [get_cells]
```

See Also

- [create_property](#)
- [get_cells](#)
- [get_property](#)
- [list_property](#)
- [list_property_value](#)
- [report_property](#)
- [set_property](#)

reset_run

Reset an existing run.

Syntax

```
reset_run [-prev_step] [-from_step arg] [-noclean_dir] [-quiet]  
[-verbose] run
```

Returns

Nothing

Usage

Name	Description
<code>[-prev_step]</code>	Reset last run step
<code>[-from_step]</code>	First Step to reset
<code>[-noclean_dir]</code>	Do not remove all output files and directories from disk
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>run</code>	Run to modify

Categories

[Project](#)

Description

Resets the specified run to an unimplemented or unsynthesized state. Use this command to reset the run to prepare it to be run again.

Arguments

-prev_step - (Optional) Reset an implementation run from the last step completed. This can be used to reset an implementation run that is only partially completed because it was launched with the **launch_runs -to_step** command.

-from_step arg - (Optional) Reset an implementation run from a specified step. This lets you restart a run from the specified step using the **launch_runs -next_step** command. Valid step values include:

- **opt_design** - Optionally optimize the logical design to more efficiently use the target device resources.
- **power_opt_design** - Optionally optimize elements of the logic design to reduce power demands of the implemented FPGA.
- **place_design** - Place logic cells onto the target device. This is a required step.
- **power_opt_design (Post-Place)** - Optionally optimize power demands of the placed logic elements.
- **phys_opt_design** - Optionally optimize design timing by replicating drives of high-fanout nets to better distribute the loads.
- **route_design** - Route the connections of the design onto the target FPGA. This is a required step.
- **write_bitstream** - Generate a bitstream file for Xilinx device configuration. This is a required step.

-noclean_dir - (Optional) Do not clean up the run files output to the run directory. As a default the tool will delete the run directory and all files within that directory when resetting the run in order to ensure a clean start when the run is reimplemented. This argument directs the tool not to remove the run directory and its content when resetting the run. In this case, when the run is reimplemented a new run directory will be created in the project runs directory.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

run - (Required) The name of the run to reset.

Examples

The following example resets the implementation run:

```
reset_run impl_1
```

Note The run directory and its contents will be removed from the hard disk since **-noclean_dir** is not specified.

The following example resets the synthesis run, but disables the cleanup of the run directory:

```
reset_run -noclean_dir synth_1
```

In this example, because **-noclean_dir** is specified, the synth_1 run directory is not removed and a new run directory called synth_1_2 will be created when the run is launched.

See Also

- [create_run](#)
- [launch_runs](#)
- [opt_design](#)
- [place_design](#)
- [route_design](#)

reset_simulation

Reset an existing simulation run.

Syntax

```
reset_simulation [-mode arg] [-type arg] [-quiet] [-verbose] [simset]
```

Returns

Nothing

Usage

Name	Description
[-mode]	Remove generated data for the specified mode. Values: behavioral, post-synthesis, post-implementation Default: behavioral
[-type]	Remove generated data for the specified type. Applicable mode is post-synthesis or post-implementation. Values: functional, timing
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[simset]	Name of the simulation fileset to reset

Categories

[Simulation](#)

reset_ssn

Clear a SSN results set from memory.

Syntax

```
reset_ssn [-quiet] [-verbose] name
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of the set of results

Categories

[Report](#)

Description

Clear the SSN results from the specified named result set.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) Specifies the name of the results to be cleared.

Examples

The following example clears the specified results set from memory:

```
reset_ssn SSN1
```

See Also

[report_ssn](#)

reset_switching_activity

Reset switching activity on specified objects.

Syntax

```
reset_switching_activity [-static_probability] [-signal_rate] [-hier]  
[-quiet] [-verbose] objects...
```

Returns

Nothing

Usage

Name	Description
<code>[-static_probability]</code>	Reset static probability
<code>[-signal_rate]</code>	Reset signal rate
<code>[-hier]</code>	Hierarchically resets the switching activity on a hierarchical cells provided as .
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>objects</i>	Objects to reset switching activity on

Categories

Power

Description

Resets the attributes of the switching activity on nets, ports, pins, and cells in the design.

Note This command operates silently and does not return direct feedback of its operation.

The switching activity can be defined using the **set_switching_activity** command. The current switching activity defined for a specific port, pin, net, or cell can be found by using the **report_switching_activity** command.

Arguments

-static_probability - (Optional) Reset the static probability of the specified object.

-signal_rate - (Optional) Reset the signal rate of the specified object.

-hier - (Optional) Reset the switching activity across all levels of a hierarchical object. Without **-hier**, the switching activity is applied to the specified *objects* at the current level of the hierarchy.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) The list of objects for which to reset the switching activity.

Examples

The following example resets the signal_rate and static probability value on all output ports:

```
reset_switching_activity -signal_rate -static_probability [all_outputs]
```

See Also

- [power_opt_design](#)
- [report_default_switching_activity](#)
- [report_power](#)
- [report_switching_activity](#)
- [reset_default_switching_activity](#)
- [set_default_switching_activity](#)
- [set_switching_activity](#)

reset_target

Reset target data for the specified source.

Syntax

```
reset_target [-quiet] [-verbose] name objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	List of targets to be reset, or 'all' to reset all generated targets
<i>objects</i>	The objects for which data needs to be reset

Categories

[Project](#), [XPS](#), [IPFlow](#)

Description

Remove the current target data for the specified IP core. This deletes any files that were delivered during generation of the specified targets. This does not remove the core from the current project, but does remove the associated target data from its referenced location.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) Specifies the name of the type of target to reset. Valid values are:

- **all** - Reset all targets for the specified core.
- **synthesis** - Reset the synthesis netlist for the specified core. This will remove the netlist files for the specified core.
- **simulation** - Reset the simulation netlist for the specified core.
- **instantiation_template** - Reset the instantiation template for the specified core.

objects - (Required) The IP core objects to remove the target data from.

Examples

The following example resets the instantiation template for the specified IP core:

```
reset_target instantiation_template [get_ips blk_mem*]
```

See Also

[generate_target](#)

reset_timing

Resets the timing information on the current design.

Syntax

```
reset_timing [-invalid] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-invalid]	Also reset invalid timing constraints.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#), [Timing](#)

Description

Reset the timing data for the current design. Use this command to clear the current in-memory timing data, and force the timing engine to reevaluate the design comprehensively rather than iteratively.

Note This command deletes the in-memory timing view, not the timing report. Use the **delete_timing_results** command to delete the reported timing information.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example clears the current timing data from memory:

```
reset_timing
```

See Also

- [delete_timing_results](#)
- [report_timing](#)

reset_ucf

Clear floorplan constraints read in from a file.

Syntax

```
reset_ucf [-quiet] [-verbose] file
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	UCF file to be reset

Categories

Floorplan

Description

Clear placement constraints defined in the specified UCF constraints file from the current design. This command requires an open design.

The constraints found in the specified file will be removed from the current design, but are not immediately saved to the constraints file. The specified constraint file will be updated when you use the **save_design** command to rewrite the constraints.

You can save the constraints to a new file without saving the design by using the **write_ucf** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The UCF file to be reset.

Note The tool will look for the specified file in the constraint filesets

Examples

The following example removes placement constraints found in the specified file:

```
reset_ucf top_full.ucf
```

resize_net_bus

Resize net bus in the current design.

Syntax

```
resize_net_bus [-from arg] [-to arg] [-quiet]  
[-verbose] net_bus_name ...
```

Returns

Nothing

Usage

Name	Description
<code>[-from]</code>	New starting bus index
<code>[-to]</code>	New ending bus index
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>net_bus_name</code>	Name of the net bus to resize

Categories

[Netlist](#)

Description

Resize an existing bus net, to grow the bus, shrink the bus, or renumber the current range of indexes. You can only do a single grow, shrink, or renumber operation with each command.

- You can grow the bus by indicating a new range of indexes outside the current range of indexes. Growing the bus leaves existing bits connected as they currently are.
- You can shrink the bus by indicating a new range of indexes inside the current range of indexes. Shrinking the bus, eliminates connections to removed bits, but leaves the remaining bits connected as they currently are.
- You can renumber the current bus indexes by providing a new range of indexes with the same width as the current range. Renumbering bits changes bus bit numeric identifiers, but doesn't otherwise change connections.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source filesset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

This command returns nothing if successful, and returns an error if it fails.

Arguments

-from *arg* - (Optional) The new starting index of the specified bus net.

-to *arg* - (Optional) The new ending index of the specified bus.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

net_bus_name - (Required) The names of an existing bus net.

Example

The following example creates a new 24-bit bus, then rennumbers the bus indexes to include negative indexes, and then resizes the bus to shrink it to an 8-bit bus:

```
create_net tempBus -from 23 -to 0
resize_net_bus tempBus -from -12 -to 11
resize_net_bus tempBus -from 0 -to 7
```

See Also

- [connect_net](#)
- [create_net](#)
- [create_pin](#)
- [create_port](#)
- [disconnect_net](#)
- [get_nets](#)
- [remove_net](#)
- [resize_pin_bus](#)
- [resize_port_bus](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

resize_pblock

Move, resize and add and/or remove UCF ranges.

Syntax

```
resize_pblock [-add args] [-remove args] [-from args] [-to args]
[-replace] [-locs arg] [-quiet] [-verbose] pblock
```

Returns

Nothing

Usage

Name	Description
[-add]	Add site ranges(s)
[-remove]	Remove site ranges(s)
[-from]	Site range(s) to move
[-to]	Site range destination(s)
[-replace]	Remove all existing ranges
[-locs]	LOC treatment Default: keep_all
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>pblock</i>	Pblock to resize

Categories

Floorplan, XDC

Description

Place, resize, move, or remove the specified Pblock. The Pblock must have been created using the **create_pblock** command.

A Pblock consists of a group of cells that can be assigned to one or more independent or overlapping rectangles. Using the various options defined below, you can add sites to a rectangle, or remove sites from a rectangle, or define a new rectangle to be associated with an existing Pblock.

Arguments

-add *args* - (Optional) Add the specified range of sites to the Pblock. The SLICE range is specified as a rectangle from one corner to the diagonally opposite corner. For example SLICE_X0Y0:SLICE_X20Y12.

Note Multiple site types are added as separate rectangles.

-remove *args* - (Optional) Remove the specified range of sites from the Pblock. Removing sites from a Pblock may result in the Pblock being broken into multiple smaller rectangles to enforce the requirement that Pblocks are defined as one or more rectangles.

-from *args* - (Optional) The **-from** and **-to** options must be used as a pair, and specify a site or range of sites to relocate from one location to another.

-to *args* - (Optional) The **-from** and **-to** options must be used as a pair, and specify a site or range of sites to relocate from one location to another.

-locs *args* - (Optional) Specifies how the placed logic in the Pblock will be handled as the Pblock is moved or resized. Valid values are:

- **keep_all** - leave all locs placed as they are currently. This is the default setting when **-locs** is not specified. Logic that is placed outside of the Pblock will no longer be assigned to the Pblock.
- **keep_only_fixed** - Specifies that only user-placed logic (fixed) will be preserved. Unfixed placed logic will be unplaced.
- **keep_none** - Unplace all logic.
- **move** - Specifies that all locs should be moved relative to the coordinates of the Pblock.
- **move_unfixed** - Specifies that only the unfixed placed elements should be moved. Logic placed by the user (fixed) will not be moved.
- **trim** - Specifies that logic that falls outside of the new Pblock boundaries will be unplaced. Any placed logic that still falls inside of the Pblock boundary will be left placed as it is.
- **trim_unfixed** - Trim only the unfixed placed logic.

-replace - (Optional) Remove all rectangles associated with the Pblock.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pblock - (Required) Specify the Pblock to be resized, moved, or removed.

Examples

The following example resizes the Pblock by adding a range of SLICES, and removing other SLICES, but keeps all instances placed at their current location:

```
resize_pblock block3 -add SLICE_X6Y67:SLICE_X11Y71 -remove SLICE_X6Y71:SLICE_X7Y71 \
-locs keep_all
```

The following example moves the specified Pblock by adding a range of SLICES, removing the existing range of SLICES, and trims any placed logic that falls outside the new Pblock. Then it adds a new range of SLICES and block ram to the specified Pblock in a second separate rectangle:

```
resize_pblock block3 -add SLICE_X3Y8:SLICE_X10Y3 -remove SLICE_X6Y67:SLICE_X11Y71 \  
-locs trim  
resize_pblock block3 -add {SLICE_X6Y67:SLICE_X11Y71 RAMB18_X0Y2:RAMB18_X1Y4}
```

See Also

- [add_cells_to_pblock](#)
- [create_pblock](#)
- [place_pblocks](#)

resize_pin_bus

Resize pin bus in the current design.

Syntax

```
resize_pin_bus [-from arg] [-to arg] [-quiet]  
[-verbose] pin_bus_name ...
```

Returns

Nothing

Usage

Name	Description
<code>[-from]</code>	New starting bus index
<code>[-to]</code>	New ending bus index
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>pin_bus_name</code>	Name of the pin bus to resize

Categories

[Netlist](#)

Description

Resize an existing bus pin, to grow the bus, shrink the bus, or renumber the current range of pin indexes. You can only do a single grow, shrink, or renumber operation with each command.

- You can grow the bus by indicating a new range of pin indexes outside the current range of indexes. Growing the bus leaves existing pins connected as they currently are.
- You can shrink the bus by indicating a new range of pin indexes inside the current range of indexes. Shrinking the bus, eliminates connections to removed bus pins, but leaves the remaining pins connected as they currently are.
- You can renumber the current bus indexes by providing a new range of pin indexes with the same width as the current range. Renumbering pins changes the pin index, but does not otherwise change connections.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

This command returns nothing if successful, and returns an error if it fails.

Arguments

-from *arg* - (Optional) The new starting index of the specified bus pin.

-to *arg* - (Optional) The new ending index of the specified bus pin.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

bus_pin_name - (Required) The name of the bus pin to modify. You must specify the pin names hierarchically from the cell instance the pin is assigned to. Pins created at the top-level of the design are ports, and should be resized with the **resize_port_bus** command.

Examples

The following example creates a blackbox cell, then creates a 24-bit bidirectional bus for the specified hierarchical cell, then resizes the bus pin to expand the width to 32-bits, then renumbers the index to include negative bus indexes:

```
create_cell -reference dmaBlock -black_box usbEngine0/myDMA
create_pin -direction INOUT -from 0 -to 23 usbEngine0/myDMA/dataBus
resize_pin_bus -from 0 -to 31 usbEngine0/myDMA/dataBus
resize_pin_bus -from -16 -to 15 usbEngine0/myDMA/dataBus
```

See Also

- [create_net](#)
- [create_pin](#)
- [create_port](#)
- [get_pins](#)
- [remove_pin](#)
- [resize_net_bus](#)
- [resize_port_bus](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

resize_port_bus

Resize port bus in the current design.

Syntax

```
resize_port_bus [-from arg] [-to arg] [-quiet]  
[-verbose] port_bus_name ...
```

Returns

Nothing

Usage

Name	Description
<code>[-from]</code>	New starting bus index
<code>[-to]</code>	New ending bus index
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>port_bus_name</i>	Name of the port bus to resize

Categories

[PinPlanning](#)

Description

Resize an existing bus port, to grow the bus, shrink the bus, or renumber the current range of port indexes. You can only do a single grow, shrink, or renumber operation with each command.

- You can grow the bus by indicating a new range of port indexes outside the current range of indexes. Growing the bus leaves existing port indexes connected as they currently are.
- You can shrink the bus by indicating a new range of port indexes inside the current range of indexes. Shrinking the bus, eliminates connections to removed bus ports, but leaves the remaining ports connected as they currently are.
- You can renumber the current bus indexes by providing a new range of port indexes with the same width as the current range. Renumbering ports changes the port index, but does not otherwise change connections.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write_*** command.

Note Netlist editing is not allowed on an RTL design.

This command returns nothing if successful, and returns an error if it fails.

Arguments

-from *arg* - (Optional) The new starting index of the specified bus port.

-to *arg* - (Optional) The new ending index of the specified bus port.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

bus_port_name - (Required) The name of the bus port to modify.

Examples

The following example creates a 32-bit output bus port, then rennumbers the ports to include negative bus indexes, then shrinks the bus width from 32-bits to 16-bits:

```
create_port -direction out -from 0 -to 31 outPorts
resize_port_bus -from -16 -to 15 outPorts
resize_port_bus -from -8 -to 7 outPorts
```

See Also

- [create_net](#)
- [create_pin](#)
- [create_port](#)
- [get_ports](#)
- [remove_port](#)
- [resize_net_bus](#)
- [resize_pin_bus](#)
- [write_checkpoint](#)
- [write_edif](#)
- [write_verilog](#)
- [write_vhdl](#)

restart

Rewind simulation to post loading state (as if design was reloaded), time is set to 0.

Syntax

```
restart [-quiet] [-verbose]
```

Returns

Restart retains breakpoints, Tcl forces, and settings in the waveform viewer but clears up the effects of all other Tcl commands

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Simulation](#)

route_design

Route the current design.

Syntax

```
route_design [-unroute] [-re_entrant arg] [-nets args] [-physical_nets]
[-pin arg] [-directive arg] [-no_timing_driven] [-preserve] [-delay]
[-free_resource_mode] -max_delay arg -min_delay arg [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-unroute]	Unroute whole design or the given nets/pins if used with -nets or -pin.
[-re_entrant]	Use "-re_entrant on" to remain in re_entrant mode. Default is to not enter re_entrant mode. Not applicable with -pin or -nets. They are inherently re_entrant commands. Default: off
[-nets]	Operate on the given nets.
[-physical_nets]	Operate on all physical nets.
[-pin]	Operate on the given pin.
[-directive]	Mode of behavior (directive) for this command. Please refer to Arguments section of this help for values for this option. Default: Default
[-no_timing_driven]	Do not run in timing driven mode.
[-preserve]	Preserve existing routing.
[-delay]	Use with -nets or -pin option to route in delay driven mode.
[-free_resource_mode]	Router will run in free resource mode
-max_delay	Use with -pin option to specify the max_delay constraint on the pin. When specified -delay is implicit.
-min_delay	Use with -pin option to specify the max_delay constraint on the pin. When specified -delay is implicit.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Tools

Description

Route the nets in the current design to complete logic connections on the target part.

Predefined routing strategies can be quickly selected using the **route_design -directive** command, or specific route options can be configured to define your own routing strategy.

Routing can be completed automatically with **route_design**, or can be completed iteratively using the various arguments of the **route_design** command. Iterative routing provides you some control over the routing process to route critical nets first and then route less critical nets, and to control the level of effort and the timing algorithms for these various route passes.

Routing is one step of the complete design implementation process, which can be run automatically through the use of the **launch_runs** command when running the Vivado tools in Project Mode.

In Non-Project Mode, the implementation process must be run manually with the individual commands: **opt_design**, **place_design**, **phys_opt_design**, **power_opt_design**, and **route_design**. Refer to the *Vivado Design Suite User Guide: Design Flows Overview (UG892)* for a complete description of Project Mode and Non-Project Mode.

Both placement and routing can be completed incrementally, based on prior results stored in a Design Checkpoint file (DCP), using the incremental compilation flow. Refer to the **read_checkpoint** command, or to *Vivado Design Suite User Guide: Implementation (UG904)* for more information on incremental place and route.

This command requires a placed design, and it is recommended that you have optimized the design with **opt_design** prior to placement.

Arguments

-unroute arg - (Optional) Unroute nets in the design. If no arguments are specified, all nets in the design are unrouted. The **route_design** command will not route any nets when the **-unroute** option is specified.

- Combine with the **-nets** option to limit unrouting to a list of nets.
- Combine with the **-pin** option to unroute from the pin to the nearest branch of the net.
- Combine with the **-physical_nets** option to unroute all logic 1 and logic 0 nets.

-re_entrant [on | off] - (Optional) Runs the router in re-entrant mode. The default setting is off. Running **route_design -re_entrant off**, or running the router in the default mode, directs the router to clean up its in-memory data structures when routing is complete. Running **route_design** with **-re_entrant on** preserves the in-memory routing data structures so that subsequent routing passes are faster. Run re-entrant routing when you anticipate successive routing operations, so that initialization is not repeated for each routing pass.

Note The **-re_entrant** option is not required with **-nets** or **-pin** because those options are inherently re-entrant.

-directive *arg* - (Optional) Direct routing to achieve specific design objectives. Only one directive can be specified for a single **route_design** command, and values are case-sensitive. Supported values are:

- **Explore** - Allows the Vivado router to explore different critical path routes after an initial route.
- **NoTimingRelaxation** - Prevents the router from relaxing timing to complete routing. If the router has difficulty meeting timing, it will run longer to try to meet the original timing constraints.
- **MoreGlobalIterations** - Uses detailed timing analysis throughout all stages instead of just the final stages, and will run more global iterations even when timing improves only slightly.
- **HigherDelayCost** - Adjusts the router's internal cost functions to emphasize delay over iterations, allowing a trade-off of runtime for better performance.
- **AdvancedSkewModeling** - Uses more accurate skew modeling throughout all routing stages which may improve design performance on higher-skew clock networks.
- **RuntimeOptimized** - Run fewest iterations, trade higher design performance for faster runtime.
- **Quick** - Absolute fastest runtime, non-timing-driven, performs the minimum required routing for a legal design.
- **Default** - Run **route_design** with default settings.

Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on the effects of each directive.

Note The **-directive** option controls the overall routing strategy, and is not compatible with any specific **route_design** options. It can only be used with **-quiet** and **-verbose**. In addition, the **-directive** option is ignored if the design is using the incremental compilation flow as defined by **read_checkpoint -incremental**

-physical_nets - (Optional) Route or unroute only logic zero and logic one nets.

-nets *args* - (Optional) Route or unroute only the specified net objects. Net objects must be specified using the **get_nets** command.

Note The router uses a quick route approach to find a routing solution for the specified nets, ignoring timing delays, when routing with **-nets** or **-pin** specified. Use **-delay** to find a route with the shortest delay.

-pin *arg* - (Optional) Route or unroute to the given pin which must be a cell input. If a pin is driven by a multiple fanout net, only the route segment between the net and pin are affected.

Note The router uses a quick route approach to find a routing solution for the specified nets, ignoring timing delays, when routing with **-nets** or **-pin** specified. Use **-delay** to find a route with the shortest delay.

-delay - (Optional) Can only be used in combination with the **-nets** or **-pin** options. By default nets are routed to achieve the fastest routing runtime, ignoring timing constraints, when using **-nets** and **-pin** options. The **-delay** option directs the router to try to achieve the shortest routed interconnect delay, but still ignores timing constraints.

-max_delay *arg* - (Optional) Can only be used with **-pin**. Directs the router to try to achieve a delay less than or equal to the specified delay given in picoseconds. When this options is specified, the **-delay** option is implied.

-min_delay *arg* - (Optional) Can only be used with **-pin**. Directs the router to try to achieve a delay greater than or equal to the specified delay given in picoseconds. When this option is specified, the **-delay** option is implied.

-no_timing_driven - (Optional) Disables the default timing driven routing algorithm. This results in faster routing results, but ignores any timing constraints during the routing process.

-preserve - (Optional) Existing routing will be preserved and not subject to the rip-up and reroute phase. This does not apply to routing that is fixed using the IS_ROUTE_FIXED or FIXED_ROUTE properties. Routing is preserved only for the current routing session.

-free_resource_mode - (Optional) This is an advanced option to control routing resources. With the Vivado router, by default, net routing may initially overlap using the same routing resources to speed routing solutions. These overlaps are later resolved as required to complete the connection. This option will prevent routes from initially overlapping, requiring new routes to use available routing resources only. This option can be used in later routing passes to prevent new connections from overlapping or disturbing completed routes.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

Route the entire design, and direct the router to try multiple algorithms for improving critical path delay:

```
route_design -directive Explore
```

The following example routes the specified set of timing critical nets to the shortest interconnect delay, and then preserves those routes while routing the rest of the design with low effort for fast routing: :

```
route_design -delay -nets $myCriticalNets
route_design -preserve -directive RuntimeOptimized
```

The following example turns on re-entrant routing, unroutes the current design, then re-routes it using the default settings:

```
route_design -re_entrant on
route_design -unroute
route_design
```

Route the specified nets using the fastest runtime:

```
route_design -nets [get_nets ctrl0/ctr*]
```

Route the specified nets to get the shortest interconnect delays:

```
route_design -nets [get_nets ctrl0/ctr*] -delay
```

Route to a particular pin:

```
route_design -pin [get_pins ctrl0/reset_reg/D]
```

Route to a particular pin, try to achieve less than 500 ps delay:

```
route_design -pin [get_pins ctrl0/reset_reg/D] -max_delay 500
```

Route to a particular pin, try to achieve more than 200 ps delay:

```
route_design -pin [get_pins ctrl0/ram0/ADDRARDADDR] -min_delay 200
```

See Also

- [get_nets](#)
- [get_pins](#)
- [launch_runs](#)
- [opt_design](#)
- [phys_opt_design](#)
- [place_design](#)
- [power_opt_design](#)
- [read_checkpoint](#)
- [write_checkpoint](#)

run

Run the simulation for the specified time.

Syntax

```
run [-all] [-quiet] [-verbose] [time] [unit]
```

Returns

Nothing

Usage

Name	Description
[-all]	Runs simulation till a breakpoint, an exception or no events left in the queue
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[time]</i>	Length of simulation time
<i>[unit]</i>	Unit for time from the following time units: fs, ps, ns, us, ms, sec

Categories

[Simulation](#)

run_hw_ila

Arm hardware ILAs.

Syntax

```
run_hw_ila [-trigger_now arg] [-quiet] [-verbose] [hw_ilas...]
```

Returns

Nothing

Usage

Name	Description
[-trigger_now]	Trigger and capture immediately.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[hw_ilas]	hardware ILAs Default: Current hardware ILA

Categories

[Hardware](#)

run_hw_sio_scan

Run hardware SIO scans.

Syntax

```
run_hw_sio_scan [-quiet] [-verbose] hw_sio_scans
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_sio_scans</i>	hardware SIO scans

Categories

[Hardware](#)

save_bd_design

Save an existing IP subsystem design to disk file.

Syntax

```
save_bd_design [-quiet] [-verbose] [name]
```

Returns

The design object. Returns nothing if the command fails

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>name</i>]	Name of design to save.

Categories

[IPIntegrator](#)

Description

Saves any changes to the current or specified IP subsystem design in the IP Integrator feature of the Vivado Design Suite.

This command returns the name of the file written.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - Specify the name of the IP subsystem design to save. If name is not specified, the current IP subsystem design will be saved.

Examples

The following example saves the current IP subsystem design in the current project:

```
save_bd_design
```

See Also

- [close_bd_design](#)
- [create_bd_design](#)
- [current_bd_design](#)
- [get_bd_designs](#)
- [open_bd_design](#)

save_constraints

Save the current design's constraints.

Syntax

```
save_constraints [-force] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-force]	Force constraints save, overwriting the target and source XDC if necessary
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Project](#)

Description

Saves any changes to the constraints files of the active constraints set. This command writes any changes to the constraints files to the project data on the hard drive; saving any work in progress and committing any changes.

Arguments

-force - (Optional) Save the active constraints files regardless of whether any changes have been made, overwriting the current target constraints file.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example saves the constraints files for the active constraints set regardless of any changes to the files:

```
save_constraints -force
```

See Also

[save_constraints_as](#)

save_constraints_as

Save current design's constraints as a new set of constraints files.

Syntax

```
save_constraints_as [-dir arg] [-target_constrs_file arg] [-quiet]  
[-verbose] name
```

Returns

Nothing

Usage

Name	Description
<code>[-dir]</code>	Directory to save constraints to
<code>[-target_constrs_file]</code>	Target constraints file for the new fileset
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>name</code>	Name of the new constraints fileset

Categories

[Project](#)

Description

Copies the active constraints set to create a new constraints set, with local copies of any constraints files that are part of the constraints set. You can also specify a new constraints file to use as the target for the copied constraints set.

Use this command to save changes to the constraints in a design without affecting the current constraints files. This allows you to do some "what-if" type development of design constraints.

Note The new constraint set created by the **save_constraints_as** command will not be active in the design, although it will be referenced by the design. To make the constraints set active you must set the `constrset` property to point to the new constraints set for specific runs. See the example below.

Arguments

-dir arg - (Optional) The directory into which constraints files are saved. If the directory is not specified, the new constraints set is located in the project sources directory. The constraints files from the active constraints set are copied into the specified directory.

-target_constrs_file *arg* - (Optional) Specifies a new target constraints file for the new constraints fileset. If a path is not specified as part of the file name, the file will be created in the fileset directory.

Note You must specify the **.xdc** file extension, or the command will report a warning that the filetype is invalid, and cannot be set to the target constraint set. In this case, the existing target constraints file will be used as the target

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the constraints set to write.

Examples

The following example saves the active constraints set into a new constraints set called `constrs_2`, and copies any constraints files into the specified directory, as well as creating a new target constraints file for the constraints set:

```
save_constraints_as -dir C:/Data/con1 -target_constrs_file rev1.xdc constrs_2
```

The following example saves the active constraints set as a new constraints set called `newCon2`, and copies any constraint files into the `newCon2` constraint directory under project sources. The `constrset` property for the specified synthesis and implementation runs are then set to point to the new constraints set:

```
save_constraints_as newCon2
set_property CONSTRSET newCon2 [get_runs synth_1]
set_property CONSTRSET newCon2 [get_runs impl_1]
```

Note The constraints set is not active in the design until it has been set to active for the current runs.

See Also

[save_constraints](#)

save_project_as

Save the current project under a new name.

Syntax

```
save_project_as [-force] [-quiet] [-verbose] name [dir]
```

Returns

Saved project object

Usage

Name	Description
[-force]	Overwrite existing project directory
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	New name for the project to save
<i>[dir]</i>	Directory where the project file is saved Default: .

Categories

[Project](#)

Description

Saves a currently open project file under a new name in the specified directory, or in the current working directory if no other directory is specified.

Arguments

-force - (Optional) Overwrite the existing project. If the project name is already define in the specified directory then you must also specify the **-force** option for the tool to overwrite the existing project.

Note If the existing project is currently open, the new project will overwrite the existing project on the disk, but both projects will be opened in the tool. In this case you should probably run the **close_project** command prior to running **create_project**.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) The name of the new project. This argument must appear before the specified directory. Since these commands do not have parameters, the tool interprets the first argument as *name* and uses the second argument as *dir*. The project file is saved as *name.ppr* and is written into the specified directory *dir*.

dir - (Optional) The directory name in which to write the new project file. If the specified directory does not exist a new directory will be created. If the directory is specified with the complete path, the tool uses the specified path name. However, if *dir* is specified without a path, the tool looks for or creates the directory in the current working directory, or the directory from which the tool was launched.

Note When creating a project in GUI-mode, the tool appends the filename *name* to the directory name *dir* and creates a project directory with the name *dir/name* and places the new project file and project data folder into that project directory.

Examples

The following example saves the active project as a new project called myProject in a directory called myProjectDir:

```
save_project_as myProject myProjectDir
```

Note Because *dir* is specified as the folder name only, the tool will create the project in the current working directory, or the directory from which the tool was launched.

The following example saves the current project to a new project called myProject in a directory called C:/Designs/myProjectDir. If you use the **-force** argument, the tool will overwrite an existing project if one is found in the specified location.

```
save_project_as myProject C:/Designs/myProjectDir -force
```

See Also

- [create_project](#)
- [open_project](#)

save_wave_config

Saves the specified or current wave configuration object to the given filename.

Syntax

```
save_wave_config [-object args] [-quiet] [-verbose] [filename]
```

Returns

The wave configuration object saved

Usage

Name	Description
[-object]	The WCFG or wave configuration to save. Default: Current wave configuration
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[filename]</i>	Filename to save the specified or current wave configuration object

Categories

[Waveform](#)

select_objects

Select objects in GUI.

Syntax

```
select_objects [-add] [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
<code>[-add]</code>	Add to existing selection list
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>objects</code>	Objects to select

Categories

[GUIControl](#)

Description

Selects the specified object in the appropriate open views in the GUI mode. This command is for display purposes only. You must use the **get_selected_objects** command to return the selected objects for use in other commands.

The **select_objects** command may select secondary objects in addition to the primary object specified. The selection of secondary objects is controlled through the use of Selection Rules defined in the **Tools > Options** command. Refer to the *Vivado Design Suite User Guide: Using the IDE (UG893)* for more information on Setting Selection Rules.

Selected objects can be unselected with the **unselect_objects** command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) Specifies one or more objects to be selected.

Examples

The following example selects the specified site on the device:

```
select_objects [get_sites SLICE_X56Y214]
```

See Also

- [get_selected_objects](#)
- [unselect_objects](#)

set_case_analysis

Specify that an input is 1, 0, rising or falling.

Syntax

```
set_case_analysis [-quiet] [-verbose] value objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>value</i>	Logic value on the pin: Values: 0, 1, rising, falling, zero, one, rise, fall
<i>objects</i>	List of ports or pins

Categories

[SDC](#), [XDC](#)

Description

Specifies that an input is 1, 0, rising or falling. This command is usually used to force values onto the ports and do the analysis. This is ideally suited for driving the select line of a BUFGMUX. Driving the value on the select line of BUFGMUX, will select one of the TIMESPECs for timing analysis. This prevents the analysis of unwanted TIMESPECs.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

value - (Required) The value to use on the port or pin for timing analysis. The valid values are 0 or zero, 1 or one, rise or rising, fall or falling. The default setting is 1.

objects - (Required) One or more ports or pins on which to apply the *value*.

Examples

The following example is for a design with two clocks which are multiplexed using a BUFGMUX. Both the clocks are running at different frequencies. The default analysis of this design would be done for the clock that is defined later in the XDC. In this case, it would be CLK_B. Using **set_case_analysis** the analysis would be different. When (SEL = 0), the analysis would be based on CLK_A. When (SEL = 1), the analysis would be based on CLK_B.

```
create_clock -period 10.0 [get_ports CLK_A]
create_clock -period 15.0 [get_ports CLK_B]
set_case_analysis 0 [get_ports SEL]
```

See Also

[report_timing](#)

set_clock_groups

Set exclusive or asynchronous clock groups.

Syntax

```
set_clock_groups [-name arg] [-logically_exclusive]
[-physically_exclusive] [-asynchronous] [-group args] [-quiet]
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[-name]	Name for clock grouping
[-logically_exclusive]	Specify logically exclusive clock groups
[-physically_exclusive]	Specify physically exclusive clock groups
[-asynchronous]	Specify asynchronous clock groups
[-group]	Clocks List
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

Define clocks, or groups of clocks, that are exclusive with or asynchronous to other clocks in the design. Exclusive or asynchronous clocks are not active at the same time, and paths between them can be ignored during timing analysis.

Using this command is similar to defining false path constraints for data paths moving between exclusive or asynchronous clock groups.

This command can also be used for multiple clocks that are derived from a single BUFGMUX as both of the clocks will not be active at the same time.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-name *<group_name>* - (Optional) Name of the clock group to be created. A name will be automatically assigned if one is not specified.

-logically_exclusive - (Optional) The specified clocks are logically exclusive.

-physically_exclusive - (Optional) The specified clocks are physically exclusive, and cannot exist in the design at the same time.

-asynchronous - (Optional) The specified clocks are asynchronous to one another.

Note **-logically_exclusive**, **-physically_exclusive** and **-asynchronous** are mutually exclusive arguments.

-group *<args>* - (Optional) The list of clocks to be included in the clock group. Each group of clocks is exclusive with or asynchronous with the clocks specified in all other groups. If only one group of clocks is specified, that group is exclusive with or asynchronous to all other clocks in the design.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

Group all the elements driven by src_clk and sync_clk. Both the clock groups are asynchronous.

```
set_clock_groups -group [get_clocks src_clk] -group [get_clocks sync_clk] \  
-asynchronous
```

See Also

[set_false_path](#)

set_clock_latency

Capture actual or predicted clock latency.

Syntax

```
set_clock_latency [-clock args] [-rise] [-fall] [-min] [-max] [-source]
[-late] [-early] [-quiet] [-verbose] latency objects
```

Returns

Nothing

Usage

Name	Description
[-clock]	List of relative clocks
[-rise]	Specify clock rise latency
[-fall]	Specify clock fall latency
[-min]	Specify clock rise and fall min condition latency
[-max]	Specify clock rise and fall max condition latency
[-source]	Specify clock rise and fall source latency
[-late]	Specify clock rise and fall late source latency
[-early]	Specify clock rise and fall early source latency
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>latency</i>	Latency value
<i>objects</i>	List of clocks, ports or pins

Categories

[SDC](#), [XDC](#)

Description

This command defines a clock's source or network latency for specified clocks, ports, or pins.

Note This command operates silently and does not return direct feedback of its operation.

Source latency is the time a clock signal takes to propagate from its waveform origin to the clock definition point in the design. For example, this would be the time delay for the clock to propagate from its source (oscillator) on the system board to the FPGA input port.

Network latency is the time a clock signal takes to propagate from its definition point in the design to a register clock pin on the timing path. The total clock latency at a register clock pin is the sum of a clock's source latency and network latency.

Arguments

-clock *args* - (Optional) Specifies a list of clocks associated with the *latency* assigned to the specified *objects*. If the **-clock** argument is not used, the clock *latency* will be applied to all clocks passing through the specified pins and ports.

-rise - (Optional) Defines the latency for the rising clock edge.

-fall - (Optional) Defines the latency for the falling clock edge.

-min - (Optional) Defines the minimum latency for the specified clocks for multi-corner analysis.

-max - (Optional) Defines the maximum latency for the specified clocks for multi-corner analysis.

Note The **-min** and **-max** options are mutually exclusive

-source - (Optional) Defines the specified *latency* as a source latency. Clock source latencies can only be specified for clock objects and clock source pins.

Note Without the **-source** argument the *latency* is considered as network latency

-late - (Optional) The time delay specified by **-latency** is how late the clock edge arrives.

-early - (Optional) The time delay specified by **-latency** is how early the clock edge arrives.

Note The **-early** and **-late** options are mutually exclusive, and can only be specified when **-source** is also specified

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

latency - (Optional) The amount of clock latency to apply.

objects - (Optional) The clock, port, or pin objects on which to apply the latency. Specifying pin or port objects assigns the latency to all register clock pins in the transitive fanout of the pins or ports. If **-clock** is used, the latency is applied to all register clock pins of the specified clocks.

Note If *objects* specifies a clock, the **-clock** argument is unnecessary, and is ignored.

Examples

This example will set an early latency on the rising edge of CLK_A.

```
set_clock_latency -source -rise -early 0.4 [get_ports CLK_A]
```

See Also

[report_timing](#)

set_clock_sense

Set clock sense on ports or pins.

Syntax

```
set_clock_sense [-positive] [-negative] [-stop_propagation]
[-pulse arg] [-clocks args] [-quiet] [-verbose] pins
```

Returns

Nothing

Usage

Name	Description
[-positive]	Specify positive unate (non_inverting) clock sense
[-negative]	Specify negative unate (inverting) clock sense
[-stop_propagation]	Stop clock propagation from specified pins
[-pulse]	Specify pulse clock sense
[-clocks]	List of clocks
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>pins</i>	List of port and/or pins

Categories

[SDC](#), [XDC](#)

Description

Sets clock sense at specified ports or pins. This is used to define the positive or negative unateness at the pin relative to a clock object. However, the specified unateness only applies at a non-unate point in the clock network, at a point where the clock signal cannot be determined. Since the clock signal is not determined, the defined clock sense propagates forward from the given pins.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-positive - (Optional) The unate clock sense is positive (non_inverting).

-negative - (Optional) The unate clock sense is negative (inverting).

-stop_propagation - (Optional) Stop the propagation of clocks in the **-clocks** argument from the specified pins or ports. Propagation of the clock as clock and data is stopped.

-pulse *arg* - (Optional) The pulse clock sense.

Note **-positive**, **-negative**, **-stop_propagation** and **-pulse** are mutually exclusive.

-clocks *args* - (Optional) A list of clocks on which to apply the clock sense for the specified pins and ports . If the **-clocks** argument is not used, the clock sense will be applied to all clocks passing through the specified pins and ports.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

pins - (Required) List of ports and pins to propagate the clock sense to.

Examples

The following example specifies that only the positive unate paths will propagate through the output pin of the XOR gate as compared with the original clock.

```
set_clock_sense -positive [get_pins xor_a.z]
```

See Also

- [create_clock](#)
- [get_pins](#)

set_clock_uncertainty

Set clock uncertainty.

Syntax

```
set_clock_uncertainty [-setup] [-hold] [-from args] [-rise_from args]
[-fall_from args] [-to args] [-rise_to args] [-fall_to args] [-quiet]
[-verbose] uncertainty [objects]
```

Returns

Nothing

Usage

Name	Description
[-setup]	Specify clock uncertainty for setup checks
[-hold]	Specify clock uncertainty for hold checks
[-from]	Specify inter-clock uncertainty source clock
[-rise_from]	Specify inter-clock uncertainty source clock with rising edge
[-fall_from]	Specify inter-clock uncertainty source clock with falling edge
[-to]	Specify inter-clock uncertainty destination clock
[-rise_to]	Specify inter-clock uncertainty destination clock with rising edge
[-fall_to]	Specify inter-clock uncertainty destination clock with falling edge
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>uncertainty</i>	Uncertainty of clock network
<i>[objects]</i>	List of clocks, ports or pins

Categories

[SDC](#), [XDC](#)

Description

This command is used to define the uncertainty of a clock in the design. Clock uncertainty is the maximum difference between the arrival of clock signals at registers within one clock domain or between clock domains. This is also known as skew. The clock uncertainty is used while doing setup and hold checks.

Clocks can be created using the **create_clock** or the **create_generated_clock** commands, or can be automatically generated by the tool, at the output of MMCM for instance.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

- setup** - (Optional) Specify the clock uncertainty for the setup checks
- hold** - (Optional) Specify the clock uncertainty for the hold checks
- from** <clock_source_name> - (Optional) Specify inter-clock uncertainty source clock
- rise_from** <clock_source_name> - (Optional) Specify inter-clock uncertainty source clock with rising edge
- fall_from** <clock_source_name> - (Optional) Specify inter-clock uncertainty source clock with falling edge
- to** <clock_destination_name> - (Optional) Specify inter-clock uncertainty destination clock
- rise_to** <destination_clock_name> - (Optional) Specify inter-clock uncertainty destination clock with rising edge
- fall_to** <destination_clock_name> - (Optional) Specify inter-clock uncertainty destination clock with falling edge
- quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.
- verbose** - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

<**uncertainty**> - (Required) Uncertainty of the clock network

<**objects**> - (Optional) List of clocks, cells, ports, or pins to apply the uncertainty to

Examples

The following examples define clock uncertainty for clock wbClk and between different clock domains.

The following example defines the uncertainty between all clock domains:

```
set_clock_uncertainty 0.225 -from [get_clocks] -to [get_clocks]
```

The following command defines setup and hold uncertainty within the wbClk clock domain:

```
set_clock_uncertainty -setup 0.213 [get_clocks wbClk]
set_clock_uncertainty -hold 0.167 [get_clocks wbClk]
```

See Also

- [create_clock](#)
- [create_generated_clock](#)
- [get_clocks](#)

set_data_check

Create data to data checks.

Syntax

```
set_data_check [-from args] [-to args] [-rise_from args]
[-fall_from args] [-rise_to args] [-fall_to args] [-setup] [-hold]
[-clock args] [-quiet] [-verbose] value
```

Returns

Nothing

Usage

Name	Description
[-from]	From pin/port of data to data check
[-to]	To pin/port of the data to data check
[-rise_from]	Rise from pin/port of data to data check
[-fall_from]	Fall from pin/port of data to data check
[-rise_to]	Rise to pin/port of data to data check
[-fall_to]	Fall to pin/port of data to data check
[-setup]	Specify data check setup time
[-hold]	Specify data check hold time
[-clock]	Specify the clock domain at related pin/port of the checks
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>value</i>	Setup or hold time of the defined checks

Categories

[SDC](#), [XDC](#)

Description

Performs a setup and hold check for a data pin with respect to another data pin. This is different from a conventional setup and hold check that is done with respect to a clock pin. Setup and hold checks are referenced from the related pin, specified by **-from**, to the constrained pin, specified by **-to**. The related pin is similar to the clock pin in a conventional setup and hold check.

Note This command operates silently and does not return direct feedback of its operation

Arguments

- from** *value* - (Optional) From pin/port of data to data check. The **-from** argument specifies the related pin.
- to** *value* - (Optional) To pin/port of the data to data check. The **-to** argument specifies the constrained pin
- rise_from** *value* - (Optional) Rise from pin/port of data to data check.
- fall_from** *value* - (Optional) Fall from pin/port of data to data check.
- rise_to** *value* - (Optional) Rise to pin/port of data to data check.
- fall_to** *value* - (Optional) Fall to pin/port of data to data check.
- setup** *value* - (Optional) Perform the setup data check.
- hold** *value* - (Optional) Perform the hold data check.
- clock** *value* - (Optional) Specify the clock domain at the related pin or port of the checks.
- quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.
- verbose** - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

value - (Required) The setup or hold time for the defined data checks.

Examples

The following example defines a data check for a setup violation from pin A_IN to pin C_IN:

```
set_data_check -from A_IN -to C_IN -setup 2.0
```

In the above example, A_IN is the related pin and C_IN is the constrained pin. The above constraint would do a setup check of C_IN with respect to A_IN. The data at C_IN should arrive 2.0 ns prior to the edge of A_IN.

See Also

[report_timing](#)

set_default_switching_activity

Set default switching activity on specified types.

Syntax

```
set_default_switching_activity [-toggle_rate arg]
[-static_probability arg] [-quiet] [-verbose] type...
```

Returns

Nothing

Usage

Name	Description
[-toggle_rate]	Toggle rate value: 0% <= Value <= 200%. The value is % of clock. Default: 0.0
[-static_probability]	Static probability value: 0 <= Value <= 1 Default: 0.0
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>type</i>	Sets the default seed values on specified types for vector-less propagation engine. List of valid default type values: input, input_set, input_reset, input_enable, register, dsp, bram_read_enable, bram_write_enable, output_enable, clock, all

Categories

[XDC](#), [Power](#)

Description

Sets a default activity rate for a broad class of signals when performing power estimation.

Note This command operates silently and does not return direct feedback of its operation.

The switching activity of a design affects both the static and dynamic power consumption. The static power is often dependent on logic state transitions, and the dynamic power is directly proportional to the toggle rate or switching activity.

The current default switching activity attributes can be found by using the **report_default_switching_activity** command. The values can be set to their default values by using the **reset_default_switching_activity** command.

Use the **set_switching_activity** command to define the activity of one or more signals, rather than the whole class.

Arguments

-toggle_rate *rate* - (Optional) The toggle rate describes how often the output switches relative to the controlling clock. Valid values are between 0 and 200%. An output that switches once per clock cycle toggles at 100%. The toggle rate is directly used for power calculation for the specified type of signals.

-static_probability *value* - (Optional) The static probability or percentage of the clock period that the output is at a logic value of '1'. Valid values are $0 < \text{value} < 1$. A value of 0 means the output is always 0 and a value of 1 means that the output is always 1. A clock port with a 50% duty cycle would have for example a value of 0.5. The static probability is used to calculate the propagation of the known switching activity (toggle rate) through all the nodes of the design and is therefore essential to perform power calculation.

Note One or both of **-static_probability** or **-toggle_rate** must be specified with the **set_switching_activity** command.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

type - (Required) Specify the class of signals to apply the defined switching activity to. Valid values are: input, input_set, input_reset, input_enable, register, dsp, bram_read_enable, bram_write_enable, output_enable, clock, all.

Examples

The following example specifies a toggle rate of 85% for all DSP blocks:

```
set_default_switching_activity -toggle_rate 85 dsp
```

The following example specifies the toggle rate and switching probability for all supported types:

```
set_default_switching_activity -toggle_rate 19 -static_probability .22 all
```

See Also

- [power_opt_design](#)
- [report_default_switching_activity](#)
- [report_power](#)
- [report_switching_activity](#)
- [reset_default_switching_activity](#)
- [reset_switching_activity](#)
- [set_switching_activity](#)

set_delay_model

Timing Delay Model.

Syntax

```
set_delay_model [-interconnect arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-interconnect]</code>	Interconnect delay model used for timing analysis: Values: estimated, actual(default), none
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

Timing

Description

Sets the interconnect delay model for timing analysis. There are three settings for the interconnect delay model: "actual", "estimated", or "none".

- If "actual" is selected, the actual delay from the routed interconnect will be used in timing analysis. If the design is only partially routed, then the actual delay from the routed portion will be used, along with estimated delay for the unrouted portion. The timing report will provide details regarding the source of the calculated delay.
- If "estimated" delays are selected, the timing analysis will include an estimate of the interconnect delays based on the placement and connectivity of the design onto the device prior to implementation. Estimated delay can be specified even if the design is fully routed.
- If "none" is selected, then no interconnect delay is included in the timing analysis, and only the logic delay is applied.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-interconnect [actual | estimated | none] - (Optional) Delay model to be used. The default setting is **actual**.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following command will use a timing delay model which is an estimated value.

```
set_delay_model -interconnect estimated
```

See Also

[report_timing](#)

set_disable_timing

Disable timing arcs.

Syntax

```
set_disable_timing [-from arg] [-to arg] [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-from]	From pin on cell
[-to]	To pin on cell
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	List of cells or pins, ports, lib-cells, lib-pins, libcell/cell timing-arcs

Categories

[SDC](#), [XDC](#), [Timing](#)

Description

Disables timing arcs within the specified cell that lead to the output pins of the cell. Only the I/O paths between the clock port and the outputs of the cell are disabled.

The purpose of disabling a timing arc is to prevent timing analysis through the arc.

Note This command operates silently and does not return direct feedback of its operation

Arguments

-from *pin_name* - (Optional) Specifies the source pin of an object cell.

-to *pin_name* - (Optional) Specifies the destination pin of an object cell.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) A list of objects on which to disable the timing arcs. Can be any of the following types: cells, pins, lib-cells, lib-pins, lib-cell/cell timing arcs.

Examples

The following example disable the timing check between AX to AMUX pin of cell abc:

```
set_disable_timing -from AX -to AMUX abc
```

The following example disables the timing arcs between the specified input pin to the specified output pin of a BRAM cell:

```
set_disable_timing -from WEBWE[3] -to CLKMEM \  
  [get_cells ldpc_dout360_channel/U_AP_FIFO_ldpc_dout360_channel_ram/mem_reg_0]
```

The following example disables all timing arcs of the specified cell:

```
set arcs [get_timing_arcs -of_objects \  
  [get_cells ldpc_dout360_channel/U_AP_FIFO_ldpc_dout360_channel_ram/mem_reg_0]]  
set_disable_timing $arcs
```

See Also

- [get_cells](#)
- [get_timing_arcs](#)
- [report_timing](#)

set_external_delay

Set external delay.

Syntax

```
set_external_delay -from arg -to arg [-min] [-max] [-quiet]  
[-verbose] delay_value
```

Returns

Nothing

Usage

Name	Description
-from	Output port
-to	Input port
[-min]	Specifies minimum delay
[-max]	Specifies maximum delay
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>delay_value</i>	External (feedback) delay value

Categories

[XDC](#)

Description

Sets the external (feedback) delay between an output and input port. The external delay is used in the calculation of the PLL/MMCM compensation delay for PLLs/MMCMs with external feedback.

A min or max value can be specified. By default the value specified applies to both min (hold) and max (setup) compensation delays.

The command returns the defined delay.

Arguments

-from *arg* - (Required) The output port name.

-to *arg* - (Required) The input port name.

-min - (Optional) Specifies the *delay_value* is a minimum delay value for hold time analysis.

-max - (Optional) Specifies the *delay_value* is a maximum delay value for setup analysis.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

delay_value - (Required) The external delay value. The default value is 0.

Examples

The following example sets the external feedback delay to 1.0 ns between the port ClkOut and ClkFb:

```
set_external_delay -from [get_ports ClkOut] -to [get_ports ClkFb] 1.0
```

See Also

- [report_timing](#)
- [set_input_delay](#)
- [set_output_delay](#)

set_false_path

Define false path.

Syntax

```
set_false_path [-setup] [-hold] [-rise] [-fall] [-reset_path]
[-from args] [-rise_from args] [-fall_from args] [-to args]
[-rise_to args] [-fall_to args] [-through args] [-rise_through args]
[-fall_through args] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-setup]	Eliminate setup timing analysis for paths
[-hold]	Eliminate hold timing analysis for paths
[-rise]	Eliminate only rising delays for the defined paths
[-fall]	Eliminate only falling delays for the defined paths
[-reset_path]	Reset this path before setting false path
[-from]	List of path startpoints or clocks
[-rise_from]	Apply to paths rising from the list of startpoints or clocks
[-fall_from]	Apply to paths falling from the list of startpoints or clocks
[-to]	List of path endpoints or clocks
[-rise_to]	Apply to paths with rise transition at the list of endpoints or clocks
[-fall_to]	Apply to paths with fall transition at the list of endpoints or clocks
[-through]	List of through pins, cells or nets
[-rise_through]	Apply to paths rising through pins, cells or nets
[-fall_through]	Apply to paths falling through pins, cells or nets
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

Sets false timing paths in the design that are ignored during timing analysis.

Note This command operates silently and does not return direct feedback of its operation

Arguments

- setup** - (Optional) Eliminate setup timing analysis for specified timing paths.
- hold** - (Optional) Eliminate hold timing analysis for specified timing paths.
- rise** - (Optional) Eliminate rising delays for the specified timing paths.
- fall** - (Optional) Eliminate falling delays for the specified timing paths.
- reset_path** - (Optional) Reset the timing path before setting false path. This clears all exception-based timing constraints from the defined timing path.
- from** <element_name> - (Optional) List of path origins or clocks
- rise_from** <element_name> - (Optional) Apply to paths rising from the list of origins or clocks
- fall_from** <element_name> - (Optional) Apply to paths falling from the list of origins or clocks
- to** <element_name> - (Optional) List of path endpoints or clocks
- rise_to** <element_name> - (Optional) Apply to paths with rise transition at the list of endpoints or clocks
- fall_to** <element_name> - (Optional) Apply to paths with fall transition at the list of endpoints or clocks
- through** <element_name> - (Optional) List of through pins, cells or nets
- rise_through** <element_name> - (Optional) Apply to paths rising through pins, cells or nets
- fall_through** <element_name> - (Optional) Apply to paths falling through pins, cells or nets
- quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.
- verbose** - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example eliminates the setup timing for paths from the bftClk:

```
set_false_path -setup -from bftClk
```

The following example excludes paths between the two clocks from timing analysis:

```
set_false_path -from [get_clocks GT0_RXUSRCLK2_OUT] -to [get_clocks DRPCLK_OUT]
```

See Also

- [get_clocks](#)
- [get_pins](#)
- [get_ports](#)
- [report_timing](#)

set_hierarchy_separator

Set hierarchical separator character.

Syntax

```
set_hierarchy_separator [-quiet] [-verbose] [separator]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[separator]	Hierarchy separator character Default: /

Categories

[SDC](#), [XDC](#)

Description

Sets the character that will be used for separating levels of hierarchy in the design.

Note This command operates silently and does not return direct feedback of its operation

Arguments

separator - (Optional) The new character to use as a hierarchy separator. Valid characters to use as the hierarchy separator are: '/', '@', '^', '#', '.', and '|'. The default character is '/', and is used when no *separator* is specified.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

This example changes the hierarchy separator to the '|' character:

```
set_hierarchy_separator |
```

The following example restores the default hierarchy separator, '/':

```
set_hierarchy_separator
```

See Also

[get_hierarchy_separator](#)

set_input_delay

Set input delay on ports.

Syntax

```
set_input_delay [-clock args] [-reference_pin args] [-clock_fall]
[-rise] [-fall] [-max] [-min] [-add_delay] [-network_latency_included]
[-source_latency_included] [-quiet] [-verbose] delay objects
```

Returns

Nothing

Usage

Name	Description
<code>[-clock]</code>	Relative clock
<code>[-reference_pin]</code>	Relative pin or port
<code>[-clock_fall]</code>	Delay is relative to falling edge of clock
<code>[-rise]</code>	Specifies rising delay
<code>[-fall]</code>	Specifies falling delay
<code>[-max]</code>	Specifies maximum delay
<code>[-min]</code>	Specifies minimum delay
<code>[-add_delay]</code>	Don't remove existing input delay
<code>[-network_latency_included]</code>	Specifies network latency of clock already included
<code>[-source_latency_included]</code>	Specifies source latency of clock already included
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>delay</i>	Delay value
<i>objects</i>	List of ports

Categories

[SDC](#), [XDC](#)

Description

Sets the input delay on ports.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-clock *arg* - (Optional) Indicates that the input delay is relative to the specified clock. By default the rising edge is used. However the **-clock_fall** argument can be used to indicate that the falling edge should be used instead.

-reference_pin *arg* - (Optional) Specifies that the delay is relative to the active edge of a clock appearing on the specified pin or port rather than a clock.

-clock_fall - (Optional) Specifies that the delay is relative to a falling edge of the clock rather than rising edge.

-rise - (Optional) Specifies the input delay applies to rising transitions on the specified ports. The default is to apply the delay for both rising and falling transitions.

-fall - (Optional) Specifies the input delay applied to falling transitions on the specified ports. The default is to apply the delay for both rising and falling transitions.

-max - (Optional) Indicates the input delay specified is only used when calculating the maximum (longest) path delays.

-min - (Optional) Indicates the input delay specified is only used when calculating the minimum (shortest) path delays.

-add_delay - (Optional) Add the specified delay to any existing delay on the port. The default behavior is to replace the existing delays.

-network_latency_included - (Optional) Indicates that the network latency of the reference clock is included in the delay value.

-source_latency_included - (Optional) Indicates that the source latency of the relative clock is included in the specified delay.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

delay - (Required) The input delay to apply to the specified ports.

objects - (Required) The list of ports to which the delay value will be assigned.

Examples

The following example specifies the input delay on port DIN. The input delay is 3 and is relative to the rising edge of clock clk1:

```
set_input_delay -clock clk1 3 DIN
```

The following example specifies the input delay on port DIN. The input delay is 2 and is relative to the falling edge of the clock clk1:

```
set_input_delay -clock_fall -clock clk1 2 DIN
```

The following example specifies the input delay on port reset. The input delay is 2 and is relative to the rising edge of the clock that appears on the pin wbClk_IBUF_BUFG_inst/O, originating from the clock wbClk:

```
set_input_delay -clock wbClk 2 -reference_pin [get_pin wbClk_IBUF_BUFG_inst/O] reset
```

The following example specifies the input delay on all non clock input ports of the design. Although all_inputs returns all ports of the design, including clock ports, set_input_delay will skip setting input delays on the clock ports. The input delay is 1 relative to the rising edge of the clock wbClk:

```
set_input_delay -clock wbClk 1 [all_inputs]
```

The following example sets an input delay of 4 relative to the rising edge of the clock wbClk on the ports reset and wbDataForInput:

```
set_input_delay -clock wbClk 4 [list reset wbDataForInput]
```

See Also

- [all_clocks](#)
- [all_inputs](#)
- [check_timing](#)
- [create_clock](#)
- [get_ports](#)
- [report_timing](#)
- [set_output_delay](#)

set_input_jitter

Set input jitter for a clock object.

Syntax

```
set_input_jitter [-quiet] [-verbose] clock input_jitter
```

Returns

Clock

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>clock</code>	Clock
<code>input_jitter</code>	Input jitter: Value >= 0

Categories

[XDC](#)

Description

Input jitter is the difference between successive clock edges due to variation from the ideal arrival times. This command sets the input jitter for a specified primary clock, defined with the **create_clock** command. Because the command accepts a single clock, the jitter for each primary clock must be set individually.

You can only use the **set_input_jitter** command to specify input jitter on primary clocks. You cannot use the command to set input jitter on generated or auto generated clocks. Input jitter is propagated to generated and auto derived clocks from the master clock.

The **set_input_jitter** command is not supported for elaborated designs or for synthesis.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

clock_name - (Required) The clock name of a primary clock, defined with the **create_clock** command.

input_jitter - (Required) The input jitter for the specified clock object (value ≥ 0).

Examples

The following example sets an input jitter value of 0.3 on two clocks, sysClk and procClk. Although the jitter values are the same, you must use two **set_input_jitter** commands since the command only takes one clock as an argument:

```
set_input_jitter sysClk 0.3
set_input_jitter procClk 0.3
```

The following example defines a primary clock, sysClk, and a generated clock, sysClkDiv2, that is a divide by two version of the primary clock. An input jitter of 0.15 is specified on the primary clock. The input jitter is automatically propagated to the generated clock:

```
create_clock -period 10 -name sysClk [get_ports sysClk]
create_generated_clock -name sysClkDiv2 -source [get_ports sysClk] -divide_by 2 \ [get_pins clkgen/sysClkDiv2]
set_input_jitter sysClk 0.15
```

See Also

- [all_clocks](#)
- [check_timing](#)
- [create_clock](#)
- [create_generated_clock](#)
- [report_clocks](#)
- [report_timing](#)
- [set_clock_latency](#)
- [set_system_jitter](#)

set_load

Set capacitance on ports and nets.

Syntax

```
set_load [-rise] [-fall] [-max] [-min] [-quiet]  
[-verbose] capacitance objects
```

Returns

Nothing

Usage

Name	Description
[-rise]	Specify the rise capacitance value (for ports only)
[-fall]	Specify the fall capacitance value (for ports only)
[-max]	Specify the maximum capacitance value
[-min]	Specify the minimum capacitance value
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>capacitance</i>	Capacitance value
<i>objects</i>	List of ports or nets

Categories

[SDC](#), [XDC](#)

Description

Sets the load capacitance on output ports to the specified value. The load capacitance is used during power analysis when running the **report_power** command, but is not used during timing analysis.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

- max** - (Optional) Specify the maximum load capacitance value.
- min** - (Optional) Specify the minimum load capacitance value.
- rise** - (Optional) Defines the rising edge load capacitance on the specified ports.

-fall - (Optional) Defines the falling edge load capacitance on the specified ports.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

capacitance - (Required) The value of the load capacitance. The value is specified as a floating point value ≥ 0 . The default is 0.

Note The default unit of capacitance is picofarads (pF).

objects - (Required) A list of output port objects to assign the capacitance load to. All outputs in the design may be obtained using the **all_outputs** command.

Examples

The following example sets the specified load capacitance value for all ports:

```
set_load 5.5 [all_outputs]
```

The following example sets the rising and falling edge load capacitance for the specified output ports:

```
set_load -rise -fall 8 [get_ports wbOutput*]
```

See Also

- [all_outputs](#)
- [get_ports](#)
- [report_power](#)

set_logic_dc

Sets logic dc for port/pins.

Syntax

```
set_logic_dc [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	List of ports or pins

Categories

[SDC](#), [XDC](#)

Description

Sets the specified input ports or input pins to a logic value of 'X', as unknown or don't care. This command is NOT supported in Synthesis.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) A list of the input ports and pins to be affected.

Examples

The following example sets the specified port to 'X':

```
set_logic_dc [get_ports reset]
```

See Also

- [all_inputs](#)
- [get_pins](#)
- [get_ports](#)
- [set_logic_one](#)
- [set_logic_unconnected](#)
- [set_logic_zero](#)

set_logic_one

Sets logic one for port/pins.

Syntax

```
set_logic_one [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	List of ports or pins

Categories

[SDC](#), [XDC](#)

Description

Sets the specified input ports or input pins to a logic one. This command is NOT supported in Synthesis.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) A list of the input ports and pins to be affected.

Examples

The following example sets the specified input port to a logic one:

```
set_logic_one [get_ports reset]
```

The following example sets the input ports reset and wbDataForInput to a logic one:

```
set_logic_one [list [get_ports reset] [get_ports wbDataForInput]]
```

The following example sets the input pin I on instance reset_IBUF to a logic one:

```
set_logic_one [get_pins reset_IBUF_inst/I]
```

See Also

- [all_inputs](#)
- [get_pins](#)
- [get_ports](#)
- [set_logic_dc](#)
- [set_logic_unconnected](#)
- [set_logic_zero](#)

set_logic_unconnected

Sets logic unconnected for port/pins.

Syntax

```
set_logic_unconnected [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	List of ports or pins

Categories

[SDC](#), [XDC](#)

Description

Defines the specified output ports or pins as unconnected.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) A list of the output ports and pins to be affected.

Examples

The following example sets the specified port to unconnected:

```
set_logic_unconnected [get_ports OUT1]
```

See Also

- [set_logic_dc](#)
- [set_logic_one](#)
- [set_logic_zero](#)

set_logic_zero

Sets logic zero for port/pins.

Syntax

```
set_logic_zero [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	List of ports or pins

Categories

[SDC](#), [XDC](#)

Description

Sets the specified input ports and input pins to a logic zero. This command is NOT supported in Synthesis.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) A list of the input ports and pins to be affected.

Examples

The following example sets the specified port to logic state 0:

```
set_logic_zero [get_ports reset]
```

See Also

- [all_inputs](#)
- [get_pins](#)
- [get_ports](#)
- [set_logic_one](#)
- [set_logic_unconnected](#)
- [set_logic_zero](#)

set_max_delay

Specify maximum delay for timing paths.

Syntax

```
set_max_delay [-rise] [-fall] [-reset_path] [-from args]
[-rise_from args] [-fall_from args] [-to args] [-rise_to args]
[-fall_to args] [-through args] [-rise_through args]
[-fall_through args] [-datapath_only] [-quiet] [-verbose] delay
```

Returns

Nothing

Usage

Name	Description
<code>[-rise]</code>	Delay value applies to rising path delays
<code>[-fall]</code>	Delay value applies to falling path delays
<code>[-reset_path]</code>	Reset this path before setting max delay
<code>[-from]</code>	List of path startpoints or clocks
<code>[-rise_from]</code>	Apply to paths rising from the list of startpoints or clocks
<code>[-fall_from]</code>	Apply to paths falling from the list of startpoints or clocks
<code>[-to]</code>	List of path endpoints or clocks
<code>[-rise_to]</code>	Apply to paths with rise transition at the list of endpoints or clocks
<code>[-fall_to]</code>	Apply to paths with fall transition at the list of endpoints or clocks
<code>[-through]</code>	List of through pins, cells or nets
<code>[-rise_through]</code>	Apply to paths rising through pins, cells or nets
<code>[-fall_through]</code>	Apply to paths falling through pins, cells or nets
<code>[-datapath_only]</code>	Remove clock skew and jitter from calculation
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>delay</code>	Delay value

Categories

[SDC](#), [XDC](#)

Description

Sets the maximum delay allowed (in time units) on a timing path. The specified delay value is assigned to both the rising and falling edges of the defined timing paths unless the **-rise** or **-fall** arguments are specified.

Note This command operates silently and does not return direct feedback of its operation.

The maximum rising and falling delay cannot be less than the minimum rising and falling delay on the same path. If this happens, the older assigned delay value is removed from the timing path and reset to 0.

The delay value must be assigned to a timing path as defined by at least one **-from**, **-through**, or **-to** argument. A general path delay such as **-to** endpoint will be over written by a more specific path definition such as **-from/-to**, or **-from/-through/-to** path definition.

Arguments

-rise - (Optional) Apply the delay value to the rising edge of the timing path.

-fall - (Optional) Apply the delay value to the falling edge of the timing path.

Note If neither **-rise** nor **-fall** is specified, the delay is applied as both rising and falling edge path delay.

-reset_path - (Optional) Indicates that existing rising or falling edge max delays should be cleared before applying the new specified path delay. If only **-to** is specified all paths leading to the specified endpoints are cleared. If only **-from** is specified, all paths leading from the specified start points are cleared. When **-from/-to** or **-from/-through/-to** are specified, the defined paths are reset.

-from value - (Optional) A list of path start points or clocks. A valid startpoint is a primary input or inout port, or the clock pin of a sequential element. If a clock is specified then all the primary input and inout ports related to that clock as well as all the clock pin of the registers connected to that clock are used as starpoints.

-rise_from element_name - (Optional) The max delay applied to paths rising from the list of origins or clocks.

-fall_from element_name - (Optional) The max delay applied to paths falling from the list of origins or clocks.

-to element_name - (Optional) A list of path endpoints or clocks. A valid endpoint is a primary output or inout port, or the data pin of a sequential element. If a clock is specified then all the primary output and inout ports related to that clock as well as all the data pins of the registers connected to that clock are used as endpoints.

-rise_to element_name - (Optional) The max delay applied to paths with rise transition at the list of endpoints or clocks.

-fall_to element_name - (Optional) The max delay applied to paths with fall transition at the list of endpoints or clocks.

-through element_name - (Optional) A list of through pins, cells, or nets.

-rise_through element_name - (Optional) The max delay applied to paths rising through pins, cells or nets.

-fall_through *element_name* - (Optional) The max delay applied to paths falling through pins, cells or nets.

-datapath_only - (Optional) Exclude clock skew and jitter from the delay calculation for the specified path. This option is used to constrain the delay between sequential elements that have different clocks, where you do not want to consider clock skew and jitter in the delay calculation. Only the Clock-to-Q delay of the first flop, the wire delay between the flops, and the setup time of the second flop should be considered.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

delay - (Required) Specifies the maximum delay value in time units. The *delay* can be specified as a floating point number greater than or equal to 0, with a default value of > 0.

Examples

The following example defines a maximum delay of 60 ns between all the input and output ports (feedthrough paths):

```
set_max_delay 60 -from [all_inputs] -to [all_outputs]
```

The following example clears the existing max delay and specifies a new > maximum delay for paths to endpoints clocked by the specified clock:

```
set_max_delay -reset_path 50 -to [get_clocks spi_clk]
```

The **set_max_delay** command is often used to define timing constraints for crossing clock domains when a simple synchronizer is used. In the following example, two flops (FF1 and FF2) are clocked by different clocks, and FF1/Q connects directly to FF2/D through net1. To limit the delay on this connection to 4.0 ns use one of the following constraints:

```
set_max_delay -from FF1 -to FF2 -datapath_only 4.0
set_max_delay -from FF1/Q -to FF2/D -datapath_only 4.0
set_max_delay -through net1 -datapath_only 4.0
```

See Also

- [get_clocks](#)
- [get_nets](#)
- [get_ports](#)
- [report_timing](#)
- [set_min_delay](#)
- [set_units](#)

set_max_time_borrow

Limit time borrowing for latches.

Syntax

```
set_max_time_borrow [-quiet] [-verbose] delay objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>delay</i>	Delay value: Value >= 0
<i>objects</i>	List of clocks, cells, data pins or clock pins

Categories

[SDC](#), [XDC](#)

Description

Sets the maximum amount of time that can be borrowed between nets when analyzing the timing on latches.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

delay - (Required) The delay (in time units) that should be applied to the specified objects. The *delay* can be specified as a floating point number >= 0, with a default value of 0.

Note The units of time are defined by the **set_units** command. The default unit of time is nanoseconds (ns).

objects - (Required) A list of clocks, cells, data pins, or clock pins to which the limit should be applied.

Examples

The following example specifies that the latches attached to "all clocks" will be allowed 0 time units of borrowing. Effectively, this disables time borrowing throughout the entire design.

```
set_max_time_borrow 0.0 [all_clocks]
```

The following example specifies that nets in the top level of hierarchy are allowed 20 time units of time borrowing:

```
set_max_time_borrow 20 {top/*}
```

See Also

- [all_clocks](#)
- [get_clocks](#)
- [get_nets](#)

set_min_delay

Specify minimum delay for timing paths.

Syntax

```
set_min_delay [-rise] [-fall] [-reset_path] [-from args]
[-rise_from args] [-fall_from args] [-to args] [-rise_to args]
[-fall_to args] [-through args] [-rise_through args]
[-fall_through args] [-quiet] [-verbose] delay
```

Returns

Nothing

Usage

Name	Description
<code>[-rise]</code>	Delay value applies to rising path delays
<code>[-fall]</code>	Delay value applies to falling path delays
<code>[-reset_path]</code>	Reset this path before setting min delay
<code>[-from]</code>	List of path startpoints or clocks
<code>[-rise_from]</code>	Apply to paths rising from the list of startpoints or clocks
<code>[-fall_from]</code>	Apply to paths falling from the list of startpoints or clocks
<code>[-to]</code>	List of path endpoints or clocks
<code>[-rise_to]</code>	Apply to paths with rise transition at the list of endpoints or clocks
<code>[-fall_to]</code>	Apply to paths with fall transition at the list of endpoints or clocks
<code>[-through]</code>	List of through pins, cells or nets
<code>[-rise_through]</code>	Apply to paths rising through pins, cells or nets
<code>[-fall_through]</code>	Apply to paths falling through pins, cells or nets
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>delay</code>	Delay value

Categories

[SDC](#), [XDC](#)

Description

Sets the minimum delay allowed (in time units) on a timing path. The specified delay value is assigned to both the rising and falling edges of the defined timing paths unless the **-rise** or **-fall** arguments are specified.

Note This command operates silently and does not return direct feedback of its operation.

The minimum rising and falling delay cannot be greater than the maximum rising and falling delay on the same path. If this happens, the older assigned delay value is removed from the timing path and reset to 0.

The delay value must be assigned to a timing path as defined by at least one **-from**, **-through**, or **-to** argument. A general path delay such as **-to** endpoint will be over written by a more specific path definition such as **-from/-to**, or **-from/-through/-to** path definition.

Arguments

-rise - (Optional) Apply the delay value to the rising edge of the timing path.

-fall - (Optional) Apply the delay value to the falling edge of the timing path.

-reset_path - (Optional) Clear existing rising or falling edge min delays before applying the new specified path delay. If only **-to** is specified all paths leading to the specified endpoints are cleared. If only **-from** is specified, all paths leading from the specified starting points are cleared. When **-from/-to** or **-from/-through/-to** are specified, the defined paths are reset.

-from objects - (Optional) The starting points of the timing paths that will be assigned the specified delay. A valid startpoint is a primary input or inout port, or the clock pin of a sequential element. If a clock is specified then all the primary input and inout ports related to that clock, as well as all the clock pins of the registers connected to that clock are used as startpoints.

-rise_from objects - (Optional) The starting points of the timing path that will have the specified delay assigned to its rising edge.

-fall_from objects - (Optional) The starting points of the timing path that will have the specified delay assigned to its falling edge.

-to objects - (Optional) The destination objects for the path that will be affected by the minimum delay. A valid endpoint is a primary output or inout port, or the data pin of a sequential element. If a clock is specified then all the primary output and inout ports related to that clock, as well as all the data pins of the registers connected to that clock are used as endpoints.

-rise_to objects - (Optional) The destination objects for the rising-edge path that will be affected by the minimum delay.

-fall_to objects - (Optional) The destination objects for the falling-edge path that will be affected by the minimum delay.

-through objects - (Optional) A list of pins, cell, or nets through which the path affected by the minimum delay travels.

-rise_through objects - (Optional) A list of pins, cell, or nets through which the rising-edge path affected by the minimum delay travels.

-fall_through objects - (Optional) A list of pins, cell, or nets though which the falling-edge path affected by the minimum delay travels.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

delay - (Required) Specifies the minimum delay value in units of time. The *delay* can be specified as a floating point number ≥ 0 , with a default value of 0.

Note The units of time are defined by the **set_units** command. The default unit of time is nanoseconds (ns).

Examples

The following example specifies a minimum delay of 20ns between the primary input and output ports (combinational/feedthrough paths):

```
set_min_delay 20 -from [all_inputs] -to [all_outputs]
```

The following example defines a minimum delay of 20ns for timing paths with endpoints at all primary output ports:

```
set_min_delay 20 -to [get_ports -filter {DIRECTION == out}]
```

See Also

- [get_clocks](#)
- [get_nets](#)
- [get_ports](#)
- [report_timing](#)
- [set_max_delay](#)
- [set_units](#)

set_msg_config

Configure how the Vivado tool will display and manage specific messages, based on message ID, string, or severity.

Syntax

```
set_msg_config [-id arg] [-string args] [-severity arg] [-limit arg]
[-new_severity arg] [-suppress] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-id]	A qualifier, apply the selected operation only to messages that match given message id. Example: '-id {Common 17-35}'. Default: match any id
[-string]	A qualifier, apply the selected operation only to messages that contain the given list of strings. Default: none
[-severity]	A qualifier, apply the selected operation only to messages at the given severity level. Example: '-severity INFO' Default: match any severity
[-limit]	for the messages that match the qualifiers, limit the number of messages displayed to the given integer value for the current project
[-new_severity]	for the messages that match the qualifiers, change the severity to the given value for the current project
[-suppress]	for the messages that match the qualifiers, suppress (do not display) any messages for the current project
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

This command lets you configure the messages returned by the Vivado tool in the current project. Use this command to change the severity of messages, to limit the number of times a message is reported, or to suppress the message altogether. However, you can only perform one of these actions at one time with **set_msg_config**:

- Customize the severity of messages returned by the tool to specific levels appropriate to your usage. For instance, set the severity of a specified message ID from one type, such as WARNING, to another type, such as ERROR.
- Define the number of messages that will be returned by the tool during a design session, or single invocation. You can specify the limit of a specific message ID, or the limit for a specific severity of messages.
- Suppress a specific message ID from being reported by the tool at all. You can enable messages that were previously suppressed using the **reset_msg_config** command.
- An error is returned if more than one action is attempted in a single **set_msg_config** command.

Message qualifiers of string, ID, and severity are used to determine which messages are configured by the **set_msg_config** command. You must supply at least one message qualifier to identify a message or group of messages to apply the command to. Multiple qualifiers have an AND relationship; the configuration rule will be applied only to messages matching all qualifiers.

Note **set_msg_config** does not support the use of wildcards in message qualifiers

Message configuration rules are project specific, and are persistent with the project when the project is closed and reopened. Use the **get_msg_config** command to report the current configuration of a specific message, or the configuration rules defined in the current project.

Restore messages to their default configurations using the **reset_msg_config** command.

Note The default message limit for all message IDs is set to 100, and is defined by the parameter **messaging.defaultLimit**. This is the limit applied to each separate message returned by the tool. You can report the current value of this parameter with the **get_param** command, and change it as needed using the **set_param** command.

Arguments

-id arg - (Optional) Specify the message ID to configure. Every message delivered by the tool has a unique global message ID that consists of an application sub-system code and a message identifier. This results in a message ID that looks like the following:

```
"Common 17-54"  
"Netlist 29-28"  
"Synth 8-3295"
```

-string args - (Optional) Apply the selected operation only to messages that contain the given list of strings. Strings must be enclosed in braces, and multiple strings can be specified separated by spaces:

```
{{Vivado} {All Programmable}}
```

Note Strings are case sensitive.

severity - (Required) The severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

Note Since this is a two word value, it must be enclosed in {} or "".

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

Note Because STATUS messages do not have message IDs, you cannot change the severity level of a STATUS message.

-limit *arg* - (Optional) The message limit specified as an integer value ≥ 1 . You can restore the message limit to the **messaging.defaultLimit** by specifying a count of -1.

-new_severity *arg* - (Optional) For the messages that match the qualifier, specify a new message severity. Valid values are defined above under the **-severity** option.

-suppress - (Optional) Suppress the specified messages from further reporting.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example elevates a common INFO message to a Critical Warning:

```
set_msg_config -id "Common 17-81" -new_severity "CRITICAL WARNING"
```

The following example suppresses messages with the specified message ID:

```
set_msg_config -suppress -id {HDL 9-1654}
```

The following example results in warning messages with message ID "17-35", and containing "clk" in the message, being redefined as Error messages:

```
set_msg_config -severity warning -string "clk" -id "17-35" -new_severity error
```

The following example gets the current default message limit, specifies a new default value, then defines a different message limit for the specified message id:

```
get_param messaging.defaultLimit  
100  
set_param messaging.defaultLimit 1000  
set_msg_config -id "common 17-81" -limit 1500
```

See Also

- [get_msg_config](#)
- [get_param](#)
- [reset_msg_config](#)
- [set_param](#)

set_msg_limit

Set message limit.

Syntax

```
set_msg_limit [-severity arg] [-id arg] [-quiet] [-verbose] count
```

Returns

New message limit

Usage

Name	Description
[-severity]	Message severity to be set (not valid with -id,) e.g. "ERROR" or "CRITICAL WARNING" Default: ALL
[-id]	Unique message id to be set (not valid with -severity,) e.g. "Common 17-99"
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>count</i>	New message limit

Categories

[Report](#)

Description

Defines the number of messages that will be returned by the tool during a design session, or single invocation. You can specify the limit for a specific message ID, such as "Common 17-54", or specify a limit for all messages of a specific severity, such as "CRITICAL WARNING".

Every message delivered by the tool has a unique global message ID that consists of an application sub-system code and a message identifier. This results in a message ID that looks like the following:

```
"Common 17-54"  
"Netlist 29-28"  
"Synth 8-3295"
```

You can report the current message limit of a message severity or ID with the **get_msg_limit** command. You can restore the default message limit using the **reset_msg_limit** command.

If you do not specify either a message ID or a message severity as a qualifier for the **set_msg_limit** command, an error is returned.

The default message limit for each message ID is defined by the parameter **messaging.defaultLimit**. This is the limit applied to each separate message returned by the tool. You can report the current value of this parameter with the **get_param** command, and change it as needed using the **set_param** command.

You can change the severity of messages, limit the count of messages, and suppress messages altogether using the **set_msg_config** command.

Arguments

-id value - (Optional) The message ID, which is included in all returned messages. For example, "Common 17-54" and "Netlist 29-28".

-severity value - (Optional) The severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

Note Since this is a two word value, it must be enclosed in {}.

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

count - (Required) The message limit specified as a value ≥ 1 . You can restore the message limit to the **messaging.defaultLimit** by specifying a count of -1.

Examples

The following example sets the message limit of the specified message ID:

```
set_msg_limit -id "Netlist 29-28" 10000
```

The following example sets the limit of the specified message severity:

```
set_msg_limit -severity WARNING 50000
```

The following example gets the current default message limit, specifies a new default value, then overrides the default for the specified message id:

```
get_param messaging.defaultLimit  
100  
set_param messaging.defaultLimit 1000  
set_msg_limit -id "common 17-81" 1500
```

See Also

- [get_msg_config](#)
- [get_param](#)
- [reset_msg_limit](#)
- [set_msg_config](#)
- [set_param](#)

set_msg_severity

Set Message Severity by ID.

Syntax

```
set_msg_severity [-quiet] [-verbose] id severity
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>id</i>	Unique message id to be set, e.g. "Common 17-54"
<i>severity</i>	Message severity to be changed to, e.g. "ERROR" or "CRITICAL WARNING"

Categories

[Report](#)

Description

While the **set_msg_config** command allows you to define message severity, message limits, and suppress messages. This command only sets the severity of a specified message ID from one type, such as WARNING, to another type, such as ERROR.

Every message delivered by the tool has a unique global message ID that consists of an application sub-system code and a message identifier. This results in a message ID that looks like the following:

```
"Common 17-54"  
"Netlist 29-28"  
"Synth 8-3295"
```

Use this command to customize the message severity returned by the tool to specific levels appropriate to your usage.

Note You can restore the message severity of a specific message ID to its original setting with the **reset_msg_severity** command.

Arguments

id - (Required) The message ID, which is included in all returned messages. For example, "Common 17-54" or "Netlist 29-28".

severity - (Required) The severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.
Note Since this is a two word value, it must be enclosed in {} or "".
- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

Note Because STATUS messages do not have message IDs, you cannot change the severity level of a STATUS message.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example reduces the significance of message ID **Common 17-54** from a CRITICAL WARNING to a WARNING so that it causes less concern when encountered:

```
set_msg_severity "Common 17-54" WARNING
```

The following example elevates a common INFO message to a Critical Warning:

```
set_msg_severity "Common 17-81" "CRITICAL WARNING"
```

See Also

- [get_msg_config](#)
- [reset_msg_severity](#)
- [set_msg_config](#)

set_multicycle_path

Define multicycle path.

Syntax

```
set_multicycle_path [-setup] [-hold] [-rise] [-fall]
[-start] [-end] [-reset_path] [-from args] [-rise_from args]
[-fall_from args] [-to args] [-rise_to args] [-fall_to args]
[-through args] [-rise_through args] [-fall_through args] [-quiet]
[-verbose] path_multiplier
```

Returns

Nothing

Usage

Name	Description
[-setup]	Only setup multiplier is set
[-hold]	Only hold multiplier is set
[-rise]	Multiplier valid for rising delays on path endpoint
[-fall]	Multiplier valid for falling delays on path endpoint
[-start]	Multiplier measured against path startpoint
[-end]	Multiplier measured against path endpoint
[-reset_path]	Reset this path before setting multicycle
[-from]	List of path startpoints or clocks
[-rise_from]	Apply to paths rising from the list of startpoints or clocks
[-fall_from]	Apply to paths falling from the list of startpoints or clocks
[-to]	List of path endpoints or clocks
[-rise_to]	Apply to paths with rise transition at the list of endpoints or clocks
[-fall_to]	Apply to paths with fall transition at the list of endpoints or clocks
[-through]	List of through pins, cells or nets
[-rise_through]	Apply to paths rising through pins, cells or nets
[-fall_through]	Apply to paths falling through pins, cells or nets
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>path_multiplier</i>	Number of cycles

Categories

SDC, XDC

Description

By default, the Vivado timing engine performs a single-cycle analysis, in which the setup check is performed at the destination on the capture edge, one clock cycle after the edge of the start clock. This may not be appropriate for certain timing paths. The most common example is a logic path that requires more than one clock cycle for the data to stabilize at the endpoint.

This command lets you choose a path multiplier, N, to establish a timing path that takes N clock cycles from the start clock edge to the capture clock edge. The path multiplier defines the total number of clock cycles required for propagation of a signal from its origin to destination when that propagation is longer than a single clock cycle. For more information on the use of this command, refer to the *Vivado Design Suite User Guide: Using Constraints (UG903)*.

The **set_multicycle_path** command is used to specify path multipliers for setup and/or hold analysis, for rising and/or falling edges, with respect to the source clock or the destination clock. This command includes three elements:

- The specification of the timing analysis affected by the multicycle path.
- The definition of the timing paths to which the multicycle path applies.
- The path multiplier defining the number of clock cycles to apply to the timing analysis.

The path multiplier applies to both the setup and hold analysis. The hold analysis is derived from the setup analysis, so it is moved along with the setup analysis. This often results in hold timing failures. You can use a second **set_multicycle_path** command with the **-hold** option to restore the hold analysis to its original location.

By default, the setup path multiplier is applied with respect to the destination clock, and the hold path multiplier is applied with respect to the source clock. Use the **-start** or **-end** options to change the default setup or hold analysis with respect to the source or destination clocks.

This command operates silently when successful, or returns an error if the command fails.

Arguments

-setup - (Optional) Apply the path multiplier only to the setup check.

-hold - (Optional) Apply the path multiplier only to the hold check.

Note When neither **-setup** or **-hold** is used, the path multiplier applies to both setup and hold checks

-rise - (Optional) Apply the multiplier specifically to rising edge delays on the path endpoint.

-fall - (Optional) Apply the multiplier specifically to falling edge delays on the path endpoint.

Note If neither **-rise** or **-fall** is specified, the multiplier is applied to both the rising and falling edge delays.

-start - (Optional) By default, the setup path multiplier is defined with respect to the destination clock (**-end**). To modify the setup requirement with respect to the source clock, the **-start** option must be used.

-end - (Optional) By default, the hold path multiplier is defined with respect to the source clock. To modify the hold requirement with respect to the destination clock, the **-end** option must be used.

Note The **-start/-end** options have no effect when applying a multicycle path constraint on paths clocked by the same clock, or clocked by two clocks having the same waveform, or with no phase shift.

-reset_path - (Optional) Reset the specified path before applying the multicycle path multiplier.

-from args - (Optional) A list of start points on the path that will be affected by the path multiplier.

-rise_from args - (Optional) A list of the start points on the rising-edge path that will be affected by the multicycle path multiplier.

-fall_from args - (Optional) A list of the start points on the falling-edge path that will be affected by the multicycle path multiplier.

-to args - (Optional) A list of the end points on the path that will be affected by the multicycle path multiplier.

-rise_to args - (Optional) A list of the end points on the rising-edge path that will be affected by the multicycle path multiplier.

-fall_to args - (Optional) A list of the end points on the falling-edge path that will be affected by the multicycle path multiplier.

-through args - (Optional) A list of pins, cell, or nets through which the path affected by the multicycle path multiplier travels.

-rise_through args - (Optional) A list of pins, cell, or nets through which the rising-edge path affected by the multicycle path multiplier travels.

-fall_through args - (Optional) Specifies the list of pins, cell, or nets through which the falling-edge path affected by the multicycle path multiplier travels.

Important! Although **-to**, **-through**, and **-from** (in their various forms) are all optional arguments, at least one **-from**, **-to**, or **-through** argument must be specified to define a timing path for the **set_multicycle_path** constraint, or an error will be returned.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

path_multiplier - (Required) The number of clock cycles to move the timing analysis.

Examples

The following example establishes a path multiplier of 3 clock cycles for the setup check of the timing path defined by the **-from/-to** options. A path multiplier of N-1, or 2 in this example, is used for the hold check on the same timing path:

```
set_multicycle_path 3 -setup -from [get_pins data0_reg/C] -to [get_pins data1_reg/D]  
set_multicycle_path 2 -hold -from [get_pins data0_reg/C] -to [get_pins data1_reg/D]
```

Note For more information on the relationship between the setup and hold analysis refer to the *Vivado Design Suite User Guide: Using Constraints (UG903)*

See Also

- [report_timing](#)
- [report_timing_summary](#)
- [set_input_delay](#)
- [set_output_delay](#)

set_operating_conditions

Set operating conditions for power estimation.

Syntax

```
set_operating_conditions [-voltage args] [-grade arg] [-process arg]
[-junction_temp arg] [-ambient_temp arg] [-thetaja arg] [-thetasa arg]
[-airflow arg] [-heatsink arg] [-thetajb arg] [-board arg]
[-board_temp arg] [-board_layers arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-voltage]	List of voltage pairs, e.g., {name value}. Supported voltage supplies vary by family.
[-grade]	Temperature grade. Supported values vary by family. Default: commercial
[-process]	Process data: typical or maximum Default: typical
[-junction_temp]	Junction Temperature (C): auto degC Default: auto
[-ambient_temp]	Ambient Temperature (C): default degC Default: default
[-thetaja]	ThetaJA (C/W): auto degC/W Default: auto
[-thetasa]	ThetaSA (C/W): auto degC/W Default: auto
[-airflow]	Airflow (LFM): 0 to 750 Default: varies by family
[-heatsink]	Dimensions of heatsink: none, low, medium, high, custom Default: medium
[-thetajb]	ThetaJB (C/W): auto degC/W Default: auto
[-board]	Board type: jedec, small, medium, large, custom Default: medium
[-board_temp]	Board Temperature degC
[-board_layers]	Board layers: 4to7, 8to11, 12to15, 16+ Default: 8to11
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#), [Power](#)

Description

Sets the real-world operating conditions that are used when performing analysis of the design. The environmental operating conditions of the device are used for power analysis when running the **report_power** command, but are not used during timing analysis.

Note This command operates silently and does not return direct feedback of its operation.

Operating conditions can be restored to their default values with the use of the **reset_operating_conditions** command.

Current operating conditions can be reported with the **report_operating_conditions** command.

Arguments

-voltage *arg* - (Optional) List of voltage supply names and their values specified in pairs. Supported voltage supply names and their values vary by family. For example if a family supports a voltage supply named Vccint, you can set the supply voltage to 0.8 with the following argument and value : **-voltage {Vccint 0.8}**

-grade *arg* - (Optional) The temperature grade of the target device. Supported values vary by family. The default value is "commercial".

-process *arg* - (Optional) The manufacturing process characteristics to assume. Valid values are "typical" or "maximum". The default value is "typical".

-junction_temp *arg* - (Optional) The device junction temperature used for modeling. Valid values are "auto" or an actual temperature specified in degrees C. The default value is "auto".

-ambient_temp *arg* - (Optional) The environment ambient temperature in degrees C. The default setting is "default".

-thetaja *arg* - (Optional) The Theta-JA thermal resistance used for modeling in degrees C/W. The default setting is "auto".

-thetasa *arg* - (Optional) The Theta-SA thermal resistance used during modeling in degrees C/W. The default setting is "auto".

-airflow [0:750] - (Optional) Linear Feet Per Minute (LFM) airflow to be used for modeling. The default setting varies by device family.

-heatsink *arg* - (Optional) The heatsink profile to be used during modeling. Valid values are: none, low, medium, high, custom. The default setting is "medium".

-thetajb *arg* - (Optional) The Theta-JB thermal resistance used for modeling in degrees C/W. The default setting is "auto".

-board *arg* - (Optional) The board size to be used for modeling. The valid values are: jedec, small, medium, large, custom. The default value is "medium".

-board_temp *arg* - (Optional) The board temperature in degrees Centigrade to be used for modeling.

-board_layers *arg* - (Optional) The number of board layers to be used for modeling. Valid values are: "4to7" for boards with 4 to 7 layers, "8to11" for boards with 8 to 11 layers, "12to15" for boards with 12 to 15 layers, and "16+" for boards with 16 or more layers. The default setting is "12to15".

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example specifies an industrial grade device with an ambient operating temperature of 75 degrees C:

```
set_operating_conditions -grade industrial -ambient_temp 75
```

The following example sets the supply voltage Vccaux to a value of 1.9 :

```
set_operating_conditions -voltage {Vccaux 1.89}
```

The following example sets the manufacturing process corner to maximum:

```
set_operating_conditions -process maximum
```

The following example sets the manufacturing process corner to typical and the voltage supply Vccint to 0.98:

```
set_operating_conditions -process maximum -voltage {Vccint 0.98}
```

See Also

- [report_operating_conditions](#)
- [report_power](#)
- [reset_operating_conditions](#)

set_output_delay

Set output delay on ports.

Syntax

```
set_output_delay [-clock args] [-reference_pin args] [-clock_fall]
[-rise] [-fall] [-max] [-min] [-add_delay] [-network_latency_included]
[-source_latency_included] [-quiet] [-verbose] delay objects
```

Returns

Nothing

Usage

Name	Description
<code>[-clock]</code>	Relative clock
<code>[-reference_pin]</code>	Relative pin or port
<code>[-clock_fall]</code>	Delay is relative to falling edge of clock
<code>[-rise]</code>	Specifies rising delay
<code>[-fall]</code>	Specifies falling delay
<code>[-max]</code>	Specifies maximum delay
<code>[-min]</code>	Specifies minimum delay
<code>[-add_delay]</code>	Don't remove existing input delay
<code>[-network_latency_included]</code>	Specifies network latency of clock already included
<code>[-source_latency_included]</code>	Specifies source latency of clock already included
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>delay</i>	Delay value
<i>objects</i>	List of ports

Categories

[SDC](#), [XDC](#)

Description

Sets the output delays on specified ports.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-clock *arg* - (Optional) Indicates that the delay is relative to the rising edge of the specified clock.

-reference_pin *arg* - (Optional) Specifies that the delay is relative to the specified pin rather than a clock.

-clock_fall - (Optional) Specifies that the delay is relative to a falling edge of the clock rather than rising edge.

-rise - (Optional) Specifies that the delay is for a rising edge.

-fall - (Optional) Specifies that the delay is for a falling edge

-max - (Optional) Specifies that the delay specified should be treated as a maximum threshold.

-min - (Optional) Specifies that the delay specified should be treated as a minimum threshold.

-add_delay - (Optional) Specifies that the delay specified should be added to any existing delay on the path rather than replacing the existing delay.

-network_latency_included - (Optional) Specifies that the network latency of the reference clock is included in the specified delay value.

-source_latency_included - (Optional) Specifies that the source latency of the reference clock is included in the specified delay value.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

delay - (Optional) The delay to apply to the specified ports. Valid values are floating point numbers ≥ 0 , with a default value of 1.0.

objects - (Required) A list of ports to which the delay applies.

Examples

The following example sets an output delay on ports relative to the specified clock:

```
set_output_delay 5.0 -clock [get_clocks cpuClk] [get_ports]
```

The next example is the same as the prior example except that network latency is now included:

```
set_output_delay 5.0 -clock [get_clocks cpuClk] -network_latency_included [get_ports]
```

See Also

- [get_ports](#)
- [set_input_delay](#)

set_package_pin_val

Set user columns on one or more package pins.

Syntax

```
set_package_pin_val [-quiet] [-verbose] column value package_pins...
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<i>column</i>	User column name
<i>value</i>	Value to set
<i>package_pins</i>	Package pin names

Categories

[XDC](#), [PinPlanning](#)

Description

Create user-defined package pin attributes and assign values to specific pins on the package.

User-defined pin attributes can be defined in a CSV file and imported into an I/O Pin Planning project using **read_csv**, or can be edited in the project using this command.

Note Use the **set_property** command to set tool-defined properties of a package pin.

Arguments

column - (Required) Specify the user-defined column name. The column name is case-sensitive. If the column does not already exist, a new attribute is created for package pins. If the user-defined column name already exists, the specified value is assigned to the specified pins.

Note Column refers to the display of the attribute in the Package Pins view in the tool GUI. The result of the command is an attribute on the specified package pins that can be exported with **write_csv** for instance.

value - (Required) Specify the value to assign to the specified column. You can repeat the **set_package_pin_val** command to assign different values to different pins in the same column.

package_pins - (Required) Specify the package pins to assign the value to. You can use the **get_package_pins** command to specify the package pins to set.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example creates a new user-defined column in the Package Pins view, and assigns the value **true** to the specified pin:

```
set_package_pin_val -column track1 -value true -package_pins AK27
```

The following example creates a user-defined column called **Test**, then assigns the value RED to all "AK" package pins, then changes the value to GREEN for the three specified pins:

```
set_package_pin_val -column Test -value RED -package_pins [get_package_pins AK*]
set_package_pin_val -column Test -value GREEN -package_pins {AK1 AK2 AK3}
```

See Also

- [get_package_pins](#)
- [set_property](#)
- [write_csv](#)

set_param

Set a parameter value.

Syntax

```
set_param [-quiet] [-verbose] name value
```

Returns

Newly set parameter value

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Parameter name
<i>value</i>	Parameter value

Categories

[PropertyAndParameter](#)

Description

Sets the value of a user-definable configuration parameter. These parameters configure and control various behaviors of the tool. Refer to **report_param** for a description of currently defined parameters.

You can use the **reset_param** command to restore any parameter that has been modified back to its default setting.

Note Setting a specified parameter value to -1 will disable the feature.

Arguments

name - (Required) The name of the parameter to set the value of. You can only set the value of one parameter at a time.

value - (Required) The value to set the specified parameter to.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example sets the parameter defining how many threads to run for multi-threaded processes, including Placement, Routing, and Timing Analysis:

```
set_param general.maxThreads 4
```

Note The Vivado tool supports between 1 to 4 threads. Use **get_param** to determine the current setting.

The following example sets a new default value for message limit:

```
set_param messaging.defaultLimit 1000
```

See Also

- [get_param](#)
- [list_param](#)
- [report_param](#)
- [reset_param](#)

set_power_opt

Set constraints for power optimization.

Syntax

```
set_power_opt [-include_cells args] [-exclude_cells args]  
[-clocks args] [-cell_types args] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-include_cells]</code>	Include only these instances for clock gating. Default: all
<code>[-exclude_cells]</code>	Exclude these instances from clock gating. Default: none
<code>[-clocks]</code>	Clock gate instances clocked by these clocks only. Default: all clocks
<code>[-cell_types]</code>	Clock gate these types only. One of [all bram reg srl none]. Default: all. Default: all
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

Power, XDC

Description

Specify cell instances to include in power optimization. The specified cells are optimized using the **power_opt_design** command.

The effect of multiple **set_power_opt** commands is cumulative, so that you can specify a broad class of cell types to optimize, include specific hierarchical cells, and then exclude cells within the included hierarchy to refine the power optimization.

The power optimizations that have been performed can be reported using the **report_power_opt** command.

Arguments

-include_cells *args* - (Optional) Include only these instances for clock gating. Use this option to list specific cells or blocks to be optimized using **power_opt_design**. The default is to include all cells in power optimization.

-exclude_cells *args* - (Optional) Exclude these instances from clock gating. The default is to not exclude cells from power optimization. The **-exclude_cells** option excludes from the currently included cells. By default all cells are included, however, if **-include_cells** has been specified, then **-exclude_cells** applies only to the currently included cells.

-clocks *args* - (Optional) Perform power optimizations on instances clocked by the specified clocks only. The default is to include all clocks in the design.

Note It is possible to use both **-clocks** and **-include_cells** to produce a list of cells that are not clocked by the specified clocks, resulting in no power optimization

-cell_types [**all** | **bram** | **reg** | **srl** | **none**] - (Optional) Perform power optimization on the specified cell types only. The default is to perform power optimization on all types of cells. You can use **all** or **none** to reset, or clear, any prior **set_power_opt** commands.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example sets power optimization for BRAM cells only, and then runs power optimization:

```
set_power_opt -cell_types bram
power_opt_design
```

The following example sets power optimization for BRAM and REG type cells, then adds SRLs, and runs power optimization. Then all cells are cleared, and only SRLs are included, and power optimization is run again:

```
set_power_opt -cell_types { bram reg}
set_power_opt -cell_types { srl}
power_opt_design
set_power_opt -cell_types { none}
set_power_opt -cell_types { srl}
power_opt_design
```

The following example sets power optimization for BRAM cells only, excludes the cpuEngine block from optimization, but then includes the cpuEngine/cpu_dbg_dat_i block, then performs power optimization:

```
set_power_opt -cell_types bram
set_power_opt -exclude_cells cpuEngine
set_power_opt -include_cells cpuEngine/cpu_dbg_dat_i
power_opt_design
```

See Also

- [power_opt_design](#)
- [report_power_opt](#)

set_propagated_clock

Specify propagated clock latency.

Syntax

```
set_propagated_clock [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	List of clocks, ports, or pins

Categories

[SDC](#), [XDC](#)

Description

Propagates clock latency throughout a clock network, resulting in more accurate skew and timing results throughout the clock network.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) A list of the clock objects to force propagation on.

Examples

This example specifies that the primary system clock from the top-level should be propagated:

```
set_propagated_clock [get_clocks top/clk]
```

This example specifies that all clocks from "sublevel1" should be propagated:

```
set_propagated_clock [get_clocks sublevel1/*]
```

See Also

- [get_clocks](#)
- [create_clock](#)

set_property

Set property on object(s).

Syntax

```
set_property [-dict args] [-quiet] [-verbose] name value objects...
```

Returns

The value that was set if success, "" if failure

Usage

Name	Description
[-dict]	list of name/value pairs of properties to set
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name of property to set. Not valid with -dict option
<i>value</i>	Value of property to set. Not valid with -dict option
<i>objects</i>	Objects to set properties on

Categories

[Object](#), [PropertyAndParameter](#), [XDC](#)

Description

Assigns the defined property *name* and *value* to the specified *objects*.

This command can be used to define any property on an object in the design. Each object has a set of predefined properties that have expected values, or a range of values. The `set_property` command can be used to define the values for these properties. To determine the defined set of properties on an object, use **report_property**, **list_property**, or **list_property_values**.

You can also define custom properties for an object, by specifying a unique *name* and *value* pair for the object. If an object has custom properties, these will also be reported by the **report_property** and **list_property** commands.

Arguments

-dict - (Optional) Use this option to specify multiple properties (*name value* pairs) on an object with a single `set_property` command. Multiple *name value* pairs must be enclosed in braces, {}, or quotes, "".

```
-dict "name1 value1 name2 value2 ... nameN valueN"
```

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

name - (Required) Specifies the name of the property to be assigned to the object or objects. The *name* argument is case sensitive and should be specified appropriately.

value - (Required) Specifies the value to assign to the *name* on the specified object or objects. The value is checked against the property type to ensure that the value is valid. If the value is not appropriate for the property an error will be returned.

Important! In some cases the value of a property may include special characters, such as the dash character ('-'), which can cause the tool to interpret the value as a new argument to the command. In this case, you must use the explicit arguments (**-name**, **-value**, **-objects**) instead of the implied positional arguments (name, value, objects) as described here. This is shown in the **Examples** section below

objects - (Required) One or more objects to assign the property to.

Examples

Create a user-defined boolean property, TRUTH, for cell objects, and set the property on a cell:

```
create_property -type bool truth cell
set_property truth false [lindex [get_cells] 1]
```

Use the **-dict** option to specify multiple properties at one time on the current design:

```
set_property -dict "POST_CRC enable POST_CRC_ACTION correct_and_continue" \
[current_design]
```

The following example sets the TOP property of the current fileset to define the top module of the project:

```
set_property top fftTop [current_filesset]
set_property top_file {C:/Data/sources/fftTop.v} [current_filesset]
```

Note Defining the top module requires the TOP property to be set to the desired hierarchical block in the source fileset of the current project. In the preceding example TOP is the property name, fftTop is the value, and current_filesset is the object. In addition, the TOP_FILE property should be defined to point to the data source for the top module

This example shows how to set a property value that includes the dash character, '-'. The dash can cause the tool to interpret the value as a new command argument, rather than part of the value being specified, and will cause an error as shown. In this case, you must use the explicit form of the positional arguments in the set_property command:

```
set_property {XELAB.MORE_OPTIONS} {-pulse_e_style ondetect} [get_filesets sim_1]
ERROR: [Common 17-170] Unknown option '-pulse_e_style ondetect',
please type 'set_property -help' for usage info.
set_property -name {XELAB.MORE_OPTIONS} -value {-pulse_e_style ondetect} \
-objects [get_filesets sim_1]
```

The following example sets the internal VREF property value for the specified IO Bank:

```
set_property internal_vref {0.75} [get_iobanks 0]
```

The following example defines a DCI Cascade by setting the SLAVE_BANKS property for the specified IO Bank:

```
set_property slave_banks {14} [get_iobanks 0 ]
```

The following example configures the synth_1 run, setting options for Vivado Synthesis 2013, and then launches the synthesis run:

```
set_property flow {Vivado Synthesis 2013} [get_runs synth_1]
set_property STEPS.SYNTH_DESIGN.ARGS.FANOUT_LIMIT 500 [get_runs synth_1]
set_property STEPS.SYNTH_DESIGN.ARGS.GATED_CLOCK_CONVERSION on [get_runs synth_1]
set_property STEPS.SYNTH_DESIGN.ARGS.FSM_EXTRACTION one_hot [get_runs synth_1]
launch_runs synth_1
```

See Also

- [current_fileset](#)
- [create_property](#)
- [create_run](#)
- [get_cells](#)
- [get_property](#)
- [get_runs](#)
- [launch_runs](#)
- [list_property](#)
- [list_property_value](#)
- [report_property](#)
- [reset_property](#)

set_speed_grade

Timing Speed Grade.

Syntax

```
set_speed_grade [-quiet] [-verbose] value
```

Returns

String result

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>value</i>	Speed grade used for timing analysis

Categories

[Project](#)

Description

Sets the speed grade for the target device in the current design. This command is used to change the speed grade of the target device for timing analysis. It must be run on an opened Synthesized or Implemented Design. It is usually run prior to the report_timing command or other timing commands to change the speed grade for analysis.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

value - (Required) The speed grade for the target device. Valid values are -1, -2, or -3.

Examples

The following example sets the speed grade for the device in the current design to -1:

```
set_speed_grade -1
```

See Also

[set_property](#)

set_switching_activity

Set switching activity on specified objects or default types.

Syntax

```
set_switching_activity [-type args] [-static_probability arg]
[-signal_rate arg] [-hier] [-quiet] [-verbose] [objects...]
```

Returns

Nothing

Usage

Name	Description
[-type]	Use with RTL power estimation. List of valid type values: registers, inputs, outputs, inout, ports, outputEnable, three_states, dsps, brams, bramWrite, bramEnable, clockEnable
[-static_probability]	Static probability value: 0 <= Value <= 1 Default: 0.0
[-signal_rate]	The number of times an element changed state (high-to-low and low-to-high) per second. Xilinx tools express this as millions of transitions per second (Mtr/s). Default: 0.0
[-hier]	Hierarchically sets the switching activity on a hierarchical instance provided via option. This option should be used only with option
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[objects]	Objects to set switching activity on

Categories

[XDC](#), [Power](#)

Description

Sets the signal rate and the switching probability to be used when performing power estimation. These include simple signal rate and simple static probability on nets, ports, and pins; and state dependent static probabilities on cells.

Note This command operates silently and does not return direct feedback of its operation.

The switching activity of a design affects both the static and dynamic power consumption. The static power is often dependent on logic state transitions, and the dynamic power is directly proportional to the toggle rate.

This command is used to specify a default activity rate for a broad class of signals when performing power estimation. It is used to define the activity of one or more signals, rather than the whole class.

The current switching activity attributes can be found by using the **report_switching_activity** command. The values can be set to their default values by using the **reset_switching_activity** command.

Arguments

-toggle_rate *rate* - (Optional) The toggle rate, which describes how often the output switches relative to the controlling clock. Valid values are between 0 and 200%. An output that switches once per clock cycle toggles at 100%. The default value is 0.

-type *value* - (Optional) The type of logic entity for the defined switching activity. Valid types are: registers, inputs, outputs, inout, ports, outputEnable, three_states, dsps, brams, bramWrite, bramEnable, clockEnable.

-static_probability *value* - (Optional) The switching probability to be used in analysis. Valid values are $0 < \text{value} < 1$. The default value is 0.

-signal_rate *value* - (Optional) The signal frequency to be used for analysis. The default value is 0.

Note One or both of **-static_probability** or **-signal_rate** must be specified with the **set_switching_activity** command.

-hier - (Optional) Apply the switching activity to signals at all levels of a hierarchical instance. Without **-hier**, the switching activity is applied to the specified *objects* at the current level of the hierarchy.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) A list of port, pin, net, and clock objects to which the switching activity configuration should be applied.

Examples

The following example specifies a signal rate and switching probability for all ports, then reports the switching attributes for the ports:

```
set_switching_activity -signal_rate 55 -static_probability .27 [get_ports]
report_switching_activity [get_ports]
```


See Also

- [power_opt_design](#)
- [report_default_switching_activity](#)
- [report_power](#)
- [report_switching_activity](#)
- [reset_default_switching_activity](#)
- [reset_switching_activity](#)
- [set_default_switching_activity](#)

set_system_jitter

Set system jitter.

Syntax

```
set_system_jitter [-quiet] [-verbose] system_jitter
```

Returns

System_jitter

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>system_jitter</i>	System jitter: Value >= 0

Categories

[XDC](#)

Description

Sets the system jitter (in time units) for all clocks in the design, including primary and generated clocks. System jitter is used to account for noise that affects all the clocks within the FPGA, like power supply noise and board noise.

The **set_system_jitter** command applies to all the clocks in the design. Use the **set_input_jitter** command to specify additional jitter for a specific primary clock.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

jitter - (Required) Specifies the system jitter (in time units) to be applied system-wide. The default system jitter is defined in the speed files for specific Xilinx FPGAs. The jitter specified by the **set_system_jitter** command overwrites the default value.

Examples

The following example models a system-wide jitter of 0.2 time units on all the clocks in the design:

```
set_system_jitter 0.2
```

The following example defines a primary clock sysClk and specifies a system wide jitter of 0.1 time units:

```
create_clock -period 10 -name sysClk [get_ports sysClk]
set system_jitter 0.1
```

The following example defines a primary clock, sysClk, and a generated clock, sysClkDiv2, that is a divide by two version of the primary clock. A system jitter of 0.2 time units is specified that applies to all the clocks in the design. An additional input jitter of 0.09 is specified on only the primary clock :

```
create_clock -period 10 -name sysClk [get_ports sysClk]
create_generated_clock -name sysClkDiv2 -source [get_ports sysClk] -divide_by 2 \
[get_pins clkgen/sysClkDiv/Q]
set_system_jitter 0.2
set_input_jitter sysClk 0.09
```

The follow example defines two primary clocks, sysClk and procClk. A system jitter of 0.2 is defined for all the clocks in the system. An additional input jitter of 0.05 is specified for the clock procClk :

```
create_clock -period 10 -name sysClk [get_ports sysClk]
create_clock -period 25 -name procClk [get_ports procClk]
set_system_jitter 0.2
set_input_jitter procClk 0.05
```

See Also

- [report_timing](#)
- [set_input_jitter](#)
- [set_input_delay](#)
- [set_clock_uncertainty](#)

set_units

Set units for checking.

Syntax

```
set_units [-capacitance arg] [-time arg] [-current arg] [-voltage arg]
[-power arg] [-resistance arg] [-suffix arg] [-digits arg] [-quiet]
[-verbose]
```

Returns

Nothing

Usage

Name	Description
[-capacitance]	Capacitance unit in farad. Valid values are from kF-fF. Default: pF
[-time]	Time unit in seconds. Valid values are from ks-fs. Default: ns
[-current]	Current unit in ampere. Valid values are from kA-fA. Default: mA
[-voltage]	Voltage unit in volt. Valid values are from kV-fV. Default: V
[-power]	Wattage unit in watts. Valid values are from kW-fW. Default: mW
[-resistance]	Resistance unit in ohm. Valid values are from kOhm-fOhm. Default: ohm
[-suffix]	Suffix for units
[-digits]	Number of digits Default: 1
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[SDC](#), [XDC](#)

Description

This command specifies the default units to be assumed when the design is analyzed. Specifically, the **-current**, **-voltage**, **-power**, and **-resistance** options impact the values returned by the **report_power** command.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-capacitance *value* - (Optional) Specify the unit of capacitance in Farads. Valid values range from kilofarads (kF) to femtofarads (fF). The default unit of capacitance is picofarads (pF).

-current *value* - (Optional) Specify the default unit of current in amperes. Valid values range from kiloAmps (kA) to femtoAmps (fA). The default unit of amperes is milliAmps (mA).

-voltage *value* - (Optional) Specify the default unit of voltage in Volts. Valid values range from kilovolts (kV) to femtovolts (fV). The default unit of voltage is Volts (V).

-power *value* - (Optional) Specify the default unit of power in watts. Valid values range from kilowatts (kW) to femtowatts (fW). The default unit of power is milliwatts (mW).

-resistance *value* - (Optional) Specify the default unit of resistance in ohms. Valid values range from kilo-ohm (kOhm) to femto-ohm (fOhm). The default unit of resistance is ohms (Ohm).

-suffix *value* - (Optional) Specify the suffix to be used for the specified units.

-digits *value* - (Optional, Default Value of 1) Specify the number of digits to be used when printing units.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

Specify that voltage should be in millivolts and all values should use three digits

```
set_units -voltage mV -digits 3
```

The following example changes the default unit for current to Amperes:

```
set_units -current A
```

The second time that `set_units` is issued above does not over-ride or change the values set in the first instance.

See Also

- [report_power](#)
- [set_operating_conditions](#)

set_value

Set the current value of an HDL object (variable, signal, wire, or reg) to a specified value.

Syntax

```
set_value [-radix arg] [-quiet] [-verbose] hdl_object value
```

Returns

Nothing

Usage

Name	Description
[-radix]	radix specifies the radix to use for interpreting value. Allowed values are: default, dec, bin, oct, hex, unsigned, ascii
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hdl_object</i>	Set the value to the given hdl_object
<i>value</i>	

Categories

[Simulation](#)

Description

Specify the value of a single HDL object at the current simulation run time.

HDL objects include HDL signals, variables, or constants as defined in the Verilog or VHDL testbench and source files. An HDL signal includes Verilog wire or reg entities, and VHDL signals. Examples of HDL variables include Verilog real, realtime, time, and event.

HDL constants include Verilog parameters and localparams, and VHDL generic and constants. The HDL scope, or scope, is defined by a declarative region in the HDL code such as a module, function, task, process, or begin-end blocks in Verilog. VHDL scopes include entity/architecture definitions, block, function, procedure, and process blocks.

Arguments

-radix *arg* - (Optional) Specifies the radix to use when returning the value of the specified object. Allowed values are: **default**, **dec**, **bin**, **oct**, **hex**, **unsigned**, and **ascii**.

Note The radix **dec** indicates a signed decimal. Specify the radix **unsigned** when dealing with unsigned data

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

hdl_object - (Required) Specifies a single HDL object to get the value of. The object can be specified by name, or can be returned as an object from the **get_objects** command.

value - (Required) The value to set the specified object to. The specified *value* depends on the type of the *hdl_object*. HDL object types include: "logic", floating point, VHDL enumerated, and VHDL integral. For all but "logic" the -radix option is ignored.

- "Logic" does not refer to an actual HDL object type, but means any object whose values are similar to those of VHDL std_logic, such as:
 - the Verilog implicit 4-state bit type,
 - the VHDL bit and std_logic predefined types,
 - any VHDL enumeration type which is a subset of std_logic, including the character literals 0 and 1.
- For logic types the value depends on the radix:
 - If the specified value has fewer bits than the logic type expects, the value is zero extended, but not sign extended, to match the expected length.
 - If the specified value has more bits than the logic type expects, the extra bits on the MSB side should all be zeros, or the Vivado simulator will return a "size mismatch" error.
- Accepted values for floating point objects are floating point values.
- The accepted value for non-logic VHDL enumerated types is a scalar value from the enumerated set of values, without single quotes in the case of characters.
- Accepted values for VHDL integral types is a signed decimal integer in the range accepted by the type.

Examples

The following example sets the value of the sysClk signal:

```
set_value sysClk Z
```

This example uses the **bin**, **dec**, and **unsigned** radix to specify the same value on the given bus:

```
set_value -radix bin /test/bench_VStatus_pad_0_i[7:0] 10100101
set_value -radix unsigned /test/bench_VStatus_pad_0_i[7:0] 165
set_value -radix dec /test/bench_VStatus_pad_0_i[7:0] -91
```

The following example shows the bit extension performed when the provided value has fewer bits than the logic type expects :

```
set_value -radix bin /test/bench_VStatus_pad_0_i[7:0] 101
get_value -radix bin /test/bench_VStatus_pad_0_i[7:0]
00000101
```

The following example shows the bit truncation performed when the provided value has more bits than the logic type expects :

```
set_value -radix bin /test/bench_VStatus_pad_0_i[7:0] 0010100101
get_value -radix bin /test/bench_VStatus_pad_0_i[7:0]
10100101
set_value -radix bin /test/bench_VStatus_pad_0_i[7:0] 1110100101
ERROR: [#UNDEF] Object size 8 does not match size of given value 1110100101
```

Note In the second **set_value** command, the extra bits are not zero, and so an error is returned

See Also

- [current_time](#)
- [get_objects](#)
- [get_value](#)
- [report_values](#)

show_objects

Show objects in Find Results view.

Syntax

```
show_objects [-name arg] [-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-name]	Tab title
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	Objects to show Find Results view

Categories

[GUIControl](#)

show_schematic

Show netlist items in schematic view.

Syntax

```
show_schematic [-add] [-remove] [-regenerate] [-pin_pairs] [-name arg]
[-quiet] [-verbose] objects
```

Returns

Nothing

Usage

Name	Description
[-add]	Add to existing schematic view
[-remove]	Remove from existing schematic view
[-regenerate]	Regenerate layout of schematic view
[-pin_pairs]	objects are treated as pair of connected pins. This can be useful to display paths
[-name]	Schematic window title
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>objects</i>	Netlist items to show in schematic view

Categories

[GUIControl](#)

Description

Create a schematic view containing the specified design objects when the tool is invoked in GUI mode.

The scope of the schematic that is displayed depends on the objects specified. A schematic created with cells, shows the specified cells and any connections between the cells. A schematic created with pins, shows the pin objects, or shows them connected as appropriate if **-pin_pairs** is specified. A schematic created with nets shows the nets, as well as the cells and ports connected to the nets.

To display a schematic with multiple levels of hierarchy, use the **current_instance** command to set the top-level of the hierarchy, or the level of interest, and use the **-hierarchical** option when specifying design objects with a **get_*** command.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-add - (Optional) Add the specified objects to the schematic window.

-remove - (Optional) Remove the specified objects from the schematic window.

-regenerate - (Optional) Regenerate the schematic window.

-pin_pairs - (Optional) When specified with a pair of connected pin objects, the schematic shows the pins and the wire between the pins. When the **-pin_pairs** option is not specified, or is specified with disconnected pins, the wire is not shown.

-name arg - (Optional) Defines a name for the schematic window opened in the GUI. Use this name to add to, remove from, or regenerate the schematic.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Required) The netlist objects to display in the schematic window.

Examples

The following example creates a schematic for the top-level of the design, displaying the nets as well as the ports and cells they connect to:

```
show_schematic [get_nets]
```

The following example sets the level of hierarchy of interest, and creates a hierarchical schematic from the current level down:

```
current_instance A  
show_schematic [get_nets -hier]
```

The following example creates a schematic window showing the specified pins, and the wire connection between them:

```
show_schematic -pin_pairs [get_pins {data0_i/O data_reg/D}]
```

See Also

- [current_instance](#)
- [get_cells](#)
- [get_nets](#)
- [get_pins](#)
- [get_ports](#)

split_diff_pair_ports

Remove differential pair relationship between 2 ports.

Syntax

```
split_diff_pair_ports [-quiet] [-verbose] ports...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>ports</i>	Ports to split

Categories

[PinPlanning](#)

Description

Splits an existing differential pair of ports into two single-ended ports.

Note This command operates silently and does not return direct feedback of its operation.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

ports - (Required) The names of two ports of a differential pair to split into single-ended ports.

Examples

The following example splits the specified diff pair ports to form two single ended ports:

```
split_diff_pair_ports PORT_N PORT_P
```

See Also

- [make_diff_pair_ports](#)
- [create_port](#)
- [create_interface](#)

start_gui

Start GUI.

Syntax

```
start_gui [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[GUIControl](#)

Description

Launches the GUI when the tool is running in the Vivado Design Suite Tcl shell. The GUI is invoked with the current project, design, and run information.

Arguments

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example is executed from the command prompt when the tool is running in Tcl mode:

```
Vivado% start_gui
```

See Also

[stop_gui](#)

start_vcd

Start capturing VCD output (equivalent of \$dumpon verilog system task). This can be used after a stop_vcd TCL command has stopped VCD generation started by open_vcd.

Syntax

```
start_vcd [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Simulation](#)

Description

The **start_vcd** command specifies that the tool start writing Value Change Dump (VCD) information into the specified VCD object. This Tcl command models the behavior of the Verilog **\$dumpon** system task.

Note: You must execute the **open_vcd** command before using the **start_vcd** command.

Nothing is returned by this command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example starts the writing of HDL signals into the current VCD file:

```
start_vcd
```

See Also

- [close_vcd](#)
- [open_vcd](#)
- [stop_vcd](#)

startgroup

Start a set of commands that can be undone/redone as a group.

Syntax

```
startgroup [-try] [-quiet] [-verbose]
```

Returns

Int

Usage

Name	Description
-try	Don't start a group if one has already been started
-quiet	Ignore command errors
-verbose	Suspend message limits during command execution

Categories

[GUIControl](#)

Description

Starts a sequence of commands that can be undone or redone as a series. Use **endgroup** to end the sequence of commands.

Note You can have multiple command groups to **undo** or **redo**, but you cannot nest command groups. You must use **endgroup** to end a command sequence before using **startgroup** to create a new command sequence

The **startgroup** command returns an integer value of 0 if a group is already started, and returns an integer value of 1 if the startgroup command has started a new group.

Arguments

-try - (Optional) Returns 1 if a new group is started. Returns 0 if a group has already been started, and therefore a new group cannot be started.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example defines a **startgroup**, executes a sequence of related commands, and then executes the **endgroup**. This sequence of commands can be undone or redone as a group:

```
startgroup
create_pblock pblock_wbArbEngine
create_pblock pblock_usbEngnSRM
add_cells_to_pblock pblock_wbArbEngine [get_cells [list wbArbEngine]] -clear_locs
add_cells_to_pblock pblock_usbEngnSRM [get_cells [list usbEngine1/usbEngineSRAM]] \
-clear_locs
endgroup
```

See Also

- [endgroup](#)
- [redo](#)
- [undo](#)

step

Step simulation to the next statement.

Syntax

```
step [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Simulation](#)

stop

Use within a condition to tell simulation to stop when a condition is true.

Syntax

```
stop [-quiet] [-verbose]
```

Returns

A "stop" in simulation is a pause and not a quit

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Simulation](#)

stop_gui

Close GUI.

Syntax

```
stop_gui [-verbose]
```

Returns

Nothing

Usage

Name	Description
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[GUIControl](#)

Description

Stops GUI mode and places the tool into Tcl mode, running in the Vivado Design Suite Tcl shell. In Tcl mode, all commands must be entered as Tcl commands or through Tcl scripts.

Arguments

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example stops and closes the GUI and places the tool into Tcl mode:

```
stop_gui
```

See Also

[start_gui](#)

stop_hw_sio_scan

Stop hardware SIO scans.

Syntax

```
stop_hw_sio_scan [-quiet] [-verbose] hw_sio_scans
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_sio_scans</i>	hardware SIO scans

Categories

[Hardware](#)

stop_vcd

Stop capturing VCD output (equivalent of \$dumpoff verilog system task). The start_vcd TCL command can be used to resume capturing VCD.

Syntax

```
stop_vcd [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Simulation](#)

Description

Stop writing the simulation values to the current Value Change Dump (VCD) file. This suspends the output of simulation information to the file until the process is resumed using the **start_vcd** command.

This Tcl command models the behavior of the Verilog **\$dumpoff** system task.

Note: You must execute the **open_vcd** command before using the **stop_vcd** command.

Nothing is returned by the command.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example stops writing simulation values to the current VCD file:

```
stop_vcd
```

See Also

- [close_vcd](#)
- [open_vcd](#)
- [start_vcd](#)

swap_locs

Swap two locations.

Syntax

```
swap_locs [-quiet] [-verbose] aloc bloc
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>aloc</i>	First location (port/cell/site - should be of same type as 'bloc')
<i>bloc</i>	Second location (port/cell/site - should be of same type as 'aloc')

Categories

[Floorplan](#)

Description

Swaps the LOC constraints assigned to two similar logic elements. A logic element is an element that can be placed onto a device resource on the FPGA.

Some DRC checking is performed when the **swap_locs** command is executed to ensure that the two selected elements can in fact be assigned to their new locations. If the location of either element is invalid for any reason, the **swap_locs** command will fail and an error will be returned.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

alloc - (Required) The location of the first logic element to swap. This can be specified as a port, a cell, or a device site.

bloc - (Required) The location of the second logic element to swap. This can be specified as a port, a cell, or a device site. This must match the type specified by the *alloc* variable.

Examples

The following example swaps the instances assigned to the two specified device sites:

```
swap_locs [get_sites {OLOGIC_X2Y1}] [get_sites {OLOGIC_X2Y0}]
```

See Also

- [get_cells](#)
- [get_ports](#)
- [get_sites](#)

synth_design

Synthesize a design using Vivado Synthesis and open that design.

Syntax

```
synth_design [-name arg] [-part arg] [-constrset arg] [-top arg]
[-include_dirs args] [-generic args] [-verilog_define args]
[-flatten_hierarchy arg] [-gated_clock_conversion arg]
[-directive arg] [-rtl] [-bufg arg] [-no_lc] [-fanout_limit arg]
[-mode arg] [-fsm_extraction arg] [-keep_equivalent_registers]
[-resource_sharing arg] [-control_set_opt_threshold arg] [-quiet]
[-verbose]
```

Returns

Design object

Usage

Name	Description
[-name]	Design name
[-part]	Target part
[-constrset]	Constraint fileset to use
[-top]	Specify the top module name
[-include_dirs]	Specify verilog search directories
[-generic]	Specify generic parameters. Syntax: -generic = -generic = ...
[-verilog_define]	Specify verilog defines. Syntax: -verilog_define [=] -verilog_define [=] ...
[-flatten_hierarchy]	Flatten hierarchy during LUT mapping. Values: full, none, rebuilt Default: rebuilt
[-gated_clock_conversion]	Convert clock gating logic to flop enable. Values: off, on, auto Default: off
[-directive]	Synthesis directive. Values: default, runtimeoptimized Default: default
[-rtl]	Elaborate and open an rtl design
[-bufg]	Max number of global clock buffers used by synthesis Default: 12
[-no_lc]	Disable LUT combining. Do not allow combining LUT pairs into single dual output LUTs.
[-fanout_limit]	Fanout limit Default: 10000
[-mode]	The design mode. Values: default, out_of_context Default: default

Name	Description
<code>[-fsm_extraction]</code>	FSM Extraction Encoding. Values: off, one_hot, sequential, johnson, gray, auto Default: auto
<code>[-keep_equivalent_registers]</code>	Prevents registers sourced by the same logic from being merged. (Note that the merging can otherwise be prevented using the synthesis KEEP attribute)
<code>[-resource_sharing]</code>	Sharing arithmetic operators. Value: auto, on, off Default: auto
<code>[-control_set_opt_threshold]</code>	Threshold for synchronous control set optimization to lower number of control sets. Default: 1
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

Tools

Description

Directly launches the Vivado synthesis engine to compile and synthesize a design in either Project Mode or Non-Project Mode in the Vivado Design Suite. Refer to the *Vivado Design Suite User Guide: Design Flows Overview (UG892)* for a complete description of Project Mode and Non-Project Mode.

Vivado synthesis can be launched directly with the **synth_design** command in the Non-Project Mode of the Vivado Design Suite.

In Project Mode, synthesis should be launched from an existing synthesis run created with the **create_run** command. The run is launched using the **launch_runs** command, and this in turn calls **synth_design** for Vivado synthesis.

Arguments

-name *arg* - (Optional) The name of the design to open when the **synth_design** command is complete.

-part *arg* - (Optional) The target Xilinx device to use for the design. If the part is not specified the default part assigned to the project will be used.

-constrset *arg* - (Optional) The name of the XDC constraints to use when synthesizing the design. Vivado synthesis requires the use of XDC, and does not support UCF. The **-constrset** argument must refer to a constraint fileset that exists. It cannot be used to create a new fileset. Use the **create_fileset** command for that purpose.

-top *arg* - (Optional) The top module of the design hierarchy.

Note If you use the **find_top** command to define the **-top** option, be sure to specify only one top if **find_top** returns multiple prospects. See the examples below.

-include_dirs *args* - (Optional) The directories to search for Verilog ``include` files.

-generic name=value - (Optional) The value of a VHDL generic entity, or of a Verilog parameter. The syntax for the **-generic** argument is **name=value**, specifying the name of the generic or parameter, and the value to be assigned. Repeat the **-generic** argument multiple times in the **synth_design** command for each generic or parameter value to be defined:

```
synth_design -generic width=32 -generic depth=512 ...
```

Important! When specifying binary values for boolean or std_logic VHDL generic types, you must specify the value using the Verilog bit format, rather than standard VHDL format:

```
0 = 1'b0
01010000 = 8'b01010000
```

-verilog_define name=text - (Optional) Set values for Verilog ``define`, and ``ifdef`, statements. The syntax for the **-verilog_define** argument is **name=text**, specifying the name of the define directive, and the value to be assigned. The argument can be reused multiple times in a single **synth_design** command.

```
synth_design -verilog_define name=value -verilog_define name=value ...
```

-flatten_hierarchy arg - (Optional) Flatten the hierarchy of the design during LUT mapping. The valid values are:

- **rebuilt** - This will attempt to rebuild the original hierarchy of the RTL design after synthesis has completed. This is the default setting.
- **full** - Flatten the hierarchy of the design.
- **none** - Do not flatten the hierarchy of the design. This will preserve the hierarchy of the design, but will also limit the design optimization that can be performed by the synthesis tool.

-gated_clock_conversion arg - (Optional) Convert clock gating logic to utilize the flop enables where available. This optimization can eliminate logic and simplify the netlist. Valid values are **off**, **on**, **auto**. The default setting is **off**. This optimization can also be performed on the synthesized netlist through the use of the **opt_design** command.

-directive [default | runtimeoptimized] - (Optional) Direct synthesis to achieve specific design objectives. Only one directive can be specified for a single **synth_design** command, and values are case-sensitive. Valid values are **default** and **runtimeoptimized**. The latter indicates that fewer timing optimizations will be performed, and some RTL optimizations will not be performed.

-rtl - (Optional) Elaborate the HDL source files and open the RTL design.

-bufg arg - (Optional) Specify the maximum number of global clock buffers to be used during synthesis. Specify a value ≥ 1 , which should not exceed the BUFG count on the target device. The default value is 12.

-no_lc - (Optional) Disable the default LUT combining feature of Vivado synthesis.

-fanout_limit arg - (Optional) Limit the maximum net fanout applied during synthesis. Specify a value ≥ 1 . The default value is 10,000.

-mode [default | out_of_context] - (Optional) Out of Context mode specifies the synthesis of a module for use in hierarchical design. This mode turns off I/O buffer insertion for the module, and marks it as OOC, to facilitate its use in the HD flow. Refer to the *Vivado Design Suite User Guide: Hierarchical Design (UG905)* for more information.

-fsm_extraction arg - (Optional) Finite state machine (FSM) encoding is disabled (**off**) in Vivado synthesis by default. This option enables state machine identification and specifies the type of encoding that should be applied. Valid values are: **off, one_hot, sequential, johnson, gray, auto**. Automatic encoding (**auto**) allows the tool to choose the best encoding for each state machine identified. In this case, the tool may use different encoding styles for different FSMs in the same design.

-keep_equivalent_registers - (Optional) Works like the synthesis KEEP attribute to prevent the merging of registers during optimization.

-resource_sharing arg - (Optional) Share arithmetic operators like adders or subtractors between different signals, rather than creating new operators. This can result in better area utilization when it is turned on. Valid values are: **auto, on, off**. The default is **auto**.

-control_set_opt_threshold arg - (Optional) Threshold for synchronous control set optimization to decrease the number of control sets. Specifies how large the fanout of a control set should be before it starts using it as a control set. For example, if **-control_set_opt_threshold** is set to 10, a synchronous reset that only fans out to 3 registers would be moved to the D input logic, rather than using the reset line of a register. However, if set to 1, then the reset line is used. This option is specified as an integer greater than or equal to 1; the default is 1.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example uses the **set_property** command to define the target part for the active project, then elaborates the source files and opens an RTL design:

```
set_property part xc7vx485tffg1158-1 [current_project]
synth_design -rtl -name rtl_1
```

Note The default source set, constraint set, and part will be used in this example.

The following example uses the **find_top** command to define the top of the current design for synthesis:

```
synth_design -top [lindex [find_top] 0]
```

Note Since **find_top** returns multiple possible candidates, choosing index 0 chooses the best top candidate for synthesis.

The following example runs synthesis on the current design, defining the top module and the target part, and specifying no flattening of the hierarchy. The results of the synthesis run are then opened in a netlist design:

```
synth_design -top top -part xc7k70tfbg676-2 -flatten_hierarchy none
open_run synth_1 -name netlist_1
```

See Also

- [create_run](#)
- [current_design](#)
- [current_project](#)
- [find_top](#)
- [open_run](#)
- [opt_design](#)
- [set_property](#)

tie_unused_pins

Tie off unused cell pins.

Syntax

```
tie_unused_pins [-of_objects args] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-of_objects]	tie unused pins of specified cell(s)
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Netlist](#)

Description

Tie up or down the unconnected pins of cells in the open synthesized or implemented design. The command uses an internal process to identify whether a pin should be tied up or down.

This command is intended to tie up or down the unconnected pins of cells added to the netlist with the **create_cell** command.

Arguments

-of_objects *args* - (Optional) Currently ignored.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Example

The following example ties the unused pins of cells up or down, depending on their usage:

```
tie_unused_pins
```

See Also

- [create_cell](#)
- [create_pin](#)
- [get_cells](#)
- [get_pins](#)

undo

Un-do previous command.

Syntax

```
undo [-list] [-quiet] [-verbose]
```

Returns

With -list, the list of undoable tasks

Usage

Name	Description
[-list]	Show a list of undoable tasks
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[GUIControl](#)

Description

Undo a prior command. This command can be used repeatedly to undo a series of commands.

If a group of commands has been created using the **startgroup** and **endgroup** commands, this command will undo that group as a sequence. The undo command will start at the **endgroup** command and continue to undo until it hits the **startgroup** command.

If you **undo** a command, and then change your mind, you can **redo** the command.

Arguments

-list - (Optional) Reports the list of commands that can be undone. As you execute the undo command, the tool will step backward through this list of commands.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns a list of commands that you can undo:

```
undo -list
```

See Also

- [redo](#)
- [startgroup](#)
- [endgroup](#)

ungroup_bd_cells

Move the group of cells inside a hierarchy cell to its parent cell, and then remove the hierarchical cell. The connections between these cells are maintained; the connections between these cells and other cells are maintained through crossing hierarchy cell.

Syntax

```
ungroup_bd_cells [-prefix arg] [-quiet] [-verbose] [cells...]
```

Returns

0 if success

Usage

Name	Description
[-prefix]	Prefix name to add to cells
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<i>cells</i>]	Match engine names against cell names Default: *

Categories

[IPIntegrator](#)

unhighlight_objects

Unhighlight objects that are currently highlighted.

Syntax

```
unhighlight_objects [-color_index arg] [-rgb args] [-color arg]  
[-quiet] [-verbose] [objects]
```

Returns

Nothing

Usage

Name	Description
<code>[-color_index]</code>	Color index
<code>[-rgb]</code>	RGB color index list
<code>[-color]</code>	Valid values are red green blue magenta yellow cyan and orange
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[objects]</code>	Objects to unhighlight

Categories

[GUIControl](#)

Description

This command is for use in GUI mode. This command unhighlights the specified object or objects that were previously highlighted by the **highlight_objects** command.

This command supports the color options as specified below. These options can be used to unhighlight all objects currently highlighted in the specified color. See the example below.

Arguments

-color_index arg - (Optional) Specify the color index to unhighlight. The color index is defined by the Highlight category of the Tools > Options > Themes command. Refer to the *Vivado Design Suite User Guide: Using the IDE (UG893)* for more information on setting themes.

-rgb args - (Optional) Specify the color to unhighlight in the form of an RGB code specified as {R G B}. For instance, {255 255 0} specifies the color yellow.

-color arg - (Optional) Specify the color to unhighlight. Supported highlight colors are red, green, blue, magenta, yellow, cyan, and orange.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) Specifies one or more objects to be unhighlighted. If no objects are specified, all highlighted objects of the specified color will be unhighlighted. If no color is specified, all highlighted objects will be unhighlighted.

Examples

The following example unhighlights the selected objects:

```
unhighlight_objects [get_selected_objects]
```

The following example unhighlights all objects currently highlighted in the color yellow:

```
unhighlight_objects -color yellow
```

See Also

- [get_selected_objects](#)
- [highlight_objects](#)

unmark_objects

Unmark items that are currently marked.

Syntax

```
unmark_objects [-rgb args] [-color arg] [-quiet] [-verbose] [objects]
```

Returns

Nothing

Usage

Name	Description
[-rgb]	RGB color index list
[-color]	Valid values are red green blue magenta yellow cyan and orange
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[objects]	Objects to unmark

Categories

[GUIControl](#)

Description

Unmarks the specified object or objects that were previously marked by the **mark_objects** command. This command is for use in GUI mode.

This command supports the color options as specified below. However, these options are not necessary to unmark a specific object, but can be used to unmark all objects currently marked in the specified color. See the example below.

Arguments

-rgb args - (Optional) The color to unmark in the form of an RGB code specified as {R G B}. For instance, {255 255 0} specifies the color yellow.

-color arg - (Optional) The color to unmark. Supported highlight colors are red, green, blue, magenta, yellow, cyan, and orange.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) One or more objects to be unmarked. If no objects are specified, all marked objects of the specified color will be unmarked. If no color is specified, all marked objects will be unmarked.

Examples

The following example unmarks the selected objects:

```
unmark_objects [get_selected_objects]
```

The following example unmarks all objects currently marked in the color yellow:

```
unmark_objects -color yellow
```

See Also

- [get_selected_objects](#)
- [mark_objects](#)

unplace_cell

Unplace one or more instances.

Syntax

```
unplace_cell [-quiet] [-verbose] cell_list ...
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>cell_list</i>	a list of cells to be unplaced

Categories

[Floorplan](#)

Description

Unplace the specified cells from their current placement site.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

cell_list - (Required) Specifies a list of one or more cells to be unplaced from the device.

Examples

The following example unplaces the specified cell:

```
unplace_cell {fftEngine/fftInst/ingressLoop[6].ingressFifo/buffer_fifo/i_4773_12897}
```

The following example unplaces multiple cells:

```
unplace_cell {div_cntr_reg_inferredi_4810_15889 div_cntr_reg[0] div_cntr_reg[1]}
```

See Also

- [create_cell](#)
- [place_cell](#)
- [remove_cell](#)

unselect_objects

Unselect items that are currently selected.

Syntax

```
unselect_objects [-quiet] [-verbose] [objects]
```

Returns

Nothing

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[objects]</code>	Objects to unselect

Categories

[GUIControl](#)

Description

Unselects the specified object or objects that were previously selected by the **select_objects** command.

This command will unselect both primary and secondary selected objects. The selection of secondary objects is controlled through the use of Selection Rules defined in the **Tools > Options** command. Refer to the *Vivado Design Suite User Guide: Using the IDE (UG893)* for more information on Setting Selection Rules.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) One or more objects to be unselected. If no objects are specified, all selected objects will be unselected.

Examples

The following example unselects the specified site on the device:

```
unselect_objects [get_sites SLICE_X56Y214]
```

The following example unselects all currently selected objects:

```
unselect_objects
```

See Also

- [get_selected_objects](#)
- [select_objects](#)

update_compile_order

Updates a fileset compile order and possibly top based on a design graph.

Syntax

```
update_compile_order [-fileset arg] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-fileset]	Fileset to update based on a design graph
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Project

Description

Update the compile order of the design sources in the current project, or in the specified fileset.

Arguments

-fileset *arg* - Update the compile order of the specified fileset.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example updates the compile order of the source files in the simulation fileset:

```
update_compile_order -fileset sim_1
```

See Also

- [add_files](#)
- [import_files](#)

update_design

Update the netlist of the current design.

Syntax

```
update_design -cells args [-strict] [-from_file arg]  
[-from_design arg] [-from_cell arg] [-black_box] [-quiet]  
[-verbose]
```

Returns

Nothing

Usage

Name	Description
-cells	List of cells to update with a new sub-netlist.
[-strict]	Require exact ports match for replacing cell (otherwise extra ports are allowed).
[-from_file]	Name of the file containing the new sub-netlist.
[-from_design]	Name of the an open netlist design containing the new sub-netlist.
[-from_cell]	Name of cell in the from_design which defines the new sub-netlist.
[-black_box]	Update the cell to a black_box.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Project](#)

Description

This command updates the in-memory design, replacing the current netlist in the specified cells with a netlist from a specified file, from another open design, from a specified cell of a design, or convert the cells to a black box cell. The command can update a single instance, or a list of instances, or can update all instances of a master cell.

Only the in-memory view of the design is changed by the new netlist. You must save the design using the **write_checkpoint** command, or any updates will be lost when you close the project or exit the tool.

Arguments

-cells *args* - (Required) Defines a list of hierarchical instance names, or cell objects, to update with the specified netlist. To update all instances of a cell, use the **get_cells** command with the **-filter** argument to specify the master cell:

```
get_cells -hier -filter {ref_name==aMasterCellName} <cell_name>
```

-strict - (Optional) Require the new netlist to have exactly the same ports as cells it is imported into. The tool will perform some checking on the new netlist to insure that the specified netlist has all the ports required for the specified cells. However, additional ports are also permitted, unless the **-strict** option is used.

-from_file - (Optional) Name of a file containing the new netlist. The netlist can be in the form of a structured Verilog netlist (.v) or an EDIF netlist (.edf) file.

Note **-from_file** and **-from_design** are mutually exclusive

-from_design - (Optional) Allows you to import the netlist from another open design in the current project. The design must be opened in the current tool invocation, and not a separate process.

-from_cell - (Optional) Name of a cell in the design specified with **-from_design**. The netlist from the specified cell will be used to update the cells in the current design. By default the tool will use the top-level cell of the design specified in **-from_design**.

Note This option can only be used with **-from_design**.

-black_box - (Optional) Update the specified cells to a black box cell.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example updates the netlist in the arnd4 cell with the specified Verilog netlist:

```
update_design -cells arnd4 -from_file C:/Data/round_4.v
```

The following example updates the arnd4 cell in the current design with the netlist from the same cell in the specified design:

```
update_design -cells arnd4 -from_design netlist_2 -from_cell arnd4
```

See Also

- [get_cells](#)
- [write_checkpoint](#)

update_files

Update files in the project with same named files from the files or directories specified.

Syntax

```
update_files [-from_files args] [-norecurse] [-to_files args]  
[-filesets args] [-force] [-report_only] [-quiet] [-verbose]
```

Returns

List of the files updated

Usage

Name	Description
<code>[-from_files]</code>	New files and directories to use for updating
<code>[-norecurse]</code>	Recursively search in specified directories
<code>[-to_files]</code>	Existing project files and directories to limit updates to
<code>[-filesets]</code>	Fileset name
<code>[-force]</code>	Overwrite imported files in the project, even if read-only, if possible
<code>[-report_only]</code>	Do no actual file updates, but report on updates that otherwise would have been made
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

Categories

[Project](#)

update_ip_catalog

Update the IP Catalog. Before executing this command optionally use the following to set repository paths: 'set_property ip_repo_paths [current_fileset]'.

Syntax

```
update_ip_catalog [-rebuild] [-add_ip arg] [-delete_ip arg]
-delete_mult_ip args [-repo_path arg] [-quiet] [-verbose]
```

Returns

True for success

Usage

Name	Description
[-rebuild]	Trigger a rebuild of the specified repository's index file or rebuild all repositories if none specified
[-add_ip]	Add the specified IP into the specified repository Values: Either a path to the IP's component.xml or to a zip file containing the IP
[-delete_ip]	Remove the specified IP from the specified repository Values: Either a path to the IP's component.xml or its VLNV
-delete_mult_ip	Remove the specified IPs from the specified repository Values: A list of IPs; either paths to the component.xml files or their VLNVs
[-repo_path]	Used in conjunction with rebuild, add_ip, delete_ip or delete_mult_ip to specify the path of the repository on which to operate
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[IPFlow](#)

Description

Update the IP Catalog associated with the current design.

The default IP catalog is found in the installation hierarchy of the tool. However, you can also add new IP repository paths to the IP catalog using the **set_property** command to define the IP_REPO_PATHS property of the current Source fileset.

The Xilinx IP catalog, or repository, is located in the installation hierarchy of the software release being used. You can also add custom IP to the repository by using the **set_property** command to set the IP_REPO_PATHS property on the source fileset to point to the locations of custom IP, as shown in the example below.

Arguments

-rebuild - (Optional) Rebuild the complete IP Catalog index, or just rebuild the index for the IP repository specified by the **-repo_path**.

-add_ip arg - (Optional) Add an individual IP core to the specified IP repository. This argument requires the **-repo_path** argument to also be specified. The IP is specified as a path to the `component.xml` of the IP, or the path to a zip file containing the IP.

-delete_ip arg - (Optional) Remove an IP core from the specified IP repository. This argument requires the **-repo_path** argument to also be specified. The IP is specified as a path to the `component.xml` of the IP, or as the VLNV property of the IP. The VLNV property refers to the Vendor:Library:Name:Version string which identifies the IP in the catalog.

-repo_path arg - (Optional) Specify the directory name of an IP repository to add to or delete from, or to rebuild the index for.

Note The IP repository must have been previously added to the current Source fileset using the **set_property** command. See the example below

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example sets the IP_REPO_PATHS property of the current Source fileset, to add an IP repository, then rebuilds the IP catalog index for the whole IP catalog:

```
set_property ip_repo_paths C:/Data/IP_LIB [current_fileset]
update_ip_catalog -rebuild
```

See Also

- [create_ip](#)
- [import_ip](#)
- [generate_target](#)
- [validate_ip](#)

update_macro

Update a macro.

Syntax

```
update_macro [-absolute_grid] [-quiet] [-verbose] macro rlocs
```

Returns

Nothing

Usage

Name	Description
<code>[-absolute_grid]</code>	Use absolute grid for relative locations
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>macro</code>	Macro to update
<code>rlocs</code>	a list interleaved instances and site names

Categories

XDC

Description

Populate a previously created macro with leaf cells and relative placements.

A macro is made up of primitive, or leaf-level logic cells, and their associated connections, positioned in a placement grid. The specified relative locations, or *rlocs*, can be based on a relative grid, or on an absolute grid, called an RPM_GRID. Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on absolute and relative placement grids

A cell can only belong to a one macro. If you attempt to assign a leaf-level cell to multiple macros, the Vivado tool will return an error. If you attempt to assign a non-primitive cell to a macro, the tool will return an error.

To change the contents of an existing macro, you must delete the macro with **delete_macro**, recreate it with **create_macro**, and update it with new contents. You cannot simply overwrite or modify an existing macro.

Arguments

-absolute_grid - (Optional) Use **-absolute_grid** to indicate that the *rlocs* are specified in the absolute RPM_GRID.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

macro - (Required) Specify the name of the macro to update.

rlocs - (Required) Specify the leaf-cells to include in the macro, and their relative locations (RLOCs). When **-absolute_grid** is specified, the location is based on actual device coordinates instead of relative locations. The cells and RLOCs are specified as name-value pairs, with multiple leaf-cells and RLOCs assigned to a single macro as follows:

```
{cell10 XmYn cell11 XmYn ... cell1N XmYn}
```

Where:

- m = An integer representing the relative or absolute X coordinate of the preceding cell.
- n = An integer representing the relative or absolute Y coordinate of the preceding cell.
- Cell coordinates are relative to each other, unless the **-absolute_grid** option has been specified.
- The relative coordinates are based on a theoretical array of Slices, located relative to each other.
- The absolute coordinates are determined by the RPM_X and RPM_Y properties from actual Slices of the target device. These can be determined by the **report_property** command for selected sites.

Examples

The following example creates a macro named usbMacro0, sets the current instance to the usbEngine0/u0 module, assigns three cells to the macro, with a relative placement for each cell to have two of them placed inside the same Slice, and the third placed in a vertically adjacent Slice:

```
create_macro usbMacro0
current_instance usbEngine0/u0
update_macro usbMacro0 {rx_active_reg X0Y0 rx_err_reg X0Y0 rx_valid_reg X0Y1}
```

The following example creates a macro named `usbMacro1`, assigns three cells to the macro using the hierarchical path to the leaf-cell, with absolute coordinates specified for the cells in the macro:

```
create_macro usbMacro1
set Site1 [get_sites SLICE_X8Y77]
set Site2 [get_sites SLICE_X9Y77]
set Site3 [get_sites SLICE_X8Y78]
set RPM1 X[get_property RPM_X $Site1]Y[get_property RPM_Y $Site1]
set RPM2 X[get_property RPM_X $Site2]Y[get_property RPM_Y $Site2]
set RPM3 X[get_property RPM_X $Site3]Y[get_property RPM_Y $Site3]
update_macro usbMacro1 -absolute_grid "usbEngine1/u0/rx_active_reg $RPM1 \
usbEngine1/u0/rx_err_reg $RPM2 usbEngine1/u0/rx_valid_reg $RPM3"
```

Note In the prior example, notice the use of Tcl variables to capture the Sites of interest, and extract the `RPM_X` and `RPM_Y` properties of those sites for use in the **update_macro** command. Also notice the use of quotes (") instead of curly braces ({} in the **update_macro** command. This is to allow the Tcl shell to perform variable substitution of the command. Refer to the *Vivado Design Suite User Guide: Using Tcl Scripting (UG894)* for more information on variables and variable substitution

This command reports the properties on the `usbMacro1` macro to see the absolute grid coordinates assigned to the cells in the macro:

```
report_property -all [get_macros usbMacro1]
```

See Also

- [create_macro](#)
- [delete_macros](#)
- [get_macros](#)
- [get_property](#)
- [get_sites](#)
- [place_design](#)
- [report_property](#)

update_timing

Update timing.

Syntax

```
update_timing [-full] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-full]	Perform a full timing update instead of an incremental one
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Timing](#)

upgrade_ip

Upgrade a configurable IP to a later version.

Syntax

```
upgrade_ip [-srcset arg] [-latest arg] [-vlnv arg] [-quiet] [-verbose]  
[objects...]
```

Returns

List of files that were upgraded

Usage

Name	Description
<code>[-srcset]</code>	Source set name
<code>[-latest]</code>	Upgrade the IP to the latest version
<code>[-vlnv]</code>	VLNV string for the Catalog IP to which the IP will be upgraded
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[objects]</code>	IP to be upgraded

Categories

[IPFlow](#)

Description

This command upgrades the specified IP cores from an older version to the latest version in the IP catalog.

You can only upgrade legacy IP that explicitly supports upgrading. The `UPGRADE_VERSIONS` property on the `ipdef` object indicates if there are upgrade version for an IP core.

Note Not all legacy IP support upgrading, and native IP cannot be upgraded

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

objects - (Optional) Specifies which legacy IP cores should be upgraded.

Examples

The following example upgrades all IP cores in the current project to the latest version:

```
upgrade_ip
```

See Also

- [create_ip](#)
- [import_ip](#)

upload_hw_ila_data

Stop capturing. Upload any captured hardware ILA data.

Syntax

```
upload_hw_ila_data [-quiet] [-verbose] [hw_ilas...]
```

Returns

Hardware ILA data objects

Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[hw_ilas]</code>	List of hardware ILA objects. Default: Current hardware ILA

Categories

[Hardware](#)

validate_bd_design

Run Parameter Propagation for specified design or for a specific cell.

Syntax

```
validate_bd_design [-design arg] [-cell arg] [-quiet] [-verbose]
```

Returns

TCL_OK, TCL_ERROR if failed

Usage

Name	Description
[-design]	Design name. If not specified, run parameter propagation on current design
[-cell]	Cell name. If not specified, run parameter propagation on whole design
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[IPIntegrator](#)

validate_ip

This command applies any pending set_property commands and returns parameter validation messages, if any exist.

Syntax

```
validate_ip [-save_ip] [-quiet] [-verbose] [ips]
```

Returns

Nothing

Usage

Name	Description
[-save_ip]	Write IP files on the disk
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[ips]</i>	IPs to be validated

Categories

IPFlow

Description

Perform DRC check on IP to ensure that it is properly constructed. This command returns 1 when all IPs have been validated, or 0 when there is a problem.

Arguments

-save_ip - (Optional) Updates the existing IP files after validation. No new files are created but the XCI and BOM files for the core are updated.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

ips - (Optional) Specifies the set of IP cores to be validated.

Examples

The following example validates the IPs in the current project, and updates the persistent representation of the IP.

```
validate_ip -save_ip [get_ips]
```

See Also

- [create_ip](#)
- [generate_target](#)
- [upgrade_ip](#)
- [update_ip_catalog](#)
- [import_ip](#)

version

Returns the build for Vivado and the build date.

Syntax

```
version [-short] [-quiet] [-verbose]
```

Returns

Vivado version

Usage

Name	Description
[-short]	Return only the numeric version number
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

[Report](#)

Description

Returns the version number of the Xilinx® tool. This includes the software version number, build number and date, and copyright information.

Arguments

-short - (Optional) Returns the version number of the software only.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example returns only the version number for the software:

```
version -short
```

wait_on_hw_ila

Wait until all hardware ILA data has been captured.

Syntax

```
wait_on_hw_ila [-timeout arg] [-quiet] [-verbose] [hw_ilas...]
```

Returns

Nothing

Usage

Name	Description
[-timeout]	Timeout in minutes. Decimal value allowed Default: No timeout
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>[hw_ilas]</i>	hardware ILA objects. Default: Current hardware ILA

Categories

[Hardware](#)

wait_on_hw_sio_scan

Wait until hardware VIO scan has completed.

Syntax

```
wait_on_hw_sio_scan [-timeout arg] [-quiet] [-verbose] hw_sio_scans...
```

Returns

Nothing

Usage

Name	Description
[-timeout]	Timeout in minutes. Decimal value allowed Default: No timeout
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>hw_sio_scans</i>	List of hardware SIO scan objects.

Categories

[Hardware](#)

wait_on_run

Block execution of further Tcl commands until the specified run completes.

Syntax

```
wait_on_run [-timeout arg] [-quiet] [-verbose] run
```

Returns

Nothing

Usage

Name	Description
[-timeout]	Maximum time to wait for the run to complete (in minutes) Default: -1
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>run</i>	Run to wait on

Categories

[Project](#)

Description

Blocks the execution of Tcl commands until the specified run completes, or until the specified amount of time has elapsed.

The **wait_on_run** command can be used for runs that have been launched. If the specified run has not been launched when the **wait_on_run** command is used, you will get an error. Runs that have already completed do not return an error.

Note This command is used for running the tool in batch mode or from Tcl scripts. It is ignored when running interactively from the GUI.

Arguments

-timeout *arg* - (Optional) The time in minutes that the **wait_on_run** command should wait until the run finishes. This allows you to define a period of time beyond which the tool should resume executing Tcl commands even if the specified run has not finished execution. The default value of -1 is used if timeout is not specified, meaning that there is no limit to the amount of time the tool will wait for the run to complete.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

run - (Required) The name of the run to wait on.

Examples

The following example launches the impl_1 run, and then waits for the specified run to complete, or to wait for one hour, whichever occurs first:

```
launch_runs impl_1  
wait_on_run -timeout 60 impl_1
```

See Also

[launch_runs](#)

write_bd_tcl

Export the current design to a Tcl file on disk.

Syntax

```
write_bd_tcl [-quiet] [-verbose] name
```

Returns

TCL_OK, TCL_ERROR if failed

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>name</i>	Name exported Tcl file

Categories

[IPIntegrator](#)

write_bitstream

Write a bitstream for the current design.

Syntax

```
write_bitstream [-force] [-raw_bitfile] [-no_binary_bitfile]
[-mask_file] [-logic_location_file] [-bin_file]
[-reference_bitfile arg] [-quiet] [-verbose] file
```

Returns

Nothing

Usage

Name	Description
[-force]	Overwrite existing file
[-raw_bitfile]	Write raw bit file (.rbt)
[-no_binary_bitfile]	Do not write binary bit file (.bit)
[-mask_file]	Write mask file (.msk)
[-logic_location_file]	Write logic location file (.ll)
[-bin_file]	Write binary bit file without header (.bin)
[-reference_bitfile]	Reference bitfile to be used for generating partial bitstream
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	The name of the .bit file to generate

Categories

[FileIO](#)

Description

Writes a bitstream file for the current project. This command must be run on an Implemented Design. The bitstream written will be based on the open Implemented Design.

```
write_bitstream [-force] [-raw_bitfile] [-no_binary_bitfile] [-mask_file] [-logic_location_file]
[-bin_file] [-reference_bitfile <arg>] [-quiet] [-verbose] <file>
```

Arguments

-force - (Optional) Force the overwrite of an existing bitstream file of the same name.

-raw_bitfile - (Optional) Write a raw bit file (.rbt) which contains the same information as the binary bitstream file, but is in ASCII format. The output file will be named *<file>.rbt*.

-no_binary_bitfile - (Optional) Do not write the binary bitstream file (.bit). Use this command when you want to generate the ASCII bitstream or mask file, or to generate a bitstream report, without also generating the binary bitstream file.

-mask_file - (Optional) Write a mask file (.msk), which has mask data where the configuration data is in the bitstream file. This file determines which bits in the bitstream should be compared to readback data for verification purposes. If a mask bit is 0, that bit should be verified against the bitstream data. If a mask bit is 1, that bit should not be verified. The output file will be named *<file>.msk*.

-logic_location_file - (Optional) Creates an ASCII logic location file (.ll) that shows the bitstream position of latches, flip-flops, LUTs, Block RAMs, and I/O block inputs and outputs. Bits are referenced by frame and bit number in the location file to help you observe the contents of FPGA registers.

-bin_file - (Optional) Creates a binary file (.bin) containing only device programming data, without the header information found in the standard bitstream file (.bit).

-reference_bitfile *<arg>* - (Optional) Read a reference bitstream file, and output an incremental bitstream file containing only the differences from the specified reference file. This partial bitstream file can be used for incrementally programming an existing device with an updated design.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The name of the bitstream file (.bit) to write. If you do not specify a file extension, the .bit extension will be added by the tool, but you cannot specify an extension other than .bit.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example writes a bitstream file of the specified name:

```
write_bitstream design1.bit
```

The following example writes both the binary and ASCII forms of the bitstream:

```
write_bitstream -raw_bitfile C:/Data/design
```

Note The appropriate file extension will be added by the tool

See Also

[launch_runs](#)

write_bmm

Write a bmm file.

Syntax

```
write_bmm [-force] [-quiet] [-verbose] file
```

Returns

The name of the bmm file

Usage

Name	Description
[-force]	Overwrite existing checkpoint file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Design bmm file Values: A filename with alphanumeric characters and .bmm extension.

Categories

[FileIO](#)

write_checkpoint

Write a checkpoint of the current design.

Syntax

```
write_checkpoint [-force] [-quiet] [-verbose] file
```

Returns

The name of the checkpoint file

Usage

Name	Description
[-force]	Overwrite existing checkpoint file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Design checkpoint file Values: A filename with alphanumeric characters and .dcp extension.

Categories

[FileIO](#)

Description

Saves the design at any point in the design process so that you can quickly import it back into the tool as needed. A design checkpoint (DCP) can contain the netlist, the constraints, and any placement and routing information from the implemented design.

Use the **read_checkpoint** command to import a checkpoint file.

Arguments

- force** - (Optional) Overwrite an existing checkpoint file of the same name if it already exists.
- quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.
- verbose** - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The name of the checkpoint file to be created. A .dcp extension will be added if no extension is specified.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example creates the specified checkpoint file, overwriting a file of the same name if one already exists:

```
write_checkpoint C:/Data/checkpoint1 -force
```

Note The tool will add the .dcp extension to the specified file name, and will overwrite an existing `checkpoint1.dcp` file

See Also

[read_checkpoint](#)

write_chipscope_cdc

Export nets that are connected to debug ports.

Syntax

```
write_chipscope_cdc [-quiet] [-verbose] file
```

Returns

Name of the output file

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	ChipScope CDC file name

Categories

[FileIO](#), [ChipScope](#)

Description

Writes a ChipScope Definition and Connection (CDC) file containing the debug cores, ports, and signals defined in the current project.

ChipScope debug cores are added to a project through the use of the **create_debug_core** command. The CDC file stores information about source files, destination files, core parameters, and core settings for the ChipScope Pro Analyzer.

You can import this CDC file into the ChipScope Analyzer to automatically set up the net names on the ILA core data and trigger ports.

Arguments

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The CDC file name to be written.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example writes a CDC file called bft.cdc:

```
write_chipscope_cdc bft.cdc
```

The written CDC file will include signals to be debugged by ChipScope as well as the clock domain for the signals, and other settings appropriate for use in ChipScope.

See Also

[create_debug_core](#)

write_csv

Export package pin and port placement information.

Syntax

```
write_csv [-force] [-quiet] [-verbose] file
```

Returns

Name of the output file

Usage

Name	Description
[-force]	Overwrite existing file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Pin Planning CSV file

Categories

[FileIO](#)

Description

Writes package pin and port placement information into a comma separated value (CSV) file.

The specific format and requirements of the CSV file are described in the *Vivado Design Suite User Guide: I/O and Clock Planning (UG899)*.

Arguments

-force - (Optional) Overwrite the CSV file if it already exists.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The filename of the CSV file to write.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example exports a CSV file from the current project:

```
write_csv C:/Data/pinList.csv
```

See Also

[read_csv](#)

write_debug_probes

Write debug probes to a file.

Syntax

```
write_debug_probes [-force] [-quiet] [-verbose] file
```

Returns

Name of the output file

Usage

Name	Description
[-force]	Overwrite existing file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Debug probes file name (default extension is .ltx)

Categories

[FileIO](#)

Description

Writes an ILA probes file that contains information about the nets that you probed in the current design using the ILA v2.0 core. The debug probes data file typically has a .ltx file extension.

ILA cores are added to the design using the **create_debug_core** command, and connected to nets in your design using the **connect_debug_core** command.

The specific information and use of the debug probes file is described in the *Vivado Design Suite User Guide: Programming and Debugging (UG908)*.

Arguments

-force - (Optional) Overwrite the specified file if it already exists.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The file name of the debug probes file to write.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example write a debug probe file from the current design:

```
write_debug_probes C:/Data/designProbes.ltx
```

See Also

- [create_debug_core](#)
- [implement_debug_core](#)

write_edif

Export the current netlist as an EDIF file.

Syntax

```
write_edif [-pblocks args] [-cell arg] [-force] [-security_mode arg]
[-quiet] [-verbose] file
```

Returns

The name of the output file or directory

Usage

Name	Description
[-pblocks]	Export netlist for these pblocks (not valid with -cell)
[-cell]	Export netlist for this cell (not valid with -pblocks)
[-force]	Overwrite existing file
[-security_mode]	If set to 'all', and some of design needs encryption then whole of design will be written to a single encrypted file Default: multifile
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Output file (directory with -pblocks or -cell)

Categories

[FileIO](#)

Description

Writes the current netlist as an EDIF file, or outputs the contents of specific Pblocks or hierarchical cells as EDIF netlist files.

In the case of either the -pblock or -cell argument being used, this argument specifies a directory name where the EDIF netlist files for each Pblock or cell will be written. The EDIF netlist file will be named after the Pblock or cell. If the directory specified does not exist, the tool will return an error.

Arguments

-pblocks arg - (Optional) Instructs the tool to output the contents of the specified Pblocks as EDIF netlist files. The contents of each Pblock will be written to a separate EDIF file.

-cell *arg* - (Optional) Instructs the tool to output the contents of the specified hierarchical cell as EDIF netlist files. Only one cell can be specified for output.

Note The **-pblock** and **-cell** arguments are mutually exclusive. Although they are optional arguments, only one may be specified at one time.

-force - (Optional) Overwrite the EDIF file if it already exists.

-security_mode [multifile | all] - (Optional) Write a multiple EDIF files when encrypted IP is found in the design, or write a single file.

- **multifile** - This is the default setting. By default the command writes out the full design netlist to the specified file. However, if the design contains secured IP, it creates an encrypted file containing the contents of the secured module. This will result in the output of multiple EDIF files, containing secured and unsecured elements of the design.
- **all** - Write both encrypted and unencrypted cells to a single specified file.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The filename of the EDIF file to write. The default file extension for an EDIF netlist is .edn. If the **-pblocks** or **-cell** options are used, the name specified here refers to a directory rather than a single file.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example writes an EDIF netlist file for the whole design to the specified file name:

```
write_edif C:/Data/edifOut.edn
```

The following example outputs an EDIF netlist for all Pblocks in the design. The files will be written to the specified directory.

```
write_edif -pblocks [get_pblocks] C:/Data/FPGA_Design/
```

See Also

[read_edif](#)

write_hw_ila_data

Write hardware ILA data to a file.

Syntax

```
write_hw_ila_data [-force] [-quiet] [-verbose] file [hw_ila_data]  
[hw_ila_data]
```

Returns

Name of the output file

Usage

Name	Description
[-force]	Overwrite existing file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	hardware ILA data file name
[<i>hw_ila_data</i>]	hardware ILA data object Default: Current hardware ILA data

Categories

[Hardware](#)

write_hw_sio_scan

Write scan data to a file.

Syntax

```
write_hw_sio_scan [-force] [-quiet] [-verbose] file hw_sio_scan
```

Returns

Name of the output file

Usage

Name	Description
[-force]	Overwrite existing file
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	hardware SIO_scan file name
<i>hw_sio_scan</i>	hardware SIO scan data object

Categories

[Hardware](#)

write_ibis

Write IBIS models for current floorplan.

Syntax

```
write_ibis [-force] [-allmodels] [-nopin] [-truncate arg] [-ibs arg]  
[-pkg arg] [-quiet] [-verbose] file
```

Returns

Name of the output file

Usage

Name	Description
[-force]	Overwrite existing .ibs file
[-allmodels]	Include all available buffer models for this architecture. By default, only buffer models used by the floorplan are included.

Categories

[FileIO](#)

Description

Writes the IBIS models for the target device in the current design. The netlist and implementation details from the design are combined with the per-pin parasitic package information to create a custom IBIS model for the design.

Because the write_ibis command incorporates design information into the IBIS Model, you must have an RTL, Netlist, or Implemented Design open when running this command.

Arguments

-allmodels - (Optional) Export all buffer models for the target device. By default the tool will only write buffer models used by the design.

-nopin - (Optional) Disable per-pin modeling of the path from the die pad to the package pin. The IBIS model will include a single RLC transmission line model representation for all pins in the [Package] section. By default the file will include per-pin modeling of the package as RLC matrices in the [Define Package Model] section if this data is available.

-truncate *arg* - (Optional) The maximum length for a signal name in the output file. Names longer than this will be truncated. Valid values are 20, 40, or 0 (unlimited). By default the signal names are truncated to 40 characters in accordance with the IBIS version 4.2 specification.

-ibs *arg* - (Optional) Specify an updated generic IBIS models file. This is used to override the IBIS models found in the tool installation under the parts directory. This argument is required for any parts that do not have generic models in the installation directory.

-pkg *arg* - (Optional) Specify an updated per pin parasitic package data file. This is used to override the parasitic package file found in the the tool installation hierarchy under the parts directory. This argument is required for any parts that do not have generic models in the installation directory.

file - (Required) The filename of the IBIS file to write.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

Examples

The following example exports all buffer models for the target device, eliminates truncation of signal names, and specifies the file name and path to write:

```
write_ibis -allmodels -truncate 0 C:/Data/FPGA_Design/ibisOut.txt
```

write_project_tcl

(User-written application).

Syntax

```
::write_project_tcl [-force arg] [-all_properties arg]  
[-no_copy_sources arg] [-dump_project_info arg] [-quiet]  
[-verbose] file
```

Returns

True

Usage

Name	Description
[-force]	Overwrite existing tcl script file Default: 0
[-all_properties]	write all properties (default & non-default) for the project object(s) Default: 0
[-no_copy_sources]	Do not import sources even if they were local in the original project Default: 1
[-dump_project_info]	Write object values Default: 0
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Name of the tcl script file to generate

Description

Creates a Tcl script to recreate the current project.

The generated script will contain the Tcl commands for creating the project, setting the project type, creating filesets, adding/importing source files, defining runs and run properties.

This Tcl project script and the various design sources can be stored in a version control system for source file management and project archival.

Arguments

-force - (Optional) Overwrite an existing project script file of the same name. If the script file already exists, the tool returns an error unless the **-force** argument is specified.

-all_properties - (Optional) Write all properties (default and non-default) for the project. The tool will write **set_property** commands for all the objects like project, filesets, files, runs etc.

Note By default, if the **-all_properties** switch is not specified, only non-default properties will be written to the script.

-no_copy_sources - (Optional) Do not import sources even if they are local to the original project. The tool will not import the files that were local in the original project into the new project.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The name of the script file to be created by the **write_project_tcl** command. The tool will apply an extension of `.tcl` if a file extension is not supplied.

Examples

The following example exports Tcl script named "recreate.tcl" for the current project:

```
write_project_tcl recreate.tcl
```

The following command exports Tcl script for the current project and writes all the properties, both default or non-default values:

```
write_project_tcl -all_properties recreate.tcl
```

The following command exports Tcl script for the current project and adds files that are local in this project. The recreated project will reference these files:

```
write_project_tcl -no_copy_sources recreate.tcl
```

The following command exports "recreate.tcl" script for the current project in the current working directory, creates a new project in `./my_test` directory, prints the list of files in the new project, prints the current project settings and then closes the newly created project:

```
open_project ./test/test.xpr
write_project_tcl -force recreate.tcl
close_project
file mkdir my_test
cd my_test
source ../recreate.tcl
get_files -of_objects [get_filesets sources_1]
report_property [current_project]
close_project
```

The following command creates a new project named `bft_test`, adds files to the project, sets the `fileset` property, exports a tcl script named `"bft.tcl"` in the current working directory, creates a new project in `"./my_bft"` directory, prints the list of files in the new project (`test_1.v` and `test_2.v`), prints the `"verilog_define"` property value and then closes the newly created project:

```
create_project bft_test ./bft_test
add_files test_1.v
add_files test_2.v
set_property verilog_define {a=10} [get_filesets sources_1]
write_project_tcl -force bft.tcl
close_project
file mkdir my_bft
cd my_bft
source ../bft.tcl
get_files -of_objects [get_filesets sources_1]
get_property verilog_define [get_filesets sources_1]
close_project
```

See Also

- [add_files](#)
- [archive_project](#)
- [close_project](#)
- [create_project](#)
- [current_project](#)
- [get_files](#)
- [get_property](#)
- [open_project](#)
- [report_property](#)
- [set_property](#)

write_sdf

Write_sdf command generates flat sdf delay files for event simulation.

Syntax

```
write_sdf [-process_corner arg] [-cell arg] [-rename_top arg] [-force]  
[-mode arg] [-quiet] [-verbose] file
```

Returns

Nothing

Usage

Name	Description
[-process_corner]	Specify process corner for which SDF delays are required; Values: slow, fast Default: slow
[-cell]	Root of the design to write, e.g. des.subblk.cpu Default: whole design
[-rename_top]	Replace name of top module with custom name e.g. netlist Default: new top module name
[-force]	Overwrite existing SDF file
[-mode]	Specify sta (Static Timing Analysis) or timesim (Timing Simulation) mode for SDF Default: timesim
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	File name

Categories

[FileIO](#), [Simulation](#)

Description

Writes the timing delays for cells in the design to a Standard Delay Format (SDF) file.

The output SDF file can be used by the **write_verilog** command to create Verilog netlists for static timing analysis and timing simulation.

Arguments

-process_corner [fast | slow] - (Optional) Write delays for a specified process corner. Delays are greater in the slow process corner than in the fast process corner. Valid values are 'slow' or 'fast'. By default, the SDF file is written for the slow process corner.

-cell *arg* - (Optional) Write the SDF file from a specific cell of the design hierarchy. The default is to create an SDF file for the whole design.

-rename_top *arg* - (Optional) Rename the top module in the output SDF file as specified.

-force - (Optional) Forces the overwrite of an existing SDF file of the same name.

-mode [timesim | sta] - (Optional) Specifies the mode to use when writing the SDF file. Valid values are:

- timesim - Output an SDF file to be used for timing simulation. This is the default setting.
- sta - Output an SDF file to be used for static timing analysis (STA).

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The file name of the SDF file to write. The SDF file is referenced in the Verilog netlist by the use of the **-sdf_anno** and **-sdf_file** arguments of the **write_verilog** command.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example writes an SDF file to the specified directory:

```
write_sdf C:/Data/FPGA_Design/designOut.sdf
```

See Also

[write_verilog](#)

write_verilog

Export the current netlist in Verilog format.

Syntax

```
write_verilog [-cell arg] [-mode arg] [-lib] [-port_diff_buffers]
[-write_all_overrides] [-rename_top arg] [-sdf_anno arg]
[-sdf_file arg] [-force] [-include_xilinx_libs] [-quiet]
[-verbose] file
```

Returns

The name of the output file or directory

Usage

Name	Description
[-cell]	Root of the design to write, e.g. des.subblk.cpu Default: whole design
[-mode]	Values: design, port, sta, funcsim, timesim Default: design
[-lib]	Write each library into a separate file
[-port_diff_buffers]	Output differential buffers when writing in -port mode
[-write_all_overrides]	Write parameter overrides on Xilinx primitives even if the override value is the same as the default value
[-rename_top]	Replace top module name with custom name e.g. netlist Default: new top module name
[-sdf_anno]	Specify if sdf_annotate system task statement is generated
[-sdf_file]	Full path to sdf file location Default: .sdf
[-force]	Overwrite existing file
[-include_xilinx_libs]	Include simulation models directly in netlist instead of linking to library
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Which file to write

Categories

[FileIO](#), [Simulation](#)

Description

Write a Verilog netlist of the current design or from a specific cell of the design to the specified file or directory. The output is a IEEE 1364-2001 compliant Verilog HDL file that contains netlist information obtained from the input design files.

You can output a complete netlist of the design or specific cell, or output a port list for the design, or a Verilog netlist for simulation or static timing analysis.

Arguments

-cell *arg* - (Optional) Write the Verilog netlist from a specified cell or block level of the design hierarchy. The output Verilog file or files will only include information contained within the specified cell or module.

-mode *arg* - (Optional) The mode to use when writing the Verilog file. By default, the Verilog netlist is written for the whole design. Valid mode values are:

- **design** - Output a Verilog netlist for the whole design. This acts as a snapshot of the design, including all post placement, implementation, and routing information in the netlist.
- **port** - Outputs only the I/O ports for the top-level of the design.
- **sta** - Output a Verilog netlist to be used for static timing analysis (STA).
- **funcsim** - Output a Verilog netlist to be used for functional simulation. The output netlist is not suitable for synthesis.
- **timesim** - Output a Verilog netlist to be used for timing simulation. The output netlist is not suitable for synthesis.

-lib - (Optional) Create a separate Verilog file for each library used by the design.

Note This option is the opposite of, and replaces the **-nolib** option from prior releases. Previously the default behavior of **write_verilog** was to output a separate Verilog file for each library used in the design, unless **-nolib** was specified. Now you must specify the **-lib** option to output separate Verilog files for each library

-port_diff_buffers - (Optional) Add the differential pair buffers and internal wires associated with those buffers into the output ports list. This argument is only valid when **-mode port** is specified.

-write_all_overrides [true | false] - (Optional) Write parameter overrides, in the design to the Verilog output even if the value of the parameter is the same as the defined primitive default value. If the option is false then parameter values which are equivalent to the primitive defaults are not output to the Verilog file. Setting this option to true will not change the result but makes the output Verilog more verbose.

-rename_top *arg* - (Optional) Rename the top module in the output as specified. This option only works with **-mode funcsim** or **-mode timesim** to allow the Verilog netlist to plug into top-level simulation test benches.

-sdf_anno [true | false] - (Optional) Add the **\$sdf_annotate** statement to the Verilog netlist. Valid values are true (or 1) and false (or 0). This option only works with **-mode timesim**, and is set to false by default.

-sdf_file *arg* - (Optional) The path and filename of the SDF file to use when writing the `$sdf_annotate` statement into the output Verilog file. When not specified, the SDF file is assumed to have the same name and path as the Verilog output specified by `<file>`, with a file extension of `.sdf`. The SDF file must be separately written to the specified file path and name using the **write_sdf** command.

-force - (Optional) Overwrite the Verilog files if they already exists.

-include_xilinx_libs - (Optional) Write the simulation models directly in the output netlist file rather than pointing to the libraries by reference.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The path and filename of the Verilog file to write. The path is optional, but if one is not provided the Verilog file will be written to the current working directory, or the directory from which the Vivado tool was launched.

Examples

The following example writes a Verilog simulation netlist file for the whole design to the specified file and path:

```
write_verilog C:/Data/my_verilog.v
```

In the following example, because the **-mode timesim** and **-sdf_anno** options are specified, the **\$sdf_annotate** statement will be added to the Verilog netlist. However, since the **-sdf_file** option is not specified, the SDF file is assumed to have the same name as the Verilog output file, with an `.sdf` file extension:

```
write_verilog C:/Data/my_verilog.net -mode timesim -sdf_anno true
```

Note The SDF filename written to the **\$sdf_annotate** statement will be `my_verilog.sdf`

See Also

- [write_sdf](#)
- [write_vhdl](#)

write_vhdl

Export the current netlist in VHDL format.

Syntax

```
write_vhdl [-cell arg] [-mode arg] [-lib] [-port_diff_buffers]
[-write_all_overrides] [-rename_top arg] [-arch_only] [-force]
[-include_xilinx_libs] [-quiet] [-verbose] file
```

Returns

The name of the output file or directory

Usage

Name	Description
[-cell]	Root of the design to write, e.g. des.subblk.cpu Default: whole design
[-mode]	Output mode. Valid values: funcsim, port Default: funcsim
[-lib]	Write each library into a separate file
[-port_diff_buffers]	Output differential buffers when writing in -port mode
[-write_all_overrides]	Write parameter overrides on Xilinx primitives even if the same as the default value
[-rename_top]	Replace top module name with custom name e.g. netlist Default: new top module name
[-arch_only]	Write only the architecture, not the entity declaration for the top cell
[-force]	Overwrite existing file
[-include_xilinx_libs]	Include simulation models directly in netlist instead of linking to library
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>file</i>	Which file to write

Categories

[FileIO](#), [Simulation](#)

Description

Write a VHDL netlist of the current design or from a specific cell of the design to the specified file or directory.

The output of this command is a VHDL IEEE 1076.4 VITAL-2000 compliant VHDL file that contains netlist information obtained from the input design files. You can output a complete netlist of the design or specific cell, or output a port list for the design.

Arguments

-cell *arg* - (Optional) Write the VHDL netlist from a specified cell or block level of the design hierarchy. The output VHDL file or files will only include information contained within the specified cell or module.

-mode *arg* - (Optional) The mode to use when writing the VHDL file. By default, the simulation netlist is written for the whole design. Valid mode values are:

- **funcsim** - Output the VHDL netlist to be used as a functional simulation model. The output netlist is not suitable for synthesis. This is the default setting.
- **port** - Outputs only the I/O ports in the entity declaration for the top module.

-lib - (Optional) Create a separate VHDL file for each library used by the design.

Note This option is the opposite of, and replaces the **-nolib** option from prior releases. Previously the default behavior of **write_vhdl** was to output a separate VHDL file for each library used in the design, unless **-nolib** was specified. Now you must specify the **-lib** option to output separate files for each library

-port_diff_buffers - (Optional) Add the differential pair buffers and internal wires associated with those buffers into the output ports list. This argument is only valid when **-mode port** is specified.

-write_all_overrides [true | false] - (Optional) Write parameter overrides in the design to the VHDL output even if the value of the parameter is the same as the defined primitive default value. If the option is false then parameter values which are equivalent to the primitive defaults are not output to the VHDL file. Setting this option to true will not change the result but makes the output netlist more verbose.

-rename_top *arg* - (Optional) Rename the top module in the output as specified. This option only works with **-mode funcsim** to allow the VHDL netlist to plug into top-level simulation test benches.

-arch_only - (Optional) Suppress the entity definition of the top module, and outputs the architecture only. This simplifies the use of the output VHDL netlist with a separate test bench.

-include_xilinx_libs - (Optional) Write the simulation models directly in the output netlist file rather than pointing to the libraries by reference.

-force - (Optional) Overwrite the VHDL files if they already exists.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The filename of the VHDL file to write. If the file name does not have either a .vhd or .vhd1 file extension then the name is assumed to be a directory, and the VHDL file is named after the top module, and is output to the specified directory.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example writes a VHDL simulation netlist file for the whole design to the specified file and path:

```
write_vhdl C:/Data/bft_top.vhd
```

In the following example the entity definition of the top-level module is not output to the VHDL netlist:

```
write_vhdl C:/Data/vhdl_arch_only.vhd -arch_only
```

See Also

[write_verilog](#)

write_xdc

Writes constraints to a Xilinx Design Constraints (XDC) file. The default file extension for a XDC file is .xdc.

Syntax

```
write_xdc [-no_fixed_only] [-constraints arg] [-cell arg] [-sdc]
[-no_tool_comments] [-force] [-exclude_timing] [-quiet] [-verbose]
[file]
```

Returns

Nothing

Usage

Name	Description
[-no_fixed_only]	Export fixed and non-fixed placement (by default only fixed placement will be exported)
[-constraints]	Include constraints that are flagged invalid Values: valid, invalid, all Default: valid
[-cell]	Export placement for this cell.
[-sdc]	Export all timing constraints.
[-no_tool_comments]	Don't write verbose tool generated comments to the xdc when translating from ucf.
[-force]	Overwrite existing file.
[-exclude_timing]	Don't export timing constraints.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[file]	Output constraints to the specified XDC file.

Categories

[FileIO](#)

Description

Writes constraints to a Xilinx Design Constraints file (XDC). The XDC can be exported from the top-level, which is the default, or from a specific hierarchical cell.

This command can be used to create an XDC file from a design with UCF files. All constraints from the active constraint files set will be exported to the XDC, even if they come from multiple files.

Arguments

-no_fixed_only - (Optional) Export both fixed and unfixed placement LOCs to the constraint file being written. By default only the fixed LOCs will be written to the XDC file. Fixed LOCs are associated with user-assigned placements, while unfixed LOCs are associated with tool assigned placements.

-constraints arg - (Optional) Export constraints that are flagged valid, invalid, or all constraints (both valid and invalid). The default behavior is to export only valid constraints to the XDC file. Valid values are VALID, INVALID, or ALL.

-cell arg - (Optional) The name of a hierarchical cell in the current design to export the constraints from. The constraints will be written to the specified XDC file relative to the specified cell.

Note A design must be open when using this option.

-sdc - (Optional) Export only the timing constraints in an SDC format from the current design. Does not export any other defined constraints.

-force - (Optional) Overwrite the XDC file if it already exists.

-quiet - (Optional) Execute the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

-verbose - (Optional) Temporarily override any message limits and return all messages from this command.

Note Message limits can be defined with the **set_msg_config** command.

file - (Required) The filename of the XDC file to write.

Note If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

Examples

The following example writes the valid and invalid constraints, including both fixed and unfixed cells, to the specified file:

```
write_xdc -no_fixed_only -constraints all C:/Data/design.xdc
```

See Also

[read_xdc](#)

xsim

Load a simulation snapshot for simulation and return a simulation object.

Syntax

```
xsim [-maxdeltaid arg] [-nolog] [-onfinish arg] [-onerror arg]
[-tclbatch arg] [-t arg] [-runall] [-R] [-testplusarg args]
[-vcdfile arg] [-vcdunit arg] [-view arg] [-wdb arg] [-tp] [-tl]
[-quiet] [-verbose] snapshot
```

Returns

Current simulation object

Usage

Name	Description
[-maxdeltaid]	Specify the maximum delta number. Will report error if it exceeds maximum simulation loops at the same time Default: 10000
[-nolog]	Ignored (for compatibility with xsim command-line tool)
[-onfinish]	Specify behavior at end of simulation: quit stop Default: stop
[-onerror]	Specify behavior upon simulation run-time error: quit stop Default: stop
[-tclbatch]	Specify the TCL file for batch mode execution
[-t]	Specify the TCL file for batch mode execution
[-runall]	Run simulation until completion, then quit (does 'run -all; exit')
[-R]	Run simulation until completion, then quit (does 'run -all; exit')
[-testplusarg]	Specify plusargs to be used by \$test\$plusargs and \$value\$plusargs system functions
[-vcdfile]	Specify the vcd output file
[-vcdunit]	Specify the vcd output unit. Default is the same as the engine precision unit
[-view]	Open a wave configuration file. This switch should be used together with -gui switch
[-wdb]	Specify the waveform database output file
[-tp]	Enable printing of hierarchical names of process being executed
[-tl]	Enable printing of file name and line number of statements being executed.

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<i>snapshot</i>	The name of design snapshot

Categories

[Simulation](#)

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

<http://www.xilinx.com/support>

For a glossary of technical terms used in Xilinx documentation, see:

<http://www.xilinx.com/company/terms.htm>

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

Vivado Design Suite 2012.2 Documentation:

www.xilinx.com/support/documentation/dt_vivado2013-1.htm

Tcl Developer Xchange

Tcl reference material is available on the Internet. Xilinx recommends the Tcl Developer Xchange, which maintains the open source code base for Tcl, and is located at:

<http://www.tcl.tk>

An introductory tutorial is available at:

<http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html>

About SDC

Synopsys Design Constraints (SDC) is an accepted industry standard for communicating design intent to tools, particularly for timing analysis. A reference copy of the SDC specification is available from Synopsys by registering for the TAP-in program at:

<http://www.synopsys.com/Community/Interoperability/Pages/TapinSDC.aspx>